Global 16-bit Development System
Screen Presentation Manual
Version 8.1

TABLE OF CONTENTS

APPENDICES

1. Introduction

The Global Screen Presentation Manual is a complete guide to screen handling under the Global System Manager. The Global System Manager provides facilities for simple console handling as well as a quick, flexible method of developing interactive application programs written using Global Cobol.

Throughout this document we assume you to be familiar with Global Cobol and the System Manager as described in the Global Cobol Language Manual and the Global Operating and Utilities Manuals. For some of the more advanced features you need to be familiar with topics discussed in the Global Cobol File Management Manual; where this is the case the appropriate reference is given.

Chapter two of this manual describes the statements provided for the simple management of the console; it shows how you can display information on the screen and accept input from the operator. It also describes how you may make use of your screen's ability to position the cursor anywhere within the display to provide more complex displays.

Chapter 3 described various aspects of the Global Screen Formatting system. Formats for a screen display can be defined independently of the program that processes them by using the $FORM command. Special Global Cobol statements, MAPOUT and MAPIN, enable you to display or accept an entire screen format in a single operation. The cost of application development is therefore significantly reduced and maintenance is simplified since screen formats can be amended with a minimum of effort.

Section 3.3, "Data Entry Conventions", describes the control functions available when the operator is inputting data. You will probably wish to reproduce this information along with the other operating instructions you provide for your own applications. Section 3.5, which summarises the restrictions you must bear in mind during design and development, is also particularly important. We suggest you skim through it on a first reading and then review it once you are familiar with the whole concept.

Chapter 4 describes how to run the $FORM command. In order to make the explanation as concise as possible information already introduced in Chapter 3 is not repeated. You must be familiar with section 3.4, "Creating Maps using $FORM", before you attempt to use the command.

Chapter 5 describes the additional Global Cobol statements provided as part of Screen Presentation. The MD, RECORD, VARIANT and AREAS statements are used to establish a Map Definition (MD); the MAPOUT, MAPIN and MAPCLEAR statements are used to display, accept, and blank out fields respectively. Programs using MAPOUT and MAPIN can operate on any display terminal with a screen size of at least 14 lines of 40 characters and cursor positioning capability. The $FORM command itself requires a terminal with extended control functions as well as cursor positioning.

Chapter 6 describes some more advanced features. It explains how to write your own validation routines to supplement the basic checking performed by the system itself; how to use variants to select or exclude particular fields from any operation; how to make the maps separate load modules so that they can be overlayed; and how to establish the data areas associated with the map at run-time.

Chapter 7 describes the system variables and system routines that allow you to interface with the Global System Manager in a number of ways to provide special facilities for complicated or intricate screen handling.

Chapter 8 explains how to use the Global AutoGuide system, either to set up a Help Guide for use by the presentation manager or a demonstration guide (for use as a sale aid, for example).

Chapter 9 describes the programming interface to the menu handler, $MH, which is a programming facility available with System Manager V6.2 and later. It allows you to define menus independently of your application program, and to vary the presentation format without affecting the rest of your product. The documentation of menu definition and amendment is to be found in the Utilities manual, and you should read this before attempting to program a menu interface into an application.

Appendix A contains listings of ORDENT, an example order entry program, and its copy library and maps. The example files involved are installed on the system volume containing the $FORM command as part of an extended or advanced Global development system. You can run ORDENT to see Screen Presentation at work on a real (although somewhat simplified) application. The appendix also contains design notes used in creating the program: you may wish to employ a similar technique when developing your own applications. As part of an exercise you modify the screen formats used by ORDENT, without having to recompile the program itself. You should note that the screen shown in chapter 2 was produced in the course of developing this example, so given a suitable terminal you will be able to create it yourself, should you so wish.

Appendix B contains information allowing you to make storage estimates when planning an application using Global Screen Subroutines.

Appendix C gives an overview of the Screen Formatting System, showing how the various components inter-relate.

Appendix D describes the Global Screen Formatting standards.

2. Simple Console Handling

2.1 Introduction

Global Cobol provides 7 statements to support elementary console handling at the field level. The DISPLAY and DISPLAY...LINE statements, the display operations, output a single field no more than 80 characters in length. ACCEPT and ACCEPT...LINE, the accept operations, input a field of up to 80 characters. The BELL statement sounds the console bell, clears the type-ahead buffer and terminates job management if it is in control: the statement is used to signal errors. CLEAR and SCROLL, respectively, erase the contents of a display screen and re-establish teletype-compatible working.

The basic DISPLAY and ACCEPT statements described in 2.2 and 2.3 enable your program to treat the console like a teletype. Even if it is actually a sophisticated display terminal the dialogue is developed as though only a simple printer were available: input and output affects only the last line of the screen and once it is full a request to write a new line of information causes the display to scroll and its top line to be lost. The more powerful DISPLAY...LINE and ACCEPT...LINE statements described later allow you to develop applications using formatted, menu-like, screens, but these statements do require a suitable display terminal, with cursor positioning capability, to be available.

Global Cobol provides a number of advanced console handling features, such as the ability to input and output character strings whose lengths are only known at run-time; the facility for honouring special function requests, which the operator normally signals by keying <CTRL A>, <CTRL B> or <CTRL C>; and the ability to input and output individual bytes, without involving a prompt character, as required for full screen editing. These special features are documented in chapter 7, along with a number of additional system variables which can be used to determine the type of console and its dimensions.

2.2 The DISPLAY Statement

Data can be displayed on the next line or at the current character position by coding the statement:

```
DISPLAY A [SAMELINE]
```

If the SAMELINE phrase is omitted output starts at character position one of a new line. If SAMELINE is present output continues from the current character position.

You may code DISPLAY SPACE or DISPLAY SPACES to leave a single blank line and then position output at character position one of a new line. Note, however, that this is the only way in which the figurative constants SPACE or SPACES can be used in conjunction with a console I/O statement.

The item A may be a simple variable, indexed variable or a literal. If it is a character variable or literal it will be output unchanged. If it is computational or display numeric it will be converted to a standard numeric string as described in Section 2.1.6 of the Global Language Manual before output. The character currently being used as a decimal point is held in the system variable $$DECP.

The conversion that takes place is as though the single DISPLAY statement had been replaced by:

```
        MOVE A TO B
        DISPLAY B [SAMELINE]
```

where B is display numeric with the same picture clause as A (minus the COMP phrase).

Overflow can take place when attempting to display a computational item, just as it can during a computational to display numeric move. You may check for it by coding an ON OVERFLOW statement (Section 4.6 of the Global Cobol Language Manual) as the statement immediately following the DISPLAY statement. If no such ON OVERFLOW statement is coded and overflow occurs the program will be terminated with an error.

If a DISPLAY statement suffers overflow it is suppressed and the current character position of the console remains unchanged.

The maximum size of character item that can be output by a single DISPLAY statement is 80 bytes.

The effect of displaying control characters (#01 to #1F) and characters with the senior bit set (#80 to #FF) depends on the type of terminal being used, and should be avoided in portable programs. However, it is valid to display the characters #80 to #8F on systems running under V6.0 onwards as these will appear as graphics characters (see section 7.3.4).

2.3 The ACCEPT Statement

Data may be input from the current character position or next line by coding the statement:

```
        ACCEPT A [NEWLINE] [NULL ...]
```

The first action of ACCEPT is to output the prompt character (the default for this is a colon (:), but this may be altered using $$PROM (see 7.1.5)). This appears in the current character position of the current line unless the NEWLINE phrase is supplied, in which case the colon appears in character position 1 of the next line.

Following the prompt, the operator must key zero or more characters terminated by an end of field code, usually the RETURN key. This code, which serves to delimit the input, appears on the console as a space. It is never passed to the Global Cobol application program. However, if the NULL clause is present in the ACCEPT statement, a null string, consisting of just the end of field code, is valid input.

Providing the input is not null, the System Manager performs standard validation and editing depending on the data type of the variable A:

- If A is computational the input is checked to be a valid numeric string agreeing with A's picture clause, and then converted to binary and placed in A;

- If A is display numeric the input is checked to be a valid numeric string agreeing with A's picture clause and then converted to the standard form and placed in A;

- If A is character the length of the input is checked, an error occurring if it is greater than the length of A. If the input is shorter than A it is padded with rightmost ASCII blanks during the move into A.

For computational and display numeric fields the phrase "agreeing with A's picture clause" means that: if A is positive, no sign is allowed and if A is an integer, no decimal point may be keyed. If A is defined as a decimal fraction by a picture clause of the form:

```
PIC (p,q) PIC S(p,q)
PIC (p,q) COMP PIC S(p,q) COMP
```

then the operator will be allowed to key a decimal point, followed by zero or more digits of the decimal fraction. However, if he keys more than q such digits, the extra digits will simply be ignored: truncation, rather than rounding, being the rule in this case.

If an error is detected the System Manager displays a standard error prompt and the operator must then re-submit the input. The application program will regain control, at the statement following the ACCEPT, only when the input is correct.

The maximum size of character item that can be input by a single ACCEPT statement is 80 bytes.

2.4 The NULL Clause

If the operator's very first key-stroke is the end of field code, or a special terminator such as <CTRL A>, this will be treated as an error, and the operator will be prompted again for valid input, unless the NULL clause is coded. When this is the case the operand, A, will remain unchanged and the statement or statements introduced by the NULL clause will be executed.

An ACCEPT with a NULL clause can function as the initial conditional statement of a format 1 conditional, or begin a format 2 conditional, as described in section 4.6 of the Global Cobol Language Manual. For example:-

```
START.     MOVE 0 TO SUM
GET.       ACCEPT INPUT NULL
                DISPLAY "TOTAL IS "
                DISPLAY SUM SAMELINE
                GO TO START
           ELSE
                ADD INPUT TO SUM
                GO TO GET
           END
           DISPLAY "KEY NAME OF NEXT FILE TO PROCESS"
           ACCEPT FNAME NULL STOP RUN
```

The first of these sums the numbers keyed by the operator until he or she replies with the null string (i.e. <CR>), at which time the message:

```
TOTAL IS sum
```

is output and the process is repeated. The second example simply ends the current job if the operator replies <CR> to the prompt:

```
KEY NAME OF NEXT FILE TO PROCESS:
```

2.5 The BELL Statement

The BELL statement can be used to indicate that an input error has occurred by sounding the console's audible alarm if it has one. BELL also flushes the type-ahead buffer, since this cannot now contain valid information anticipating future prompts. If job management is in control BELL will cause it to be terminated when the next accept operation takes place. All displays following the BELL statement will bypass job management so that they appear directly on the console. Therefore, when using the statement, code it immediately the error is detected before outputting any explanatory messages. For example:-

```
AA100.
        DISPLAY "KEY DEPARTMENT NUMBER"
        ACCEPT DEPT
        IF DEPT > MAXDEP
                BELL
                DISPLAY "NO MORE THAN "
                DISPLAY MAXDEP SAMELINE
                DISPLAY "DEPARTMENTS ARE SUPPORTED"
                GO TO AA100
        END
```

BELL should also be coded before any explanatory message describing the reason for a program being terminated in error. For example:

```
        IF ZMAX > 1000
                BELL
                DISPLAY "PROGRAM TABLE CAPACITY EXCEEDED"
                STOP RUN
        END
```

As well as sounding the alarm, BELL sets a flag which causes a comma prompt to be inserted into the dialogue so that the message can be read if the job producing it runs under the control of a supervisor program like $MH (as explained in the Global System Subroutines Manual). If the BELL statement is omitted the supervisor program will gain control immediately the STOP RUN is executed and if, like $MH, it begins by clearing the screen, the explanatory message will disappear before it can be read.

There is no problem in reading the final message from a program when it runs under control of the System Manager, rather than a special supervisor program, since the System Manager always employs teletype-compatible working, and does not clear the screen. However, for flexibility, you should execute BELL before any terminating message so that any code you develop can run satisfactorily under a supervisor program.

Note: If you want the console bell to sound for reasons other than errors, you should display the bell character, for example:

```
        DISPLAY #07 SAMELINE                    * SOUND BELL
```

2.6 Formatted Display Working

For formatted working the lines of the screen are numbered 1, 2, 3... etc, from top to bottom and the columns are 1, 2, 3 ...etc, from left to right. Any character position can therefore be referred to by its line and column co-ordinate, and the character in the top left-hand corner of the screen is 1,1.

The line at the foot of the screen is known as the base line, and is reserved for system and error messages. The remainder of the screen, excluding the base line, is the formatted area, which can be directly addressed by means of DISPLAY...LINE and ACCEPT...LINE statements:-

```
DISPLAY A LINE Y [COL X]
ACCEPT A LINE Y [COL X] [NULL...]
```

These allow you to display a message starting at a given x,y co-ordinate of the formatted area, and to accept input from the right of a prompt character output at a given x,y position.

If you are writing a portable program, to run on a range of terminals, you should check the screen dimensions, given by system variables $$LINE and $$WIDE, to ensure that the screen is large enough for your program. If you do not, and attempt to position at an x,y co-ordinate which is not on the screen, this will lead to unpredictable results.

The CLEAR statement causes the entire contents of the screen to be erased. In addition it informs the System Manager that the program is now operating in formatted mode and that any teletype-compatible DISPLAY or ACCEPT must be treated specially, as explained below. The CLEAR statement must be issued before the first DISPLAY...LINE or ACCEPT...LINE statement used by a program.

A teletype-compatible DISPLAY or ACCEPT statement issued when the System Manager is operating in formatted mode causes input/output to take place on the base line. Such DISPLAY statements are used to write error messages without interfering with the formatted part of the screen.

The System Manager returns automatically to teletype mode at the end of each job. In addition you can use the SCROLL statement to explicitly request to return to teletype mode working.

2.7 The CLEAR Statement

The CLEAR statement must be executed before the first ACCEPT...LINE or DISPLAY...LINE statement of a program. It is coded simply as:

```
CLEAR
```

The normal action is to cause the contents of the screen to be erased and formatted mode to be established. However, if the terminal the operator is using is not a display with cursor control, the System Manager will signal exception condition 1, inappropriate terminal type, to indicate that the terminal is unsuitable for formatted screen working and that the system must remain in teletype mode. To handle this condition you should code an ON EXCEPTION statement immediately following the CLEAR statement. If you do not, and the condition arises, your program will be terminated in error.

A CLEAR statement issued when the program is already operating in formatted mode simply causes the contents of the screen to be erased.

2.8 The DISPLAY...LINE Statement

The DISPLAY...LINE statement allows a message of up to 80 characters to be output starting at a particular x,y co-ordinate of the formatted area of the screen. It is coded:-

```
DISPLAY A LINE Y [COL X]
```

The statement is only valid when a previous CLEAR statement has established formatted mode. An attempt to use such a DISPLAY in teletype mode will cause the program to be terminated in error.

Unless overflow occurs, the effect of the DISPLAY...LINE statement coded above is to position the cursor at line Y, column X and then output the quantity A just as if, following the cursor positioning, a DISPLAY A SAMELINE statement had been issued.

If the COL clause is omitted the cursor will be positioned in the column established by the previous ACCEPT or DISPLAY with a COL clause, or column 1 if there was none.

Y (and X if it is defined) must be integer literals or the names of PIC 9(4) COMP variables defining a character location within the formatted area of the screen. If the position specified is not a valid location this will lead to unpredictable results.

The effect of displaying control characters (#01 to #1F) and characters with the senior bit set (#80 to #FF) depends on the type of terminal being used, and should be avoided in portable programs. In some countries lower case characters may be used for a different alphabet. On videotex terminals any control characters displayed must appear at the start of the display, before any normal graphic characters.

If A is computational, overflow may occur, just as it can during a DISPLAY statement. You may check for it in the same way by coding an ON OVERFLOW statement. If no such ON OVERFLOW statement is coded and overflow occurs the program will be terminated in error.

2.9 The ACCEPT...LINE Statement

The ACCEPT...LINE statement allows an input of up to 80 characters to be received following a prompt character positioned at a particular x,y co-ordinate of the formatted area of the screen. It is coded:

```
ACCEPT A LINE Y [COL X] [NULL...]
```

The statement is only valid when a previous CLEAR statement has established formatted mode. An attempt to use such an ACCEPT in teletype mode will cause your program to be terminated in error.

The effect of the ACCEPT...LINE statement coded above is to erase the input area, output the prompt character at line Y, column X, and then input the quantity A just as if, following the erasure and cursor positioning, an ordinary ACCEPT A [NULL...] statement had been issued.

If the COL clause is omitted the prompt character will be output in the column established by the previous ACCEPT or DISPLAY with a COL clause, or column 1 if there was none.

Y (and X if it is defined) must be integer literals or the names of PIC 9(4) COMP variables defining a character location within the formatted area of the screen. If the position specified is not a valid location this will lead to unpredictable results.

The optional NULL clause is constructed and handled in a similar way as for an ordinary ACCEPT statement. However, when the NULL clause is coded the ACCEPT will place the current value of the field on the screen as a default, rather than erasing the area with spaces. The default may be edited (using the Field Editing keys) or replaced by typing a new value, in which case the default value is erased.

You should note that before outputting the prompt character, ACCEPT...LINE erases an input area immediately to the right of the prompt, by writing either blanks to it or the default reply supplied in the field. This is to clear any out-of-date input that may be displayed on the screen from a previous transaction. In designing a screen format it is obviously important to know how large each input area will be so that the displayed headings can be positioned accordingly.

The length of the input area is determined from the input variable, A. If A is character or display numeric, then the input area length, in characters, is simply the size of A in bytes. If A is computational the input area length is the size in bytes of the equivalent display numeric variable. For example, consider a PIC S9(4) COMP variable. Section 3.3.2 indicates that the equivalent PIC S9(4) display numeric variable occupies 5 bytes: hence a five character input area following the prompt character will be erased prior to accepting a PIC S9(4) COMP variable.

Note: System variable $$AC-5 can be used to suppress the display of a default value by ACCEPT... LINE.

2.10 The SCROLL Statement

When issued in formatted mode the SCROLL statement, coded simply as:

```
SCROLL
```

causes teletype mode to be re-established and the cursor to be positioned at the bottom left-hand corner of the screen. This will result in the next DISPLAY A or ACCEPT A NEWLINE statement scrolling in the normal way.

If the SCROLL statement is executed and the System Manager is already operating in teletype mode the statement has no effect.

2.11 ACCEPT and DISPLAY Statements in Formatted Mode

The teletype-compatible forms of the ACCEPT and DISPLAY statements allow access to the base line when the System Manager is operating in formatted mode, providing certain conventions are obeyed.

The most important rule is that the first teletype-compatible
statement executed after a CLEAR, DISPLAY...LINE or ACCEPT...LINE
statement must be DISPLAY or ACCEPT...NEWLINE. In other words

the teletype-compatible output must be directed to a new line to cause the System Manager to position the cursor at the start of the base line. If your first teletype-compatible statement does not do this (ie is a DISPLAY with the SAMELINE option, or a normal ACCEPT) the cursor will remain undisturbed, somewhere within the formatted area of the screen, and the subsequent I/O will damage whatever menu you have established.

You may use a DISPLAY SAMELINE or ACCEPT once a new line operation has positioned the cursor on the base line. However, you must be careful not to write more characters than are allowed by the line width of the screen otherwise scrolling, which will disturb the formatted area, may occur. You should note that, in order to ensure that the information on the base line is always current, the System Manager clears it immediately before the message from a DISPLAY, or prompt character belonging to an ACCEPT NEWLINE is output, and it is also erased whenever an ACCEPT...LINE statement completes.

The main use of the base line is for error messages following
++++++++++++++
application validation checks. For example, suppose an ACCEPT...LINE statement is used to input an account number and then a check digit test is applied. If the test fails the statements:-

```
DISPLAY "ACCOUNT NUMBER "
DISPLAY ACNUM SAMELINE
DISPLAY " DOES NOT CHECK" SAMELINE
```

cause an error message such as:-

```
ACCOUNT NUMBER 12345678 DOES NOT CHECK
```

to appear on the base line. Control then returns to the ACCEPT...LINE statement to obtain new, hopefully correct, input. When this is received the error message is automatically erased, as explained above.

You can even write a short report to the base line by issuing a number of consecutive DISPLAY statements. System Manager outputs a special comma prompt to the right of the information it has displayed if it detects that a new DISPLAY is about to overwrite the base line and there has been no intervening ACCEPT to allow the operator to read its previous contents.

For example, the statements:-

```
DISPLAY "ACCOUNT NUMBER "
DISPLAY ACNUM SAMELINE
DISPLAY " REFERENCE CONDITION PREVAILS" SAMELINE
DISPLAY "CREDIT LIMIT "
DISPLAY CRLIM SAMELINE
DISPLAY " EXCEEDED" SAMELINE
```

would first cause a message such as:-

```
ACCOUNT NUMBER 12345678 REFERENCE CONDITION PREVAILS,
```

to appear. When the operator presses the end of field key in response to the comma the base line is overwritten by the continuation message:-

```
CREDIT LIMIT 1750 EXCEEDED
```

2.12 Example

Figure 2.12 shows a fragment of a program which makes use of the formatted display facility for a simple account posting application. The statements concerned with file management (FD, ASSIGN, OPEN, READ and CALL FILE$) have not yet been explained, but it is not really necessary to understand them to follow the example. Should you have difficulty, please return to this example after reading the appropriate section in the File Management Manual.

When entered the program is operating in teletype mode and therefore the DISPLAY statement and following call on FILE$ cause the last scrolled line to appear as:-

```
SPECIFY MASTER FILE NAME:ACMASTER UNIT:100
-------- ---
```

where the operator's replies are underlined. Once the operator keys <CR> following the last 0 of 100 the ACMASTER file is opened, then the statements marked (A) and (B) cause the screen to be cleared and a menu of the form shown in the screen below to appear:

[PHOTO]

The cursor is positioned for the operator to key the account number to the right of the prompt character which appears in column 40. When there are no more accounts to post the operator need only key <CR> for statement (B) to route control to ENDJOB where normal end of job processing, such as the closing of files, takes place.

When an account number is submitted the program checks to see whether a record for it is present on the account master file and if there is no such record available uses a teletype-compatible DISPLAY statement, (C), to output the warning message:-

```
ACCOUNT NOT ON MASTER FILE
```

PROGRAM EXAMP

```
DATA DIVISION
*
* ACCOUNT MASTER FILE FD AND RECORD DEFINITION
*
FD ACFILE ORGANISATION INDEXED-SEQUENTIAL
ASSIGN TO UNIT "?" FILE "MASTER"
01 ACREC
 02 ACDSID PIC X(2)
 02 ACLINK PIC X(2)
 02 ACNUM PIC X(8) * KEY
.
.
.
*
77 ZAMT PIC 9(6,2) * AMOUNT
PROCEDURE DIVISION
SECTION MAIN
 DISPLAY "SPECIFY MASTER FILE NAME"
 CALL FILE$ USING ACFILE
 OPEN OLD ACFILE
 CLEAR (A)
 DISPLAY "ACCOUNT POSTING TRANSACTION" LINE 1 COL 20 (A)
 DISPLAY "ACCOUNT NUMBER" LINE 3 COL 1 (A)
 DISPLAY "AMOUNT" LINE 4 (A)
 DISPLAY "TRANSACTION TYPE" LINE 5 (A)
 .
 . etc, etc.
 .
* THE PROCESSING OF EACH INDIVIDUAL TRANSACTION STARTS HERE
TRANS.
 ACCEPT ACNUM LINE 3 COL 40 NULL GO TO ENDJOB (B)
 READ ACFILE INTO ACREC
 ON EXCEPTION
 DISPLAY "ACCOUNT NOT ON MASTER FILE" (C)
 GO TO TRANS
 END
 ACCEPT ZAMT LINE 4 (D)
 .
 . Input the other fields, validate and update ACRES

 DISPLAY "ACCOUNT NUMBER " (E)
 DISPLAY ACNUM SAMELINE (E)
 DISPLAY " POSTED" SAMELINE (E)
 GO TO TRANS
*
ENDJOB.
 . end of job processing, etc.
 .
 .
ENDPROG
```

Figure 2.12 - A program using the formatted display facility

Because a teletype DISPLAY, rather than DISPLAY...LINE has been used, this message appears on the base line and does not interfere with the formatted information displayed on the application area of the screen. The program then returns control to statement (B) so that the operator can correct his keying error. The warning message will disappear from the base line as soon as new input is supplied.

Once a good account number is received the program continues at statement (D) by accepting the amount, transaction type, and so on. In this application only one field is input per line and a neat appearance is achieved by omitting the COL phrase from all but the first ACCEPT...LINE statement, (B), so that all the prompt characters are vertically aligned.

After all the fields are input and validated the account record is updated and the confirmatory message:-

```
ACCOUNT NUMBER xxxxxxxx POSTED
////////
```

is displayed on the baseline by the teletype-compatible DISPLAY statements marked (E). The program then passes control to TRANS to process the next transaction.

At this stage, once the ACCEPT...LINE at (B) is initiated, the display might appear as:-

[PHOTO]

The cursor is positioned to accept the next account number and the previous input has been erased. The message on the base line will disappear once the operator keys the new account number.

The quantities 1413.17, PL etc. are inputs supplied for the previous transaction, but they will be erased as soon as the cursor is positioned to accept the new amount and transaction type.

3. Concepts and Terminology of Screen Formatting

```
| | | |
--------------- ---------------
| | | |
|DATA DIVISION| | CONSOLE |
 +++++++++++++ +++++++
| | | |
| | | | | |
 ----------
| | ------------->| |------------>| |
|Output fields| |Output fields | MAPOUT | DISPLAY outputs LINE... |
 ++++++
| | |Update fields | | DISPLAY updates LINE... |
 ----------
| | | | DISPLAY text LINE... |
 --------------
| | | | | | | |
| | | | | | | |
| | | | MAP | | | |
 +++
| | | | | | | |
| | | | Text | | | |
| | | | Positioning|---> | |
| | | | Picture | | |
 -
|Update fields| | Location | | |
 -
| | | | Validation |---> | |
| | | | | | | |
 - - - - - - -
| | | | | | | |
| | | |[VALIDATION | | | |
| | | | ROUTINE] | | | |
| | | | | | | |
 --------------
| | | | | |
 ----------
| | |Update fields | | ACCEPT updates LINE... |
|Input fields | |Input fields | MAPIN | ACCEPT inputs LINE... |
 +++++
| | <-------------| |<------------| |
 ----------
| | | |
| | | |
--------------- ---------------
```

Figure 3.1 - MAPOUT and MAPIN operation
++

3.1 Introduction
+++ ++++++++++++

Applications developed using the basic console I/O facilities described in chapter 2 of this manual interact with the operator using a series of prompts, each of which requires a separate DISPLAY and ACCEPT statement. If formatted screen working is involved line and column positioning information must be provided in DISPLAY...LINE and ACCEPT...LINE statements. Normally many lines of code are involved, and some or all of them will need to be changed if the screen layout is altered in any way.

By using Screen Formatting you can code a single MAPOUT statement to display all the information contained on a formatted screen, and then accept all the subsequent input with just one MAPIN statement. The program using these statements does not need to concern itself with the positioning of the information, or the constant text used to display explanatory headings. In addition basic validity checking of input (which can be supplemented by your own special validation routine) takes place automatically during MAPIN processing. Figure 3.1, which indicates schematic-ally how MAPOUT and MAPIN operate, shows how the statements obtain text, positioning and other information they require from a map, an independently compiled module consisting only of
+++
tables. The program using MAPOUT and MAPIN need only supply the variable information associated with each screen: the output, input and update fields. By using the information in the map together with these variables the MAPOUT statement executes the appropriate DISPLAY...LINE operations to create the screen format, and MAPIN issues ACCEPT...LINE operations to obtain the operator input. In addition a special option of the MAPOUT statement is available to allow you to log the information you have displayed on a print file.

Every field accepted by MAPIN is checked to conform with its picture clause just as in conventional ACCEPT...LINE processing. There are also a number of extra tests provided by Screen Formatting which you can specify when the map is built. For example, you can insist that all characters keyed are alphabetic. You can supplement these basic reasonableness tests by providing your own validation routine to be called when selected fields are processed by MAPIN. In the figure, in the interests of simplicity, this routine is shown as part of the map. In fact, the validation routine is a separate compilation linked with the maps which use it, and in practice it will usually be part of the program which issues the MAPIN statement.

The MAPCLEAR statement allows selected fields to be cleared to spaces. This avoids the necessity to clear and re-display the whole screen when you are using the same format repeatedly to input data, and facilitates the use of two or more separate maps at the same time.

A program can use a number of different maps, so that it can handle various types of transaction appropriately. It is possible to adjust the layout, checking, and in certain circumstances the content of individual screens, without the need to re-compile programs. This is one of the major benefits of the Screen Formatting System since it enables rapid "human engineering" of an application once it is functionally correct.

3.2 Text, Fields and Records
+++ ++++++++++++++++++++++++

For the purposes of Screen Formatting, console information is considered to consist of text, output, input and update fields. Furthermore, contiguous lines of similar information can be grouped into records. The photograph below, taken whilst an operator was part of the way through keying information in response to the order data entry screen produced by the sample program, is used in explaining these terms:-

```
..............................................
. .
. .
. .
. .
. .
. .
. .
. .
. PHOTO .
. .
. .
. .
. .
. .
. .
. .
. .
. .
. .
. .
..............................................
```

3.2.1 Text
+++++ ++++

Text consists of constant explanatory information used in building a screen menu or framework, as well as boxes or lines which are used to enhance the appearance of the screen. It cannot be altered by the operator. The text in the example consists of the phrases:-

ACCOUNT ADDRESS CUSTOMER LINE
PRODUCT DESCRIPTION QTY UNIT-PRICE VALUE

and the lines and boxes.

3.2.2 Output Fields
+++++ +++++++++++++

Output fields can also supply explanatory information, which cannot be altered by the operator: however, unlike the text, the values displayed are under the direct control of the application program executing the MAPOUT statement. The account number (11111111), customer name (J. SQUIRES AND CO. LTD), line number (1), and line value (158.40) are output fields in the example.

3.2.3 Input Fields
+++++ ++++++++++++
The input fields are normally keyed by the operator, although in certain cases he or she may be allowed to skip them, in which case default values will be established. These fields are set to spaces when the framework is displayed by MAPOUT. In the example, the product description, quantity and unit price (eg RED DOORS, 3, 52.80) for order line 1 have already been keyed.

3.2.4 Update Fields
+++++ +++++++++++++
The update fields are hybrid. They appear as output fields when the framework is created by MAPOUT, but they may optionally be over keyed when a MAPIN is in control. It is also possible to skip the keying of update fields, in which case they retain their original values. In the example, the three lines of the despatch address:-

GRENVILLE HOUSE
13 SEAVIEW ROAD
OXFORD XJ4 7JK

are each update fields which can be individually modified if necessary.

3.2.5 Records
+++++ +++++++
The example map contains 9 records of one line each,
+++++++
corresponding to a maximum of 9 order lines which may be keyed by the operator. In the photograph the operator has keyed the product description, quantity and unit price information for the first order line and is about to start on the second. Each of these records contains two output fields - the record number (which appears in the LINE column) and the line value - together with the product description, quantity and unit price, which are actually update fields to allow for later amendments. Special forms of the MAPOUT and MAPIN statements are provided to allow the records of a map to be processed individually.

3.2.6 Field Definition
+++++ ++++++++++++++++
Every field participating in a mapping operation must be defined as an elementary item (ie one with a picture clause) in a copy book describing the contiguous area of storage containing the field. The same copy book definitions must be used by application programs executing MAPOUT and MAPIN statements involving the fields, and the $FORM command used to build the map, in order that the correct picture and location information is set up. Any field belonging to a record is normally defined to be multi-valued. That is, it either belongs to a repeating

++++++++++++
group or is declared as an elementary item with an OCCURS clause. The record number then determines the particular occurrence of the field to be used. You may use a multi-valued field which is not part of a record (eg the three address lines in the example) but in this case you must indicate which occurrence is to be used when the map is built.

Note that maps need not necessarily possess records, nor indeed need they contain any output, input or update fields. It is quite possible to create a map consisting only of text; a typical example would be the map of a "help" screen explaining how to use an application.

```
| | | | |
----------------------------------------------------------------
| | | | |
|KEYSTROKE| SUPPLIED |FUNCTION| DESCRIPTION (ASSUMING CORRECT USAGE)|
 +++++++++ ++++++++ ++++++++ +++++++++++++++++++++++++++++++++++++
| | | | |
----------------------------------------------------------------
| | | | |
| <CR> | after | END OF | Terminates the current input, |
| | keying a | FIELD | causing the value the operator has |
| | value | | keyed to be accepted and validated. |
| | | | The cursor is then positioned at the|
| | | | start of the next incoming field. |
| | | | |
| | as the | SKIP | The current field is skipped. (An |
| | first | | update field will retain its |
| | key | | existing value and an input field |
| | stroke | | will be assigned a default value.) |
| | | | The cursor is then positioned at the|
| | | | start of the next incoming field. |
| | | | |
| | | | |
|<LINE- | after |RE-ENTER| The cursor is re-positioned at the |
| FEED> | keying | | start of the input area to accept |
| or | a value | | a correction. |
| CURSOR | | | |
| UP | as the |BACKTAB | The cursor is positioned at the |
| | first | | beginning of the previous input |
| | key | | area. |
| | stroke | | |
| | | | |
| <CTRL A>| at any |SKIP ALL| All remaining incoming fields on |
| | time | | the map are skipped. If keyed as |
| | during | | the first character of a field, |
| | the | | that field is also skipped, |
| | keying of| | otherwise the supplied value is |
| | a field | | used. (Update fields retain their |
| | | | existing values when skipped, input |
| | | | fields are assigned default values |
| | | | if available.) |
| | | | |
| <CTRL B>| " " | DELETE | If the current field is an update |
| | | | field, it is deleted and its default|
| | | | value is established and displayed. |
| | | | DELETE issued for an input field is |
| | | | treated like SKIP. In any case the |
| | | | cursor is then positioned at the |
| | | | start of the next incoming field. |
| | | | |
| <CTRL D>| " " | HOME | The cursor is positioned at the |
| | | | beginning of the first input field. |
| | | | |
| <ESCAPE>| at any |VARIABLE| Depends on the value of $$ESC as |
| | point | | described in 7.1.3. |
| | on the | | |
| | screen | | |
| | | | |
-----------------------------------------------------------------
```

Table 3.3 - Summary of Data Entry Conventions during a MAPIN Operation

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3.3 Data Entry Conventions
+++ ++++++++++++++++++++++

A MAPOUT statement displays the screen framework line by line, ie
in the sequence in which it would normally be read. Input and
update fields are accepted by MAPIN in the same order, starting
at the top left-hand corner and working downwards, unless
sequence numbers define otherwise. Table 3.3, however, shows
that simple cursor control features are provided which can allow
the operator to change this sequence somewhat, by using the
LINE-FEED key to back tab to the previous field, or re-enter the
current field. Note that you may use <RUBOUT> to correct keying
------
errors in the normal way.

The table indicates how Screen Formatting interprets <CR>,
--
<LINE-FEED> (or <CURSOR UP>), <CTRL A>, <CTRL B> and <CTRL D>
--------- ---------- ------ ------ ------
when a MAPIN statement is accepting operator input. The use of
the ESCAPE key is also handled specially, but this is under
program control and is thus application-dependent.

3.3.1 Base Line Messages
+++++ ++++++++++++++++++

The information defined by the map can only occupy the formatted
area of the screen, deliberately excluding the base line which is
reserved for error messages and special prompts. In particular,
if the operator input fails the basic reasonableness checks
applied by Screen Formatting, the console alarm is sounded and an
error message of the form:-

INVALID * xx...xx
///////

appears on the base line. The cursor is positioned at the start
of the input area so that the operator can supply new, hopefully
correct, input. The quantity xx...xx in this message consists of
///////
the previous erroneous input, up to 20 characters of which are
re-displayed for visual checking. If you supply your own
validation routine it can write its own error messages to the
base line.

3.3.2 Skipping Input and Update Fields
+++++ ++++++++++++++++++++++++++++++++

By keying either <CR> or <CTRL A> as the first key stroke of a
-- ------
field the operator can indicate that he or she wishes to skip
either just the current field, or all input and update fields yet
to be keyed. An update field can always be skipped, in which
case its current value is retained. An input field can only be

skipped if you indicated that it could be defaulted when the map was built. In this case the field will be set to LOW-VALUES (binary zeros) unless you have provided a validation routine to establish a special default value.

If the operator attempts to skip an input field which was not defaulted, the console alarm sounds and the error message:-

INPUT REQUIRED

appears on the base line. The cursor is re-positioned at its start so that the necessary information can be keyed.

3.3.3 Deleting Update Fields
+++++ ++++++++++++++++++++++
The operator can key <CTRL B> to delete an update field,
------- ++++++
providing you have indicated that it can be defaulted when the
map was built. This causes the field to be set to LOW-VALUES
unless you establish your own special default in a validation
routine. If you have not allowed the field to be defaulted the
INPUT REQUIRED error message appears as described above.

You must be careful when using defaulted fields in an application
because the LOW-VALUES setting may lead to unexpected results.
You should not attempt to DISPLAY a PIC X field containing
LOW-VALUES, or DISPLAY or MOVE a similar display numeric field.
(MAPOUT, however, can display such fields, which appear as
spaces.)

3.4 Creating Maps using $FORM

+++ ++++++++++++++++++++++++

Screen Formatting provides the $FORM command which you use to create or amend maps. Each map is produced as a separate compilation file. So, having run $FORM, you would typically either link the map with the program or programs which require it, or save it in a compilation library.

3.4.1 The Map-id and the Book List

+++++ ++++++++++++++++++++++++++++

During its initial parameterisation phase $FORM asks you to supply the map-id, which is the name of the map you wish to create, and the book list, which is optional. The list, if specified, consists of up to five names of copy books describing the data areas containing the output, input and update fields used by the map. Since as many as five names may be provided, it is possible for a single MAPOUT or MAPIN statement to access fields in up to five different memory areas. Alternatively, in the case of a map consisting only of text, no book list need be supplied since the map contains no information used by the application program. When a book list is provided it is vital that the copy books are the same as those employed by the program to describe the data areas involved, since $FORM will use the information to determine the picture and storage location of each field defined in the map.

3.4.2 Screen Area Dimensions and Options

+++++ +++++++++++++++++++++++++++++++++++

Once the map-id and book list have been specified $FORM asks you for the dimensions of the screen area the map is to occupy. The area can be no larger than the formatted area of the current terminal (ie the total display, excluding the base line) but it may be smaller if the map is being created on one machine to run on another. Once the dimensions are supplied, you answer a number of questions which determine the run-time options:-

(i) Does the map contain records in the first record set (R)? Does the map contain records in the second record set (Q)? If so, how many lines does an individual record occupy? How many records should be contained in an individual column? How wide should an individual column be?

(ii) Is a validation routine to be specified here? If so, the basic reasonableness checks and standard defaults established by Screen Formatting will be supplemented by the routine you supply here. It is advisable, if you are using a validation routine, only to supply the routine here, rather than in the map definition in your program, if the validation routine is in a different source file from the validation routine.

(iii) Are screen overlays to be used? These allow you to edit maps that have overlapped fields; for example, maps that contain several alternative formats for one area of the screen. If you specify this option $FORM only displays variants 0-39 and then one of the decades 40-49, 50-59 etc. You can change the displayed 'overlay' using the O instruction and change the range of variants for overlays 1-6 using the V instruction. Note that this feature is purely to aid editing in $FORM: the system routine has no special support for overlays, but relies on you using the correct table of variants.

(iv) Is field sequencing to be used? If so, you can define the order in which the mapping statements handle the fields.

(v) Is field numbering to be used? If so, you can use special forms of the MAPOUT, MAPCLEAR and MAPIN statements to process numbered fields individually.

(vi) Is field level help required? If so, you can define a help book for each field in the map. You will also be prompted for a default help book to be used when none is specified. If you do not want a default help book key spaces to the default help book prompt.

(vii) Does the map contain variants? If so, mapping statements can specify a variant number to cause particular variants to be selected or excluded.

(viii) Does the map require a validation routine? If so, you must key Y to the validation numbering prompt. The basic reasonableness checks and standard defaults will be supplemented by the routine you supply to the validation routine prompt or through the map definition in your program.

(ix) Is auto-centre required? If so the map area will be automatically centred within the formatted area of the display or within the printed line width. This option is useful when a map built for a small screen (eg a 40-character by 24-line terminal) is to be used on a larger device.

(x) Are colon prompts to be used? If so, a colon is displayed immediately to the left of the current input or update field during MAPIN processing. This option can be used to emphasize the cursor position on hardware where the cursor itself is indistinct.

(ix) Do you want to use a colour map? If so, you can allocate a colour number (1-8) to selected fields when creating or editing the map.

3.4.4 Comments
++++++++++++++

Once you have specified the options $FORM asks if you want to edit the comments referring to the map. If you key Y the screen is cleared and you may key up to 58 lines of descriptive text about the map using the standard text editing facility as described in TXEDT$.

3.4.3 Format Editing
+++++ ++++++++++++++
Once you have specified the options or keyed in your comments,
$FORM clears the display and enters its full screen editing
mode. You then create within the screen area you defined in the
previous step, a picture of the information you expect the
operator to see, using the $FORM editor. If the screen contains
records you should only describe the information belonging to the
first one.

Field definition is also part of the $FORM editing process.
Whenever you want to define a field you use the F editing
instruction and you are then prompted for the detailed attributes
of the field. Table 3.4.4 gives a summary of the details you are
asked to supply.

The name must be the same as the name given to the field in its
++++
copy book. $FORM uses this to determine the picture and,
ultimately, the run-time location of the field. The type
++++
indicates whether the field is employed for output, input or
update purposes. If it is multi-valued, you will also be
prompted for the occurrence or record attribute to be used.
++++++++++

Screen Formatting provides two in-built system variables which
you can use as output fields. $$DATE is a PIC 9(6) COMP field
containing the current date in internal format. Since this field
is automatically assigned the date attribute it will actually be
++++
displayed in the form DD/MM/YY or MM/DD/YY, depending on whether
European or American processing is in force. $$RECNO is a PIC
9(2) COMP field containing the current record number. It can
only be supplied to define an output field which is part of a
record.

If you indicated that field numbering was to be used as a
run-time option, you may supply your own number. You can also
reply 0 to indicate that the field is not to be numbered. Field
-
numbers must be unique. The importance of numbered fields is
that you are able to process them individually using the MAPOUT
and MAPIN statements.

If you supplied a validation routine as a run-time option, you
will be asked to supply a validation code, a number between 1 and
+++++++++++++++
99, whenever an input or update field is processed. This code
will be passed to your routine whenever the operator supplies
input for the field, or attempts to skip or delete it. You use

the code to identify the particular test to be performed by the
routine and the actual field involved. You can also reply 0 or
-
<CR> to indicate that the validation routine is not to be used on
--
a particular field.

You can indicate that selected input and update fields are
defaulted, so that the operator can skip the keying of an input
+++++++++
field, or delete an update field, in which case either a default
of LOW-VALUES will be established by the mapping routine or a
special default value will be set up by your own validation
routine.

There are a number of miscellaneous attributes restricted to
certain types of field. These include: blank when zero and
+++++++++++++++++
number-filled, for certain types of display numeric and
+++++++++++++
computational fields; invisible, typically for passwords; and
+++++++++
date, for PIC 9(6) COMP fields used to hold internal format
++++
dates. Table 3.4.4 specifies all the attributes you can supply
during the field definition phase.

Once the last field, if any, has been defined $FORM creates the map in compilation file form and writes the format listing to the printer to provide hard-copy documentation. Control then returns to the parameterisation phase so that you can create another map, should you so wish.

DETAIL	SUPPLIED FOR	NOTES
NAME	All the fields defined by the map.	The name is either one of $$DATE or $$RECNO, or identifies a field in a copy book in the book list.
OCCURRENCE	Multi-valued fields which are not part of a record.	The occurrence indicates which particular value of the field is to be used. (The record number is used to index fields in records.)
FIELD NUMBER (1-99)	Selected fields of any type.	By supplying its number as a parameter of the MAPOUT, MAPCLEAR or MAPIN statement you can process any numbered field individually.
HELP BOOK NAME	Selected fields of any type, not text fields.	Allowing a help book to be set for a particular field.
VARIANT NUMBER (1-99)	Selected fields of any type, also text fields.	By supplying a variant number in the MD you can select or exclude particular variants.
SEQUENCE NUMBER (1-99)	Selected fields of any type, also text fields.	Re-sequence fields so that mapping statements process fields in a user-defined order instead of top to bottom, left to right.
TYPE	All fields except $$DATE & $$RECNO.	The type indicates whether the field is used for output, input or update.
VALIDATION	Selected fields	This code, together with the operator

```
| CODE (1-99) | of any type, but | input, is passed to your validation |
 +++++++++++
| | not $$DATE or | routine which can perform extra tests|
| | $$RECNO. | or establish a default value. |
| | | |
| ATTRIBUTE the following options are available:- |
 +++++++++ ////////////////////////////////////
| |
| AUTOINPUT | Selected input | The field is accepted once the |
 +++++++++
| (A) | or update fields | required no. of characters has been |
 +++
| | | keyed on input. No need for <CR>. |
| | | |
| DEFAULTED | Selected input or| The operator is allowed to skip a |
 +++++++++
| | update fields. | defaulted input field, or delete a |
 +++
| | | defaulted update field. |
| | | |
| GRAPHICS | Selected lines | If this field is displayed on a non- |
 ++++++++
| DEFAULTED | and boxes | graphics terminal, horizontal lines |
 +++++++++
| (G) | | and junctions with a full horizontal |
 +++
| | | bar are displayed as minus signs. |
| | | |
| SOLID BOX | Selected boxes | The whole area of the box is cleared |
 +++++++++
| (S) | | when displayed. |
 +++
| | | |
------------------------------------------------------------------------
```

Table 3.4.4A - Field Details Supplied during Field Definition
+++

```
| | | |
------------------------------------------------------------------------
| | | |
| DETAIL | SUPPLIED FOR | NOTES |
 ++++++ ++++++++++++ +++++
| | | |
------------------------------------------------------------------------
| | | |
| RECORD | Selected non- | This field is included in record set |
 ++++++
| (R) | indexed fields | 1. |
 +++
| | and non-graphics | |
| | text | |
| | | |
| RECORD | Selected non- | This field is included in record set |
 ++++++
| (Q) | indexed fields | 2. |
 +++
| | and non-graphics | |
| | text | |
| | | |
| USER | | |
 ++++
| CONVERSION | Selected fields | The user supplies his own display |
 ++++++++++
| (U) | with a validation| conversion routine. |
 +++
| | code | |
| | | |
| PRE-ACCEPT | Selected fields | The user supplies a pre-accept |
 ++++++++++
| VALIDATION | with a validation| validation routine. |
 ++++++++++
| (H) | code | |
 +++
| | | |
| BLANK WHEN | Selected numeric | If its value is zero, the field will |
 ++++++++++
| ZERO (B) | fields. | be displayed as spaces. If spaces |
 ++++ +++
| | | is keyed as a reply the field will be|
| | | set to zero. |
| | | |
| NUMBER- | Selected unsigned| On input all digits must be entered. |
 +++++++
| FILLED | integer fields. | On output leading zeros are displayed|
 ++++++
| (N) | | as 0's rather than spaces. |
 +++
| | | |
| INVISIBLE | Selected input | On input the characters of an |
 +++++++++
| (I) | or update fields.| invisible field are cleared after |
 +++
| | | the reply has been accepted. |
| | | |
| TRAILING | Selected output | If there are more than 1 trailing |
 ++++++++
| SPACES (T)| fields | space on a field only 1 will be |
 ++++++++++
| | | displayed |
| | | |
| VALIDATION | Selected input | Fields which fail screen formattings |
 ++++++++++
| FAIL (V) | with validation | internal validation e.g. a letter to |
 ++++++++
| | number | numeric field will still be passed |
| | | through the validation routine |
| | | |
| DATE (D) | Selected | The field is input or output in |
 ++++ +++
| | PIC 9(6) COMP | standard external date format, ie |
| | fields. | DD/MM/YY (Europe) or MM/DD/YY (USA). |
| | | |
| LONG DATE | Selected | The field is input or output in |
 +++++++++
| (L) | PIC 9(6) COMP | long date format, ie DD/MM/CCYY |
 +++
| | fields. | (Europe) or MM/DD/CCYY (USA) where |
| | | CC defaults to 19 if omitted. |
| | | |
------------------------------------------------------------------------
```

Table 3.4.4A - Field Details Supplied during Field Definition

```
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
| | | |
------------------------------------------------------------------------
| | | |
| DETAIL | SUPPLIED FOR | NOTES |
 ++++++ ++++++++++++ +++++
| | | |
------------------------------------------------------------------------
| | | |
| Y/N (Y) | Selected | The field is input and displayed as Y|
 +++ +++
| | PIC 9 COMP | or N. Y corresponds to 1, N to zero.|
| | fields. | |
| |
| The final field attribute you are explicitly prompted for is:- |
| |
| COLOUR (1-8) | Selected fields | On colour screens, displayed in the |
 ++++++++++++
| | of any type, also| selected colour. |
| | text fields. | |
| | | |
------------------------------------------------------------------------
```

Table 3.4.4B - Field Details Supplied during Field Definition (cont.)
++

3.4.5 Amending Existing Maps

+++++ ++++++++++++++++++++++

As well as using $FORM to create new maps, you can use the command to amend maps which have already been built. During the parameterisation phase the screen dimensions and run-time options previously specified are displayed, and can be selected by default. When format editing begins a picture of the current map appears, and can be amended as required: text can be rearranged, existing fields may be deleted, and new fields may be added. Record and field definition are part of the editing phase.

Note that if you make an alteration to a copy book which causes the picture clause or displacement of one or more existing fields to change, then all maps using the affected fields must be recreated. You need simply use $FORM to amend the existing maps, keeping the screen dimensions and run-time options the same, and making no changes during the format editing phase.

3.5 Restrictions and Design Consideration
+++ ++

This section lists the various restrictions you must bear in mind when you plan to use Screen Formatting in an application.

3.5.1 Screen Area Dimensions
+++++ ++++++++++++++++++++++

The absolute maximum is 23 lines by 132 characters, plus the baseline. In practice the maximum is normally further restricted because the screen area must lie entirely within the formatted area of the display terminal you are using, ie at best it can occupy the defined width of every line of the screen excluding the base line.

Note that terminals which automatically cause a line feed when the last character position of any line is written can only be supported if the sensitive character position is avoided. For this reason most Global Software using Screen Formatting works with a screen area no wider than 79 characters in order to run on 80-character terminals with automatic line feed.

3.5.2 Field Sequence
+++++ ++++++++++++++

It is important to remember that the order in which the MAPIN statement accepts its input and update fields can be defined using sequence numbers. The default if these are not used is from left to right, top to bottom, in the order in which the human operator would naturally read them. If this sequence option is used, MAPOUT...PRINT cannot be used.

Sequencing also affects the order in which fields are displayed and the appearance of the software can often be improved by careful sequencing. For example, we recommend that any boxes and lines are sequenced before other fields, so that the outline is drawn first and then filled in with the text and values.

3.5.3 Data Description
+++++ ++++++++++++++++

A single MAPOUT, MAPCLEAR or MAPIN statement can process output, input or update fields from as many as five different data areas. Each such area must be defined in a copy book supplied to $FORM in its book list. The copy books may contain a maximum of 60 fields each. All the books in the list must reside in the same copy library.

Ideally, if the recommendations made in Appendix E of the Global Cobol Language Manual are followed, each copy book should consist of the data declarations of the subordinate items of a single level 01 group. There may be level 01 or 77 re-definitions following the initial group, but they must only refer to items previous declared in that group. The following example obeys these rules:-

```
.BOOK EX
02 EXNAMS PIC X(30) * CLIENT NAMES
02 EXMSTA OCCURS 12 PIC X * MONTHLY STATUS
02 EXDATE PIC X(8) * PAYMENT DATE
02 EXTOTAL PIC 9(6,2) * AMOUNT
01 FILLER REDEFINES EXDATE
02 FILLER PIC X(6)
02 EXYEAR PIC 9(2) * YEAR OF CENTURY
77 EXDAY REDEFINES EXDATE PIC 9(6) * DAY OF MONTH
.END
```

When you use such a book in your application program you must precede the COPY statement by the necessary level 01 data declaration.

If the copy book contains &-characters substitution will take place as if the copy book name had been specified as the substitution string, just as it would in a COPY statement without a SUBSTITUTING clause. For example, if copy book EX contained lines:-

```
02 &&DATE PIC X(8)
02 &DATE2 PIC X(8)
02 &&&DAY PIC 9(2)
```

then the resulting lines would be:-

```
02 EXDATE PIC X(8)
02 EDATE2 PIC X(8)
02 EX-DAY PIC 9(2)
```

Any field name (apart from $$DATE or $$RECNO) that you supply during $FORM's field definition phase must label an elementary item belonging to one of the copy books supplied in the book list. If the item is a character field its picture clause must specify a length of no more than 80 bytes, since this is the maximum length that can be displayed or accepted in one operation.

Obviously, in order for your application program to compile correctly, each name within the book list must be unique. However, if you are using the compiler's "long names" option you should be aware that $FORM only uses the first 20 characters of the field name as an identifier, so you should make sure that your field names are unique within 20 characters.

3.5.4 Field Display Requirements
+++++ +++++++++++++++++++++++++

Any output, input or update field must be wholly contained on a single line of the display and occupy no more than 80 character positions. One field may overlap another field or a text area. A field assigned the date attribute uses 8 character positions.
++++

A field with the user conversion attribute has a user defined length. The requirements of any other type of field depend on its picture clause. The table summarises the situation.

In addition, if you have specified colon prompts as a run-time option, you must leave a space immediately to the left of each input and update field for the colon to occupy when the field is accepted.

3.5.5 Records
+++++ +++++++

When a map contains record area, each separate record must occupy
a fixed number of lines of the display. The total record area
must also occupy a fixed number of lines and columns. (The
number of records, and the number of lines they each require
together with the record area dimensions, is specified as an
option when the map is built.) The individual records must be
grouped together within a contiguous area of the screen: record
number 1 will appear at the top, left-hand position in this area,
and this will be immediately followed by the lines of record 2,

and so on. Non-indexed fields (including text) can be put on the same line as records, without them necessarily being repeated for each line of the record. If they are to be repeated for each line they must be given attribute R or Q depending on the record area. All fields in a record must have the same sequence number.

It is possible to have 2 records in a single record set, but they must either have different sequence numbers or be separated in display sequence by a non-record field. Note however that the value in the RECORD field applies to all records processed by a mapping statement. Note also that the 'numbers of record lines' in the map is not checked by the mapping routine, but used only by $FORM when displaying maps.

Output, input and update fields as well as text may be defined as part of a record.

3.5.6 Capacity

The $FORM command can only create maps containing 200 items or less. Each field and text string the map contains is an item, but for the purposes of $FORM, items belonging to records are counted only once. However, the limit of 200 items is unlikely to prove a restriction in any practical application.

TYPE OF FIELD	PICTURE CLAUSE	CHARACTER LOCATIONS REQUIRED
Date	9(6) COMP	8 (eg 01/01/88)
Long Date	9(6) COMP	10 (eg 01/01/1988)
Computational	9 [COMP]	1
or	S9 [COMP]	2
Display	9(p) [COMP]	p
Numeric	S9(p) [COMP]	p+1
	9(p,q) [COMP]	p+q+1
	S9(p,q) [COMP]	p+q+2

```
| | | |
| Character | X | 1 |
+++++++++
| | X(p) | p |
/ /
| | | |
| Pointer | PTR | 4 (four hexadecimal digits) |
+++++++
| | | |
| Y/N | PIC 9 COMP | 1 |
+++
| | | |
| $$DATE | 9(6) COMP | 8 (eg 29/04/88) |
++++++
| | | |
| $$RECNO | 9(2) | 2 |
+++++++
| | | |
| User | Any | User defined |
+++++
| Conversion | | |
++++++++++
| | | |
-------------------------------------------------------------
```

Table 3.5.5 - Field Display Requirements

4. Creating and Editing Maps Using $FORM

```
------------- ------------
S.copy C.map-id
---****----- ---******--

Copy Existing
Library Map
+ +
------------- ------------
| | @
| |
| |
| v
-------------- ------------
| | $FORM | $FMWRKnn
------------ -------**--
-----------> | |
| Screen |<----> Format
------------ | Formatter | Work File
| | |
-------------- ------------
| | | $CP $C
| | |
v | -------------- |
v v
------------ ------------ -------------
| L.map-id | C.map-id B.map-id
---******-- ---******-- ---******--
| |
| Format | Output Map Backup Map
| Listing *|
| *
------------ ------------
@ @
```

```
| |
------------------------------------------------------------
| |
| |
--------
| |
| direct access device |
| |
--------
| |
| |
| |
----------
| | | |
| | | printer or direct access device |
| | |
| |
| |
| + optional input file |
| |
| * optional output file |
| |
| @ these files, if present, must reside |
| on the same unit |
| |
------------------------------------------------------------
```

Figure 4.1 - Data flow during map preparation
++

4.1 Introduction
+++ ++++++++++++

Maps are compilation files created using $FORM. Once created, they can subsequently be edited by use of the same command. Typically this involves a certain amount of 'human engineering' - rearranging the screen to make it clearer for the operator and improving its appearance. The ability to edit maps also means that one map can be used as a 'template' for another, by editing and renaming it.

Do note that maps created using pre-V8.1 versions of $FORM will have to be run through V8.1 $FORM to be able to work with V8.1 screen formatting mapping subroutines. There will be an increase in compilation file size and a slight increase in linked program size. Maps created and updated by V8.1 $FORM will not run with pre-V8.1 screen formatting mapping subroutines.

Running $FORM involves you in three processes:-

* Map specification, in which you specify the name of the map
++++++++++++++++
to be created or edited, the copy libraries and book lists
it will reference (if any), the title of the map, the
dimensions of the screen on which it is to be run and the
map options. These last include the validation routine to
be used (if any), whether or not field sequencing is to be
used, auto-centering etc. It also includes any descriptive
comments about the map.

* Format editing, in which you define or amend the map as it
+++++++++++++++
is to appear on the screen. Some 28 editing instructions
are used to form a picture of what the operator will
actually see, and to determine the names, types and other
attributes of any fields the screen contains. Boxes and
lines can be drawn and colours defined. Autoinput is used
for all instructions to minimize key strokes: keying H on
its own, for example, causes a help screen to be displayed.

* Map generation and listing, in which the map created or
+++++++++++++++++++++++++
edited by the previous processes is converted from a work
file into a compilation file. If you have used an existing
map as a template you can change the name or unit of the
compilation file to avoid overwriting the template. When
generation is complete you have the option of producing a
hard copy listing of the screen you have created or edited
together with a field summary table, which lists the fields
and records used, and a map summary, which summarises the

map specifications defined in the first of these processes.

Figure 4.1 shows data flow during map preparation. $FORM outputs
a new map, a compilation file named C.map-id. The six-character
//////
suffix, the map-id, identifies the map when it is linked with a
//////
program which requires it. The map-id is also used in naming the
format listing, L.map-id, and the back-up copy of the original
//////
map, B.map-id.
//////

4.2 Running $FORM
+++ +++++++++++++

4.2.1 On-line volumes and units
+++++ +++++++++++++++++++++++++

$FORM uses a temporary work file, named $FMWRKnn, which it
//
creates on the compiler work unit, $C. (The quantity nn is the
//
current user number, to allow a number of different operators to
work with $FORM in a multi-user environment.)

The copy libraries containing all the book areas are optional
input files which need not be present if the map being created or
amended contains only text. The copy library volume may be
demounted once the copy library information has been confirmed,
but the volumes containing $FORM, P.$FORM and $FMWRKnn must be
//
permanently on-line.

Unless another unit is specified at the generation phase, a map
will be generated as C.map-id on the unit specified in phase 1:
//////
the original (if any) is retained for back-up purposes as
B.map-id on the original unit. Unless another unit is specified
//////
at print time, the format listing unit will be $PR.

4.2.2 Using the keyboard
+++++ ++++++++++++++++++

When $FORM requires input from the keyboard it displays a
message. On monochrome screens this is followed by a colon,
after which the reply should be typed. On colour screens the
area into which the reply should be typed is shown in the input
colour instead.

During the map specification and generation phases, when there
are a number of options available to you a message will be
displayed on the bottom line of the screen listing them. For
example:-

```
Key A to amend, C to cancel, <CR> to accept, <ESC>:
```

If there are a lot of options the message may be abbreviated as
follows:-

```
Key Amend, Cancel, <CR> to accept, <ESC>:
```

In either case you select one of the options by typing its first
letter and keying RETURN. The option <ESC> is selected by
pressing the ESCAPE key, and is standardly used to end a series

of operations and return to the previous prompt.

If you are filling in a series of fields such as the map options, you can use the LINE FEED or UP ARROW (ie CURSOR UP) key to step back from one field to the previous one in order to make corrections.

During the format editing phase auto-input is employed. A series of baseline prompts offers you a selection of the editing instructions available, most of which are chosen by your keying their initial letter. Thus, for example, to insert a blank line at the current cursor position you simply key I. You need not terminate your reply with <CR>, since this itself is one of the instructions (it causes the cursor to jump to the beginning of the next line of the screen).

Note that on some keyboards <ESC> must be keyed twice when used in the edit phase before it will take effect.

4.3 The Map Specification Process
+++ +++++++++++++++++++++++++++++

4.3.1 Selecting the Map
+++++ +++++++++++++++++
$FORM begins by displaying the software version number and
today's date, and then prompts you for the map-id of the map you
wish to create or edit and the unit on which it resides. If a
file named C.map-id does not already exist on the unit you
//////
specify, you are asked to confirm that this is indeed a new map
by replying Y or <CR> to the New map? prompt on the baseline. If
- --
you reply N, control returns to the beginning of this stage to
-
allow you to correct a keying error or mount a different volume.
If a file named C.map-id does already exist on the unit you can
//////
print it by keying Y to the baseline prompt:-
-

Do you wish to print this map? (N):

After printing control returns to the beginning of this stage so
that you can select another map.

4.3.2 Specifying Copy Libraries and Book Areas
+++++ ++
Once the map-id has been established, you can optionally supply
the file-ids and unit-ids for up to 3 copy libraries which the
map will reference. No information need be supplied if the map
is to consist of text alone. The default unit-id of the copy
libraries is $CL or, if $CL is not assigned, the unit on which
the map is being created. If you are editing an existing map,
the copy library unit will be defaulted to $CL and if $CL is not
assigned it will be set to the unit held in the map. The books
used to create it (if any) will be displayed as defaults.

In either case, the following baseline prompt appears:-

Key A to amend, <CR> to accept, <ESC>:

If you key A to amend the copy libraries and book areas screen
-
you are prompted to re-key the information (or supply it from
scratch in the case of a new map). If you do not key a prefix
when you input the file-id, $FORM assumes the library to be a
conventionally named source file and appends the S. prefix by
default. If you key <CR> to a file-id prompt then you are not

--
prompted for any more copy libraries.

If you specify one or more copy libraries, you are then prompted for the names of the book areas to be used. Note that the order in which these are named must be mirrored in the MD (map
++++
definition) of the calling program. If you key <CR> to a book
--
area prompt, then you will not be prompted for any more. To delete a copy library or book area, you must over type the entry with spaces. The following entries will be shifted up one position in the list. To replace one value with another simply over type the old value with the new one.

When all the values have been supplied the baseline prompt will be re-displayed. Note that there is no checking that the copy libraries and books exist until this screen has been accepted by a request to print or edit the map. If the library or book is not found, an error message will be displayed and the cursor positioned on the item in error.

4.3.3 Supplying the Map Title, Dimensions and Options
+++++ +++
Once you confirm that the copy libraries are correct, you must
supply the map title, the screen area dimensions and the various
options explained in 3.4.2.

If you are editing an existing file, its title will be displayed
as default. Key <CR> if you wish to accept the default title,
--
otherwise supply a new title of up to 30 characters.

If you are creating the map from scratch then the default screen
dimensions will be 23 lines by 79 columns, starting at line 1
column 1. If you are editing an existing map, the defaults will
be those used previously. The following baseline prompt is
displayed:-

Key A to amend, <CR> to accept, <ESC>:

You can key A to amend the details, <ESC> to return to the
- ---
previous stage and re-enter copy libraries and book areas, or
<CR> to accept these details.
--

The screen area dimensions cannot exceed those of the formatted
++++++
area of the terminal at which you are working.

You must reply 0 to the number of records prompt if the screen
does not contain records.

You must key spaces to the validation routine prompt if you do
not need to supply your own validation routine or if you would
prefer to set your validation routine in the map definition. If
you do provide a routine, the name you key in response to the
prompt is its entry-name, not its program name. Thus in the
++++++++++
example application there is a validation routine with program
name ORVAL in file C.ORVAL. However, the name coded in the entry
statement beginning the procedure division of the routine is
ORVALR, and this is the input which must be supplied to the
validation routine prompt when building the ORCUST map.

All the other run-time options are specified by your keying Y or
-
N (for Yes or No) or <CR> if an indicated default is to be used.
- --

The text start and end characters are defaulted to "[" and "]".
These are used by $FORM only and mark leading or trailing spaces

of text defined in the map.

INSTRUCTION	DESCRIPTION
A	Abandon the current edit
B	Box drawing
C	Centre the current line
D	Delete an area
E	Edit field, text, line or box
F	New field definition
G	Go back to options screen
H	Help
I	Insert a blank line
K	Kill the current blank line
L	Line drawing
M	Move an area
O	Overlay selection
P	Print the screen area

R	Re-display the screen
S	Copy an area
T	Text definition
U	Update the field definitions
V	Variant table for overlays re-definition
?	Display the current cursor position
SPACE	Insert a space at the current cursor position
RUBOUT	Delete the current cursor position.
RETURN	Position the cursor at the start of the next line.
HOME	Move the cursor alternately to the top left or bottom right corner.

Table 4.4 - Format Editing Instructions

++

INSTRUCTION	DESCRIPTION
CURSOR RIGHT, LEFT, UP, DOWN	The cursor is moved one character position. (Wrap-around takes place from one line to the next and from the last line to the first.)
TAB RIGHT	The cursor is moved to the start of the next field.
TAB LEFT	The cursor is moved to the start of the previous field.
ESC	End the edit phase (ESC must be keyed twice on some keyboards).

Table 4.4 (cont.) - Format Editing Instructions

4.4 Format Editing
+++ ++++++++++++++
Once you have confirmed that the screen dimensions are correct the format editing phase begins. You are able to position and rearrange text and fields within the screen area you defined in the previous phase, so that the display contains a picture of the information the operator will eventually see. If you are creating a map from scratch the screen area is initially blank. If you are modifying an existing map a representation of this map will appear in the screen area.

4.4.1 Editing Instructions
+++++ +++++++++++++++++++
A prompt:-

Move cursor and key Help, Field, Text, Box, Line, Edit, Delete

is displayed which gives the commonest editing instructions: note that auto-input is employed, so keying the initial character is sufficient for most of the instructions to take effect. Table 4.3.1 gives a full list, and can be displayed on the screen by keying H on its own.
-

A - Abandon the Current Edit
++++++++++++++++++++++++++++
Abandon the current updates leaving the original map (if any) unchanged. The prompt:-

Abandon - Are you sure?:

appears on the baseline in case you keyed A by mistake. You must
-
key Y to complete the abandon operation. Control returns to the
-
map-id specification stage described in section 4.2.2.

B - Box drawing
+++++++++++++++
Draw a box with the top left-hand corner at the current cursor position. The prompt:-

Move cursor to bottom right-hand corner of box and key B

is displayed on the baseline. You can key <ESC> to this prompt
---
to abandon the box drawing. Once the box has been correctly positioned the definition dialogue involves a series of baseline prompts.

If the field numbering option has been specified then the prompt:-

```
Field number (nn):
//
```

is displayed. If this is a new box the default is 0, otherwise
it is the old value. You should reply with a number in the range
0 - 99.

If the variant numbering option has been specified then the
prompt:-

```
Variant number (nn):
//
```

is displayed. If this is a new box the default is 0, otherwise
it is the old value. You should reply with a number in the range
0 - 99.

If the sequence numbering option has been specified then the prompt:-

Sequence number (nn):
//

is displayed. If this is a new box the default is 0, otherwise it is the old value. You should reply with a number in the range 0 - 99.

The attributes for the box must now be determined. The prompt:-

Attributes:

is displayed. Valid attributes for a box are Solid and Graphics. You should key the initial letter of any attributes required:-

S Solid
+
The whole area of the box is cleared when displayed.

G Graphics
+
If no special graphics are available, horizontal lines are drawn using minus signs.

If the colour option has been specified then the prompt:-

Colour combination (1):

is displayed on the baseline. You should key a colour combination in the range 1 - 8, or <CR> to accept the default.
--

A representation of the box will be displayed on the screen even if the terminal does not support the correct graphics.

C - Centre the current line
+++++++++++++++++++++++++++
Centre the line on which the cursor is currently positioned within the defined screen area.

D - Delete an area
++++++++++++++++++
Delete a rectangular area of the screen starting at the current cursor position. The prompt:-

Move the cursor to the end of the area to be deleted and key D

appears on the baseline. Once you have positioned the cursor at
the bottom right-hand corner of the area to be deleted and keyed
D the fields which start within this area will be deleted. If
-
the area you select consists of a single character position and
there are multiple fields starting at this position you will be
prompted for the fields to be deleted. If you have keyed D by
mistake you can key <ESC> to the prompt to abandon the delete.

E - Edit field, text, line or box
+++++++++++++++++++++++++++++++++
Edit an existing entry at the current cursor position. The
cursor may be placed anywhere within the display length of the
entry or on the edge of a box. If you have overlapping entries
at the current cursor position you will be prompted for the entry
to be edited. The dialogue then continues as documented for the
F, T, L or B instructions.

F - Field definition
++++++++++++++++++++
Define a field at the current cursor position. You are then prompted for the field name and attributes as described in section 4.4.2. If the screen contains repeated records, only the text and fields of the very first of them need be defined during format editing.

G - Go Back to Options Screen
+++++++++++++++++++++++++++++
This instructions takes you back from the map editing screen to the options update screen. You can change any options you have specified and the return to the map editing phase.

H - Help
++++++++
A help screen of all the available editing instructions is displayed. Any information previously displayed in the screen area will be restored once you continue. You can obtain a hard copy of this information by keying P in response to the prompt at
-
the foot of the screen.

I- Insert a blank line
++++++++++++++++++++++
Insert a blank line where the cursor is positioned, moving the current contents of this line, and those below it, down to make space for the new line. The current contents of the last line will disappear from the screen.

K - Kill the current blank line
+++++++++++++++++++++++++++++++
Remove the blank line where the cursor is positioned and move all the lines below it up one line.

L - Line drawing
++++++++++++++++
Draw a line starting at the current cursor position. The prompt:-

Move the cursor to the end of the line and key L

appears on the baseline. You can key <ESC> to abandon the line
---
drawing. Once the line has been correctly positioned the line definition dialogue involves a series of baseline prompts.

If the field numbering option has been specified then the prompt:-

Field number (nn):

```
//
```

is displayed. If this is a new line the default is 0; otherwise
it is the old value. You should reply with a number in the range
0 - 99.

If the variant numbering option has been specified then the
prompt:-

```
Variant number (nn):
//
```

is displayed. If this is a new line the default is 0 otherwise
it is the old value. You should reply with a number in the range
0 - 99.

If the sequence numbering option has been specified the prompt:-

Sequence number (nn):
//

is displayed. If this is a new line the default is 0 otherwise
it is the old value. You should reply with a number in the range
0 - 99.

The attributes for the line must now be determined. The prompt:-

Attributes:

is displayed. The only valid attribute for a line is Graphics.
You should key G if this attribute is required. If no special
-
graphics are available, horizontal lines are drawn using minus
signs.

If the colour option has been specified then the prompt:-

Colour combination (1):

is displayed on the baseline. You should key a colour
combination in the range 1 - 8, or <CR> to accept the default.
--

A representation of the line will be displayed on the screen even
if the terminal does not support the correct graphics.

M - Move an area
++++++++++++++++
Move a rectangular area of the screen with the top left-hand
corner starting at the current cursor position. The prompt:-

Move the cursor to the bottom right corner of the area to be
moved and key M

appears on the baseline. You should position the cursor at the
bottom right-hand corner of the area and key M to move fields
-
which start within this area. If the area you select consists of
a single character position and there are multiple fields
starting at this position, you will be prompted for the fields to
be moved.

You are then prompted for the position to which the top left-hand
corner of this area should be moved. The prompt:-

Move the cursor to where the top left corner should be moved
and key M

appears on the baseline. If you have made a mistake you can key <ESC> to either of the two prompts to abandon the move.

O - Overlay selection
+++++++++++++++++++++
If the screen overlays option has been specified for this map then you can select a new overlay for editing. The baseline prompt:-

Overlay (n):
/

is displayed. The default is the current overlay. You may clear the screen before displaying the new overlay or leave it in the current state, but you will not be able to edit the previous overlay fields in the new overlay selection. The following table shows the variant ranges belonging to each overlay:-

Overlay no.	Variant range
1	0 - 39, 40 - 49
2	0 - 39, 50 - 59
3	0 - 39, 60 - 69
4	0 - 39, 70 - 79
5	0 - 39, 80 - 89
6	0 - 39, 90 - 99

P - Print the screen area
+++++++++++++++++++++++++
Print the screen area as displayed, by writing it line by line to the format listing file. You are prompted for a listing unit: key <CR> for the default - $PR - or supply a valid unit-id or

--

address. You may key <CTRL A> if you requested the print by

------

mistake and wish to abandon the operation.

R - Re-display the screen
+++++++++++++++++++++++++
Re-display the screen if it has been corrupted.

S - Copy an area
++++++++++++++++
Copy a rectangular area of the screen with the top left-hand corner starting at the current cursor position. The prompt:-

Move the cursor to the bottom right corner of the area to be shifted and key S

appears on the baseline. You should position the cursor at the bottom right-hand corner of the area and key S to copy fields -
which start within this area. If the area you select consists of a single character position and there are multiple fields starting at this position, you will be prompted for the fields to be copied.

You are then prompted for the position to which the top left-hand
corner of this area should be copied. The prompt:-

Move the cursor to where the top left corner should be shifted
and key S

appears on the baseline. If you have made a mistake you can key
<ESC> to either of the two prompts to abandon the move.

T - Text definition
++++++++++++++++++
Define a piece of text to start at the current cursor position.
You must key in the text, which is terminated by <CR>.
--

The following functions are available to edit the text:-

<CR> Accept field;
++++

<CURSOR RIGHT> Move cursor to right within this text;
++++++++++++++

<CURSOR LEFT> Move cursor to left within this text;
+++++++++++++

<HOME> Move cursor alternately to start/end of text;
++++++

<ESC> Abandon text and return to baseline prompt;
+++++

<CLEAR> Delete text from cursor onwards;
+++++++

<RUBOUT> Erase character at cursor position, or the last
++++++++
character if at the end of the string;

<F1> Insert space at the current cursor position;
++++

<F2> Change case of the current character;
++++

If the field numbering option has been specified then the
prompt:-

Field number (nn):
//

is displayed. If this is a new piece of text the default is 0, otherwise it is the old value. You should reply with a number in the range 0 - 99.

If the variant numbering option has been specified then the prompt:-

```
Variant number (nn):
//
```

is displayed. If this is a new piece of text the default is 0, otherwise it is the old value. You should reply with a number in the range 0 - 99.

If the sequence numbering option has been specified then the prompt:-

Sequence number (nn):
//

is displayed. If this is a new piece of text the default is 0, otherwise it is the old value. You should reply with a number in the range 0 - 99.

If the colour option has been specified then the prompt:-

Colour combination (1):

is displayed on the baseline. You should key a colour combination in the range 1 - 8, or <CR> to accept the default.
--
In order to represent the length of the text on the screen, the first leading space is replaced by the text start character defined in the option select phase and the last trailing space is replaced by the text end character. If there are no leading or trailing spaces these markers are not displayed.

U - Update the field definitions
++++++++++++++++++++++++++++++++
The update function can be used to update a selected attribute of each field in turn. The attributes which may be updated are the field numbers, variant numbers, sequence numbers, colour or validation code. The prompt:-

Key Field no, Variant, Sequence, Colour, VC for validation code, <ESC>

is displayed on the baseline. If the field numbering, variant numbering, sequence numbering, colour option or validation routine has not been defined in the map, then the corresponding attribute will not be included in the prompt. When you have selected which attribute you wish to update, the cursor will be positioned at the start of each field in turn and you are prompted for the new value on the baseline. You can key in a new value, key <CR> to keep the old value, key <CTRL C> to go back to
-- ------
the previous field or key <ESC> to end the update (keeping any
---
new values already keyed). When the last field has been processed the normal editing prompt will be displayed.

V - Variant Table Definition for Overlays
+++
This command allows you to control the variant table range for various overlays of the map during this session of this map

editing using $FORM. This instruction is only valid the variant numbering and overlay options have been set.

Once you have keyed V a series of prompts asking for the variant range will be appear as follows:-

Overlay overlay number Variant range Start start End End
////////////// ***** ***

You must reply with the variant numbers which will begin and end the range for the particular overlay. The ranges for overlays 1 - 6 can be amended but overlay 0 (which appears together with every overlay is fixed as variants in the range 0 - 40.

? - Display the current cursor position
+++++++++++++++++++++++++++++++++++++++
This command displays the current cursor position on the baseline. The top left hand corner is counted as LINE 1 COLUMN 1.

<SPACE> - Insert a space
++++++++++++++++++++++++
Insert a space at the current cursor position. The rest of the line is shifted one position to the right.

<RUBOUT> - Remove character
+++++++++++++++++++++++++++
The space located at the current cursor position is deleted, and the rest of the line is shifted left one position. A space is inserted at the end of the line.

<RETURN> - Next line
++++++++++++++++++++
The cursor is positioned at the start of the next line, or at the top left hand corner if it previously occupied the last line.

<HOME> - Home the cursor
++++++++++++++++++++++++
The cursor is moved alternately to the top left hand corner and the bottom right hand corner of the screen.

<ESC> - End the edit
++++++++++++++++++++
The edit is ended and control continues at the generation phase described in 4.5. Note that on some terminals <ESC> must be keyed twice before it takes effect.

4.4.2 Field definition
+++++ ++++++++++++++++
The field definition dialogue which follows your keying the F editing instruction involves a number of baseline prompts.

4.4.2.1 Field name
+++++++ ++++++++++
When you define a field you must first supply the field name. The prompt:-

Field name, ? for help (xxxxxxxxxxxxxxxxxxxx):
///////////////////

is displayed. If a field is already defined at this position,
its name will be the default. If you are not sure of the field
name you can key ? to display the source of a copy book included
-
in the map in the formatted area of the screen.

If there is only one copy book included in the map, then it is
displayed immediately on the screen, but if there is more than
one copy book included then the prompt:-

```
Copy book(xx, xx, xx, xx, xx):
/////////////////
```

is displayed and you should key in the name of the copy book you
wish to display.

The field name prompt is re-displayed on the baseline, but there
will be no default. Keying <CR> will page through the copy book.
--

Keying <ESC> to the field name prompt at any stage will abandon
---
the field definition.

4.4.2.2 Occurrence
+++++++ ++++++++++
If this is an occurring field you will be prompted for the
occurrence number. You should key the required occurrence number
or R if this is to be part of a record.
-

4.4.2.3 Field number
+++++++ ++++++++++++
If the field numbering option has been specified then the
prompt:-

Field number (nn):
//

is displayed. If this is a new field the default is 0, otherwise
the default is the old value. You should reply with a unique
number in the range 1 - 99, or 0 if the field is not numbered.

4.4.2.4 Variant number
+++++++ ++++++++++++++
If the variant numbering option has been specified then the
prompt:-

Variant number (nn):
//

is displayed. If this is a new field the default is 0, otherwise
it is the old value. You should reply with a number in the range
0 - 99.

4.4.2.5 Sequence number
+++++++ +++++++++++++++
If the sequence numbering option has been specified then:-

Sequence number (nn):
//

is displayed. If this is a new field the default is 0, otherwise
it is the old value. You should reply with a number in the range
0 - 99.

4.4.2.6 Field type

```
+++++++ ++++++++++
```

The next stage is to determine the field type. The system variables, $$DATE and $$RECNO, are always output fields. For all other fields the prompt:-

```
Field type (x):
/
```

is displayed. The type is indicated by entering the initial character of the type Output, Input or Update. If this is a new field the default is the type used in the previous field definition or if there have been no previous definitions there is no default. For existing fields the default is the old value.

```
4.4.2.7 Validation code
+++++++ +++++++++++++++
If a validation routine has been specified and this is not one of
the system fields, $$DATE or $$RECNO, the prompt:-

Validation code (nn):
//

is displayed. If this is a new field the default is 0, otherwise
it is the old value. You should reply with a number in the range
0 - 99, indicating the user validation to be performed. A reply
of 0 implies no user validation.

| | | |
---------------------------------------------------------------
| | | |
| FIELD ATTRIBUTES | TYPICAL PICTURE | REPRESENTATION |
++++++++++++++++ +++++++++++++++ ++++++++++++++
| | | |
---------------------------------------------------------------
| | | |
| CHARACTER | X(8) | xxxxxxxx |
+++++++++ ////////
| | | |
| DISPLAY NUMERIC | 9(6,2) [COMP] | nnnnnn.nn |
+++++++++++++++ //////////
| or COMPUTATIONAL | S9(6,2) [COMP] | -nnnnnn.nn |
++++++++++++++++ //////////
| | | |
| DISPLAY NUMERIC | 9(8) [COMP] | ffffffff |
+++++++++++++++ //////////////////
| or COMPUTATIONAL | | |
++++++++++++++++
| NUMBER-FILLED | | |
+++++++++++++
| | | |
| POINTER | PTR | hhhh |
+++++++ ////
| | | |
| Y/N FLAG | 9 COMP | y |
++++++++ /
| | | |
| USER CONVERSION | 9(4) COMP | uuuuuuuu |
+++++++++++++++ /////////
| | DISPLAY LENGTH 8 | |
| | | |
| DATE | 9(6) COMP | dd/mm/yy * |
++++ /////////
| | | |
| LONG DATE | 9(6) COMP | dd/mm/yyyy * |
+++++++++ //////////
```

```
| | | |
| $$DATE | 9(6) COMP | dd/mm/yy * |
++++++ /////////
| | | |
| $$RECNO | 9(2) | nn |
+++++++ //
| | | |
--------------------------------------------------------------
```

* Dates will be represented by the characters mm/dd/yy or
mm/dd/yyyy if American processing is in force.

Table 4.4.2 - Sample Representation of Defined Fields
+++

4.4.2.8 Attributes
+++++++ ++++++++++
Next the attributes must be determined. The prompt:-

```
Attribute (xxxx):
////
```

appears. The default is the current list of attributes. To add
another attribute to the list key the single letter code
representing this attribute. If you key U (the user conversion
-

attribute) the additional Display length prompt appears. You should key the required display length of this field which must be in the range 1 - 80 or within the boundaries of the map. Key <CR> to end the list or <CTRL B> to delete the whole list.
-- ------

A list of the attribute letters to be used can be found in table 3.4.4 along with a description of the attributes and their effects.

4.4.2.9 Colour
+++++++ ++++++
If the colour option has been specified then the prompt:-

Colour combination (n):
/

is displayed. If this is a new field the default is 1, otherwise
it is the old value. You should key a colour combination in the
range 1 - 8, or <CR> to accept the default.
--

There is a representation of each field defined in the map,
++++++++++++++
taking the form of a sequence of lower case characters appearing
in the screen locations that it occupies. The representation
indicates the picture and other main attributes of the field in a
natural way.

4.4.2.10 Help Book Name
++++++++++++++++++++++
If the field help option has been set then the prompt prompt:-

Help book name (xx):

is displayed. The default is the default help book name as keyed
in the options screen You should key a help book name or <CR> to
accept the default. The help book will be set when the field is
reached. The help library must be set in the program calling
screen formatting from the maps.

4.5 Map Generation and Listing
+++ +++++++++++++++++++++++++

Map generation and listing takes place when format editing is
complete. You must first supply the map-id of the new map and
unit on which it is to be created. The prompts:-

New map-id (nnnnnn): Unit (nnn):
////// ///

appear on the baseline. The defaults are the map-id and unit-id
of the map which you are editing. If there is already an exist-
ing map of the same name, the previous version of this file will
be renamed as B.map-id. You are asked to confirm this by keying
//////
Y to the FILE ALREADY EXISTS - CONTINUE?: prompt.
-

The word GENERATING now appears on the baseline. This signifies
that $FORM is creating the file C.map-id.
//////

When the map has been generated, the final prompt:-

Listing unit:

is displayed on the baseline. You should key a unit-id or <CR>,
--
for the unit to which $PR is assigned, to document your work by
writing a representation of the screen area together with a field
summary table to format listing file L.map-id. If you wish to
//////
suppress the listing, you may key <CTRL A>.

When listing is complete control returns to the map-id and title
stage of the parameterisation phase so that you can create or
edit another map. The map-id prompt explained in 4.2.2 appears.
If you do not want to generate another map, key <ESC> to exit
from $FORM.

5. Global Cobol Support for Screen Formatting

5.1 Introduction
+++ ++++++++++++
Seven additional Global Cobol statements, not defined elsewhere are provided for Screen Formatting. They are:-

MD MAPOUT
RECORD MAPCLEAR
VARIANT MAPIN
AREAS

Those listed in the left-hand column are data division statements used to construct a special type of level 01 group item known as a map definition (or MD, for short). The MD contains most of the information required to control a mapping operation, including the address of the map involved, the record number parameter, and pointers to up to five data areas containing the output, input and update fields. Each map definition is identified by its unique mapname, a symbol labelling its start address.
+++++++

The map processing statements, MAPOUT, MAPCLEAR and MAPIN, are coded in the procedure division to cause sequences of console accept and display operations to take place. Each statement specifies a mapname as its first operand, and information from the map definition, together with other operands of the processing statement itself, serves to fully parameterise the particular mapping operation involved.

The MAPOUT, MAPCLEAR and MAPIN statements do not themselves generate any accept or display logic. They simply establish a linkage to the mapping routine which is responsible for servicing
+++++++++++++++
all mapping requests. The mapping routine is Global system routine named SM$A, which is contained in system library C.$APF. The routine is automatically linked with programs which require it.

5.1.1 Use of Basic Console Handling Facilities
+++++ ++
The basic console I/O statements described in section 2 of this Manual:-

CLEAR DISPLAY BELL DISPLAY...LINE
SCROLL ACCEPT ACCEPT...LINE

can all be used in conjunction with Screen Formatting.

A CLEAR statement must be executed prior to the first MAPOUT or
++++

MAPIN to establish the formatted mode of operation. You should terminate your program with an explanatory error message if CLEAR signals the inappropriate terminal type exception, since if you attempt a MAPOUT, MAPCLEAR or MAPIN operation when the System Manager is not operating in formatted mode the mapping routine itself will terminate your job with a stop code.

You may use the SCROLL statement to re-establish teletype mode, but if you do you must be careful to execute another CLEAR statement before your next MAPOUT, MAPCLEAR or MAPIN.

	EFFECT ON INDICATED OPERATION WHEN <ESCAPE> KEYED	
$$ESC	MAPIN	ACCEPT ACCEPT...LINE CALL ACCE$
0	Control returns immediately to the monitor	Control returns immediately to the monitor
1	Treated as though <CR> had been keyed	Treated as though <CR> had been keyed
2	Any characters keyed are accepted, and then exception condition 2 is signalled	Treated as though <CR> had been keyed

Table 5.1 - Escape key control using $$ESC

Because a map can only occupy the formatted area of the screen, which excludes the base line, the teletype DISPLAY and ACCEPT statements, which can access that line during formatted working, are particularly useful. They may be employed to display error messages or accept single field corrections. (See for example the programming notes concerning the MAPIN...FIELD statement in section 5.4.4.) You should note that the base line remains unchanged following a MAPOUT or MAPCLEAR operation, but is cleared as a result of a MAPIN.

The BELL statement can be used to sound the console's audible alarm if an input is incorrect, and thus enhance the effect of a DISPLAY statement produced by a validation routine.

DISPLAY...LINE and ACCEPT...LINE can be used in conjunction with the map processing statements, but then it is your responsibility to ensure that the resulting console I/O does not interfere with the part of the screen used by Screen Formatting. The same applies to the use of the console handling system routines (VIDEO$, POSI$, ACCE$, DISP$, PASS$, CHAR$, ECHO$ and TXEDT$) described in the section 7 of this Manual. In general, normal applications should be able to perform all their console handling using a combination of CLEAR, SCROLL, MAPOUT, MAPIN, DISPLAY and ACCEPT statements. However, Screen Formatting does not prevent you from developing more sophisticated programs, such as graph plotters, in which, for example, MAPOUT might create a basic screen frame and calls on POSI$ and DISP$ might be used to draw a chart within that frame.

5.1.2 Escape Key Control using $$ESC
+++++ +++++++++++++++++++++++++++++

By setting the system variable $$ESC you can determine how the statements responsible for console input respond when the operator keys <ESCAPE>. This is shown in Table 5.1.

When a program is first entered from the System Manager, $$ESC will be zero so, if you do not alter it, control will return to the monitor if the operator hits the ESCAPE key. This is the convention adopted throughout most System Manager commands used in program development. However, for normal application work it is customary to set $$ESC to 1 so that the operator cannot inadvertently terminate a job by touching the key by mistake. You may, however, wish to use the escape key for your own purposes, and in this case you should set $$ESC to 2. This will cause the mapping routine to handle <ESCAPE> by suppressing the

------

affected MAPIN statement with exception condition 2 so that the program in control can process the key stroke specially.

5.2 The Map Definition
+++ ++++++++++++++++++
You must code a map definition (MD) in your program's data division so that you can introduce the map itself, the record number parameter, the variant number parameter, and the data areas to be passed to the mapping routine whenever a MAPOUT or MAPIN statement is executed. The MD normally occupies working storage. However, it is possible to pass a map definition as a parameter to a routine entered by means of a CALL or EXEC statement, and in such a routine the passed MD would be coded in the linkage section.

The general format of a map definition is:-

```
MD mapname [MAP map-id]
/////// //////
[RECORD recno]
/////
[VARIANT varno [vartab]]
///// //////
[VALIDATION routine-name]
///////////
[AREAS|AREA list]
////
```

The map definition establishes a special level 01 group data item whose name is mapname. If the RECORD and VARIANT statements
///////
appear their operands label subordinate items within this group. Depending on how they are coded, map-id, and the entries within the area list, either generate symbols labelling other subordinate items or establish fixed values within the MD.

Note that if you code a map definition in the linkage section you must use the symbol option for the map-id and area list entries. Alternatively, although a mapname must be supplied, you can simply omit the MAP phrase and other optional statements.

5.2.1 The Mapname
+++++ +++++++++++
The mapname must be a symbol (see section 2.1.2 of the Language
///////
Manual). It labels the map definition and must appear as the first operand of any map processing statement accessing it.

5.2.2 The MAP Clause
+++++ ++++++++++++++
The MAP clause may be omitted when the MD is part of the linkage section, but must be present if it occupies working storage. The map-id is normally coded as "mmmmmm" where mmmmmm is the one to
////// ////// //////

six character map-id defined when the map was constructed by $FORM, used in naming the compilation file containing the map. Thus, if you had previously used $FORM to create a map file named C.ORDATA you would associate the MD with this map by coding a MAP clause of the form:-

MAP "ORDATA"

Alternatively, the map-id may be coded as a symbol in which case
//////
the statement:-

02 symbol PIC PTR
//////

is generated within the map definition. This option must be used if the map file is not linkage-edited with the program containing the map definition. It enables a separate program file containing the map to be loaded at run-time. The separately loaded map can then be used in the normal way providing its entry point address is stored in the map-id field before any MAPOUT or MAPIN statement is attempted.

Separately loaded maps and symbolic map definitions are described
in more detail in sections 5.3 and 5.4.

5.2.3 The RECORD Statement
+++++ +++++++++++++++++++

The RECORD statement must be supplied if the map to be processed
contains records. Two record statements are allowed, the first
record statement referring to record set 1 (R attribute) and the
second record set statement referring to record set 2 (Q
attribute). The quantity recno is coded as a symbol causing the
/////
statement:-

```
02 recno PIC 9(2) COMP
/////
```

to be generated in the MD. The recno field is initialised to be
/////
zero, but you may alter it either by moving a record number in
explicitly, or by using the QUERY option of the MAPOUT statement.

MAPOUT...QUERY returns you the total number of records defined by
the map in the recno fields for each record set. The value
/////
supplied will be zero if the map contains no records.

The recno field is read-only as far as the other map processing
/////
options are concerned. You are responsible for setting it to
determine how many records are affected by MAPOUT...TEXT,
MAPOUT...PRINT, MAPOUT...OUTPUT, MAPOUT...ALL and MAPIN...ALL.
If recno is zero no records will be processed: if it contains a
/////
positive number, n say, then the statement will apply to the
/
first n records that appear on the screen. In a similar way the
/
value you supply in recno determines which record is affected by
MAPOUT...RECORD and MAPIN...RECORD and (when the field involved
belongs to a record) MAPOUT...FIELD and MAPIN...FIELD. You must
be careful to set up recno correctly because the mapping routine
will terminate your program in error if the recno value is
inappropriate for the map processing statement involved. For
maps without records recno should always be zero, as will be the
case if no RECORD statement is coded in the MD.

A field within a record may be multi-valued, i.e. defined either
++++++++++++
as a member of a repeating group or as an elementary item with an
OCCURS clause. When processing a record containing such a field,
mapping selects the occurrence that corresponds to the record

number. Thus the value of recno that you supply before executing
MAPOUT...RECORD, MAPIN...RECORD, MAPOUT...FIELD or MAPIN...FIELD
also determines which occurrences (if any) of multi-valued fields
within records are affected by the statement.

5.2.4 The VARIANT Statement
+++++ +++++++++++++++++++++
The VARIANT statement must be supplied if the map contains
variants which are to be selectively processed, as described in
section 5.2. The quantity varno may be coded as a symbol,
/////
causing the statement:-

02 varno PIC 9(4) COMP
/////

to be generated in the MD. The varno field is initialised to
/////
zero, meaning all variants are to be processed. You may also
code varno as an integer literal, in which case the variant
/////
number field in the MD is initialised to this value, and cannot
be changed.

The optional field vartab is coded as a symbol, causing the
//////
statement:-

02 vartab PIC PTR
//////

to be generated in the MD. If varno is set to -1, then vartab
///// //////
must be established to point at a table of the variants to be
selected.

The varno and vartab fields are both read-only as far as map
///// //////
processing statements are concerned. They can be set up to cause
subsequent MAPOUT, MAPIN and MAPCLEAR statements to be restricted
to a selected subset of the fields within the map.

5.2.5 Validation Statement
++++++++++++++++++++++++++
The validation line must be supplied if the validation code
option has been set in the map and no validation routine has been
supplied during map generation. The routine-name is either a
////////////
character string in quotes, identifying the validation routine
label or a symbol. When a symbol is used the statement:-

02 routine-name PIC PTR
////////////

Is generated within the MD. The program must then place the
routine-name to be used in the field before using the map
////////////
processing statements.

5.2.6 The AREAS (or AREA) Statement
+++++ +++++++++++++++++++++++++++++
The AREAS (or AREA) statement must be supplied unless the map to
be processed contains only text. You code either:-

AREAS list
////
or:-
AREA list
////

where the list consists of between one and five area names each
//// +++++++++++
of which identifies a data area used by the mapping routine. The
first such area must contain the data described by the first copy
book provided in the book list supplied to $FORM when the map was

created, the second area the data described by the second copy book, and so on. If you provide more area names than there were books the extra names will simply be ignored. Note, however, that your program will be terminated in error if it executes a map processing statement requiring access to a field from an area for which an area name was not supplied.

When the area occupies a fixed location in working storage you code its area name as:-

"data-name"
/////////

where data-name labels its first byte. Normally the area will be
/////////
part of the working storage of the compilation containing the MD, but it is possible for it to reside in some other module participating in the same linkage edit, providing the data-name is defined as a global symbol using the GLOBAL statement explained in section 5.6 of the Global Cobol Language Manual.

If the address of the area is to be determined at run-time,
before the first MAPOUT or MAPIN statement affecting the MD is
executed, omit the quotes and code the area name as:-

symbol
//////

This will cause a PIC PTR field named symbol to be set up within
//////
the MD. The program using the map definition will then be
responsible for initialising this field to address the start of
the data area before it is used.

5.2.7 An Example Map Definition
+++++ ++++++++++++++++++++++++

The example below shows an MD with fixed values set at compile
time. An example of an MD using the symbol options is given in
section 5.4.

```
.
.
DATA DIVISION
.
.
MD M-DATA MAP "ORDATA"
RECORD R-DATA
VALIDATION "M-VAL"
AREAS "OR" "WK"
.
.
.
01 OR
COPY OR
01 WK
COPY WK
LINKAGE SECTION
01 FV
COPY FV
.
.
.
PROCEDURE DIVISION
.
.
.
ENTRY M-VAL USING FV
.
.
.
ENDPROG
```

The mapname is M-DATA and the map-id is ORDATA. The recno field is named R-DATA. The validation routine is set to M-VAL with the FV block as its parameter on entry. The two copy books "OR" and "WK" formed the book list previously defined during $FORM's parameterisation phase, and the AREAS statement makes the data areas described by these books available to the mapping routine. The sequence in which the books appear is important and we strongly suggest that you adopt the convention that the book list is always defined in alphabetical order.

5.2.8 Recommended Naming Conventions
+++++ +++++++++++++++++++++++++++++++

Appendix E of the Global Cobol Language Manual recommends naming conventions that you can follow to prevent inadvertent name clashes when developing large, sophisticated applications. This section explains how these conventions should be extended for Screen Formatting.

Each map-id used by a product should be named:-
//////

ppmmmm
//////

where pp is the two character product code (such as OR for "Order
//
Entry") and mmmm is an alphabetic mnemonic, between 1 and 4
////
characters in length (e.g. DATA). You will have to maintain an index to ensure that all the map-ids used throughout a product are unique, but providing this is done the other conventions prevent name clashes affecting different entities.

The mapname should be named M-mmmm, after the map-id. Similarly
/////// ////
the recno and varno symbols, if defined, should be R-mmmm and
///// ///// ////
V-mmmm.
////

The naming of copy books and validation routines should follow the rules already established for copy books, program names and entry names.

```
| | | | |
------------------------------------------------------------------------
| | | | |
| MAPOUT | CONSOLE ACTION | USE OF recno FIELD | EXCEPTION CONDITIONS |
 +++++++ ++++++++++++++ +++++++///// ++++++ ++++++++++++++++++++
| OPTION | | | ($$COND) |
 ++++++ ++++++++
| | | | |
------------------------------------------------------------------------
| | | | |
| QUERY | None. | Mapping returns n, | None |
 +++++ /
| | | the number of | |
| | | records defined in | |
| | | for each record set| |
| | | in the map. | |
| | | | |
| ALL | Display text, | You set recno to a | Display overflow (2) |
 +++
| | output and update| value between 0 and| |
| | fields. Clear | n, to indicate how | |
 /
| | input fields. | many records are to| |
| | | be processed. | |
| | | | |
| OUTPUT | Display output | Same as ALL option.| Display overflow (2) |
 ++++++
| | and update | | |
| | fields. Clear | | |
| | input fields. | | |
| | | | |
| TEXT | Display text. | Same as ALL option.| None |
 ++++
| | | | |
| RECORD | Display text, | You set recno to a | Display overflow (2) |
 ++++++ /////
| | output and update| value between 1 and| |
| | fields. Clear | n to indicate which| |
 /
| | input fields. | record is affected.| |
| | (For one selected| | |
| | record only). | | |
| | | | |
| FIELD | Display a single | If the field is | Invalid |
 +++++
| | output or update | part of a record | field number (1) |
| | field, or clear | you must set recno | Display overflow (2) |
 /////
```

```
| | an input field, | to a value between | |
| | identified by its| 1 and n to indicate| |
 /
| | field number. | which occurrence is| |
| | | affected. | |
| | | | |
| PRINT | None. The screen| You set recno to a | Irrecoverable |
 +++++ /////
| | image is written | value between 0 and| I/O error (1)*|
| | to the print file| n to indicate how | File space |
 /
| | supplied as the | many records are | exhausted (2) |
| | third operand. | to be logged. | |
| | | | |
| (Note that the PRINT option cannot be used if sequencing is employed) |
 ++++++
| | | | |
----------------------------------------------------------------------
```

* This exception is only signalled if OPTION ERROR
is coded in the printer FD.

Table 5.3 - MAPOUT Statement Processing Summary

```
++++++++++++++++++++++++++++++++++++++++++++++
```

5.3 The MAPOUT Statement
+++ +++++++++++++++++++
The MAPOUT statement is used to determine how many records, if
any, a map contains, or to display a selection of the text,
output and update fields defined when the associated map was
created. There is also an option to allow you to print all or
part of the screen image. The statement is coded:-

MAPOUT mapname option [field|filename]
////////////// ///// ////////
e.g:-

MAPOUT M-DATA QUERY

or:-

MAPOUT M-DATA FIELD 3

or:-

MAPOUT M-DATA PRINT LOGFILE

Here mapname labels the map definition on which the statement
///////
operates. The option operand is one of the words QUERY, ALL,
//////
OUTPUT, TEXT, RECORD, FIELD, or PRINT. The third operand is
mandatory for MAPOUT...PRINT or MAPOUT...FIELD, but may be
omitted in the ALL, OUTPUT and RECORD options. For the PRINT
option it is the filename of an open relative sequential FD
////////
defining the print file to which the screen image is to be
written. For MAPOUT...FIELD the third parameter is the field
/////
number of the field involved in the operation. In the ALL,
//////
OUTPUT and RECORD options the third parameter is the field number
////////////
of the first field to be processed: if this parameter is omitted
the operation will start with the first appropriate field in the
map. The field number must be either an integer literal or PIC
9(4) COMP variable.

MAPOUT only works if a previous CLEAR statement has established
formatted mode: an attempt to use MAPOUT when the System Manager
is executing in teletype mode will cause your program to be
terminated in error.

Table 5.3 and the following descriptions explain the functioning
of the various MAPOUT options. All apart from MAPOUT...QUERY and
MAPOUT...TEXT can suffer an exception condition in abnormal

circumstances, and for this reason the MAPOUT statement should be followed where appropriate by an ON EXCEPTION statement to introduce the necessary special logic. If an exception is signalled and no such statement is present, your program will be terminated in error. The display overflow exception (2) is
+++++++++++++++++
provided for use in advanced applications where the map, the data, and the program using them are constructed separately at different times, by different users. Under normal circumstances, when you are responsible for providing both clean data and a valid map, display overflow should not occur once your system has been fully debugged. The exceptions resulting from a MAPOUT...PRINT are the same as those which may be generated in response to a normal WRITE NEXT statement operating on the print file.

5.3.1 The MAPOUT...QUERY Option
+++++ +++++++++++++++++++++++++
The QUERY option is used to determine how many records, if any,
are defined in the map. This value is returned in the recno
++++++++ /////
field for both record sets. If there are no records in that
record set, recno will be set to zero. The value you supply in
///// ++++++
recno for the other mapping operations must never exceed the
/////
value returned by the QUERY option, otherwise your program will
be terminated in error.

No console I/O takes place as a result of a MAPOUT...QUERY, and
the operation cannot suffer an exception.

5.3.2 The MAPOUT...ALL Option
+++++ +++++++++++++++++++++++
The ALL option causes text, output and update fields to be
displayed, and input fields to be set to spaces. If the map
contains records, you must set recno for the appropriate record
/////
set, to indicate how many of them, if any, are to be affected by
the operation. Note that if you set recno greater than the total
/////
number of records defined by the map your program will be
terminated in error. If recno is zero, no records will be
/////
processed. If a field number is supplied output will start at
the specified field, and earlier fields will not be displayed.

The display overflow exception (2) will be signalled if any
++++++++++++++++
numeric (i.e. computational or display numeric) output or update
field involved in the operation contains a value which does not
conform with the picture clause of the field, and thus cannot be
displayed. In this case normal console output will still take
place, but the field or fields in error will appear as asterisks.

5.3.3 The MAPOUT...OUTPUT Option
+++++ ++++++++++++++++++++++++++
The OUTPUT option causes output and update fields to be displayed
and sets input fields to spaces. It also displays text fields
within records which have been selected. (It is similar to
MAPOUT...ALL, except that text fields outside records are not
involved in the processing.) If the map contains records you
must set recno for the appropriate record set, to indicate how
/////
many of them, if any, are to be affected by the operation. If
you set recno greater than the total number of records defined by
/////

the map your program will be terminated in error. If recno is
/////
zero, no records will be processed. If a field number is
supplied output will start at the specified field, and earlier
fields will not be displayed.

The display overflow exception (2) will be signalled if any
++++++++++++++++
numeric output or update field involved in the operation contains
a value which does not conform with the picture clause of the
field, and thus cannot be displayed. In this case normal console
output will still take place, but the field or fields in error
will appear as asterisks.

5.3.4 The MAPOUT...TEXT Option
+++++ +++++++++++++++++++++++

The TEXT option causes the text fields of the map to be
displayed. The $$RECNO and $$DATE fields will also be displayed
when they are present in the map. (A MAPOUT...TEXT followed by a
MAPOUT...OUTPUT has exactly the same eventual effect on the
screen as a MAPOUT...ALL.) If the map contains records you must
set recno of the appropriate record set to indicate how many of
/////
them, if any, are to be affected by the operation.

If you set recno greater than the total number of records defined
/////
by the map your program will be terminated in error. If recno is
/////
zero, no records will be processed.

Since the text consists only of character constants, the
MAPOUT...TEXT option can never suffer a display overflow
exception.

5.3.5 The MAPOUT...RECORD Option
+++++ +++++++++++++++++++++++++
The RECORD option should only be used if the map contains
records. Before executing the statement you must set recno to
/////
the record number of the appropriate record set that you wish to
process. Then MAPOUT...RECORD causes all text, output and update
fields of that record to be displayed, and all input fields to be
set to spaces. Note, however, that if the map does not contain
records, or if you supply an invalid number in recno, your
/////
program will be terminated in error. If a field number is
supplied output will start at the specified field, and earlier
fields will not be displayed.

The display overflow exception (2) will be signalled if any
++++++++++++++++
numeric output or update field involved in the operation contains
a value which does not conform with the picture clause of the
field, and thus cannot be displayed. In this case normal console
output will still take place, but the field or fields in error
will appear as asterisks.

5.3.6 The MAPOUT...FIELD Option
+++++ ++++++++++++++++++++++++
The FIELD option operates on output, update or input fields which
were assigned a unique field number when the map was created by
$FORM. The MAPOUT...FIELD statement causes a single field,
identified by the number supplied as its third operand, to be
processed. If the field is defined as output or update, its
current value is displayed: if it is an input field it is set to
spaces.

If the map contains records and the numbered field is part of a
record, you must set recno to identify the particular occurrence
/////
involved before executing the MAPOUT...FIELD statement. If you
supply an invalid record number your program will be terminated
in error.

The invalid field number exception (1) will be signalled if the

+++++++++++++++++++++
field number you supply was not defined when the map was
generated. In this case no console output will take place.

The display overflow exception (2) will occur if a numeric output
++++++++++++++++
or update field was involved and it contains a value which does
not conform with its picture clause and thus cannot be
displayed. In this case the erroneous field will be output as
asterisks.

5.3.7 The MAPOUT...PRINT Option
+++++ +++++++++++++++++++++++++
The PRINT option causes all or part of the current screen image
to be written to the print file whose FD is supplied as the third
operand of the statement. One print line is output for each line
of the screen area (the dimensions of which are specified when
the map is generated). If the map contains records, you must set
recno to indicate how many of them, if any, are to be logged. If
/////
you make recno less than the total number of records, blank print
/////

lines will appear in the place of the records you have not
selected, since the number of lines output for a particular map
is always the same. Graphics fields are not printed.
+++
You cannot use the MAPOUT...PRINT option if sequencing is
++++++
employed in the map in question.

The print file definition must be opened before MAPOUT...PRINT is
used, and it must specify an RS file with a record length between
the screen width and 133 inclusive. Each RS record produced by
the MAPOUT statement begins with a print control byte set to 1 so
that the stationery advances to a new line. You are responsible
for paginating the output correctly, and providing page headers
if you require them, by using conventional WRITE NEXT statements
to output the necessary additional print lines, as explained in
section 2 of this manual. If the map specifies auto-centre then
the image is centred within the specified record length.

The irrecoverable I/O error exception (1) will be signalled if
++++++++++++++++++++++
OPTION ERROR was coded in the FD and a hardware fault occurred
when writing to the print file. In the normal case, when OPTION
ERROR is omitted, an irrecoverable I/O error simply causes the
affected program to be terminated with a stop code.

The file space exhausted exception (2) can only occur when the
+++++++++++++++++++
print file is assigned to direct access storage, rather than a
real printer.

5.3.8 Programming Notes
+++++ ++++++++++++++++++
You will not normally need to use the QUERY option because you
will usually know at an early stage of your design how many
records a particular map can possibly contain. The option is
provided for advanced applications (such as AutoClerk's Form
handling) where the maps are created independently of the program
which is to process them. We have used MAPOUT...QUERY in the
sample application so that you can alter the number of records on
the order entry screen without having to change the ORDENT
program: in some circumstances you may wish to use the option to
achieve similar flexibility.

MAPOUT options apart from QUERY and PRINT normally cause console
output to take place, although in some cases they may have no
effect (e.g. a MAPOUT...OUTPUT applied to a map containing only
text fields, or when a MAPOUT...ALL, MAPOUT...OUTPUT or
MAPOUT...RECORD statement uses a field number or variant number
which does not exist, or specifies a field number excluded by the
selected variant). When fields are displayed they are output

from left to right, top to bottom, in the order in which they would naturally be read.

The MAPOUT...ALL and MAPOUT...TEXT options are usually employed after a CLEAR statement has established a blank screen so that no previously displayed information remains to confuse the presentation. MAPOUT...TEXT is normally used following a CLEAR to output constant information before a repetitive series of operations on variable fields. For example, when paging through a file in which all the records have the same format, descriptive information displayed by an initial MAPOUT...TEXT can remain undisturbed while the data for each record is written using MAPOUT...OUTPUT.

MAPOUT...RECORD and MAPOUT...FIELD are particularly useful in specialised data entry applications where the screen format evolves depending on earlier information submitted by the operator following an initial MAPOUT...ALL, MAPIN...ALL sequence. The sample program provides a good illustration of how these options are used in practice.

By using MAPOUT...PRINT you are able to log the effects of operator data entry and amendment in a straightforward way. Since the option operates on a standard relative sequential print file FD provided by the calling program you are able to arrange the resulting print report exactly as you wish by interspersing the appropriate WRITE NEXT statements with MAPOUT...PRINT statements. Pagination and layout is particularly simple because each MAPOUT...PRINT always uses the number of print lines equal to the depth of the screen area defined when the map was built. Even if you suppress the logging of unused records, as we do in the example program, blank lines are produced in their place so that a constant number of print lines is used.

5.4 The MAPIN Statement
+++ +++++++++++++++++++
The MAPIN statement is used to accept a selection of the input
and update fields defined when the associated map was created.
It is coded:-

MAPIN [ENTRY] mapname option [field]
///////////// /////

e.g:-

MAPIN M-DATA ALL

or:-

MAPIN ENTRY M-DATA FIELD 4

Here mapname labels the map definition on which the statement
//////
operates, option is one of ALL, RECORD or FIELD, and field, an
////// /////
integer literal or the name of a PIC 9(4) COMP variable is the
field number, which must be supplied if the FIELD option is
used. For MAPIN...ALL and MAPIN...RECORD the field number
identifies the first field to be accepted: if omitted the
operation will start with the first appropriate field in the map.

If ENTRY is specified then both input and update fields will be
treated as input fields. That is, each such field will be
cleared before it is accepted, and if <CR> is keyed the default
--
will be used, rather than the existing value. This enables you
to use the same map for inputting new data (using MAPIN ENTRY) or
updating existing data (using a normal MAPIN).

MAPIN is only valid when a previous CLEAR statement has
established formatted mode. An attempt to use MAPIN when the
System Manager is operating in teletype mode will cause your
program to be terminated in error.

Table 5.4 and the detailed descriptions below explain the
functioning of the various MAPIN options. In certain
circumstances any MAPIN statement can be terminated with an
exception, and for this reason the MAPIN statement should be
followed where appropriate by an ON EXCEPTION statement to
introduce the necessary special logic. If an exception is
signalled and no such statement is present your program will be
terminated in error. You should note, however, that the escape
exception (2) can only occur if the application has set $$ESC to
2 to indicate that it wishes to handle the operator's keying of
<ESCAPE>. Similarly the validation exception (3) will only occur

------
if the validation routine you have provided signals exception
condition 3 to terminate the MAPIN operation.

5.4.1 The MAPIN...ALL Option
+++++ +++++++++++++++++++++++
The ALL option causes input and update fields to be accepted. If
the map contains records you must set recno for the appropriate
/////
record set to indicate how many of them, if any, are to be
affected by the operation. Note that if you set recno greater
/////
than the total number of records defined by the map your program
will be terminated in error. If recno is zero, no records will
/////
be processed. If a field number is supplied that field will be
the first to be accepted; earlier fields will be accepted only
if <LINE-FEED> is used to go backwards.

The escape exception (2) will be signalled if the operator keys
++++++
<ESCAPE> and you have previously set $$ESC to 2, to indicate that
------
your program is prepared to handle this condition.

The validation exception (3) will occur if you have supplied a
++++++++++
validation routine and it has signalled exception condition 3 to
indicate that a condition has occurred which precludes further
input.

Following either of these exceptions, the state of the internal
input and update fields in the data areas is unpredictable.

5.4.2 The MAPIN...RECORD Option
+++++ ++++++++++++++++++++++++
The RECORD option should only be used if the map contains
records. Before executing the statement you should set recno for
/////
the appropriate record set to the record number that you wish to
process. Then MAPIN...RECORD causes all input and update fields
of that record to be accepted. Note, however, that if the map
does not contain records, or if you supply an invalid number in
recno, your program will be terminated in error. If a field
/////
number is supplied that field will be the first to be accepted;
earlier fields will be accepted only if <LINE-FEED> is used to go
----
backwards.

The escape exception (2) will be signalled if the operator keys
++++++
<ESCAPE> and you have previously set $$ESC to 2, to indicate that
------
your program is prepared to handle this condition.

The validation exception (3) will occur if you have supplied a
++++++++++
validation routine and it has returned exception condition 3 to
indicate that a condition has occurred which precludes further
input.

Following either of these exceptions the state of the internal
input and update fields in the data areas is unpredictable.

5.4.3 The MAPIN...FIELD Option
+++++ +++++++++++++++++++++++
The FIELD option operates on input and update fields which were
assigned a unique field number by $FORM. The MAPIN...FIELD
statement causes a single field, identified by the number

supplied as its third operand, to be accepted.

If the map contains records and the numbered field is part of a
record, you must set recno for the appropriate record set to
/////
identify the particular occurrence involved before executing the
MAPIN...FIELD statement. If you supply an invalid record number
your program will be terminated in error.

The invalid field number exception (1) will be signalled if
++++++++++++++++++++
either the field number you supply was not defined when the map
was generated, or if the field thus identified was an output
field rather than an input or update field.

The escape exception (2) will occur if the operator keys <ESCAPE>
++++++ ------
and you have previously set $$ESC to 2, to indicate that your
program is prepared to handle this condition.

```
----------------------------------------------------------------
| | | | |
| MAPIN | CONSOLE | USE OF recno FIELD |EXCEPTION CONDITIONS|
 +++++ +++++++ ++++++ ///// +++++ +++++++++++++++++++
| OPTION | ACTION | | ($$COND) |
 ++++++ ++++++ ++++++++
| | | | |
----------------------------------------------------------------
| | | | |
| ALL | Accept | You set recno to a | Escape (2)|
 +++ /////
| | input and | value between 0 and| Validation (3)|
| | update | n*, to indicate how| |
| | fields. | many records are to| |
| | | be processed. | |
| | | | |
| RECORD | Accept | You set recno to a | Escape (2)|
 ++++++ /////
| | input and | value between 1 and| Validation (3)|
| | update | n to indicate which| |
 /
| | fields of | record is affected.| |
| | one | recno must be for | |
 /////
| | selected | the appropriate | |
| | record. | record set. | |
| | | | |
| FIELD | Accept a | If the field is | Invalid |
 +++++
| | single | part of a record | field number (1)|
| | input or | you must set recno | Escape (2)|
 /////
| | update | to a value between | Validation (3)|
| | field | 1 and n to indicate| |
 /
| | identified | which occurrence | |
| | by its | is affected. | |
| | field | | |
| | number. | | |
| | | | |
----------------------------------------------------------------
```

* The quantity n is the maximum number of records defined by the
 /
 map, as returned in recno by a MAPOUT...QUERY operation
 /////

Table 5.4 - MAPIN Statement Processing Summary

++

The validation exception (3) will be signalled if you have
++++++++++
supplied a validation routine and it has returned exception
condition 3 to indicate that a condition has occurred which
precludes further input.

5.4.4 Programming Notes
+++++ ++++++++++++++++++
The MAPIN statement normally causes console input to take place,
although in some case the statement may have no effect (e.g. if
applied to a map with no input or update fields, or when a
MAPIN...ALL or MAPIN...RECORD statement uses a field number or
variant number which does not exist, or specifies a field number
excluded by the selected variant). When input does occur the
affected fields are accepted from left to right, top to bottom,
in the order in which they would naturally be read.

The resulting new value of each field thus accepted depends on
whether the operator supplies actual input or whether he simply
requests to skip or delete the field. It can also depend on your
own validation routine which may alter the standard LOW-VALUES
default established when a field assigned the defaulted attribute
is skipped or deleted. Table 5.4.4 shows how MAPIN itself checks
the input and establishes new values. If a validation routine is
present it will only be entered once these basic checks have been
satisfied and the tentative new value shown in the table has been
set up. The routine can perform additional tests on this new
value and even modify it. If the routine decides to reject the
input the current internal value of the field is not modified and
the operator is prompted again for good input.

```
-------------------------------------------------------------------
| | |
| | OPERATOR ACTION |
 +++++++++++++++
| | |
 --------------------------------------------------
|FIELD ATTRIBUTES| | | | |
 ++++++++++++++++
| |"UNREASONABLE"|"REASONABLE" | SKIP | DELETE |
 ++++++++++++++ ++++++++++++ ++++ ++++++
| | INPUT | INPUT | REQUEST | REQUEST |
 +++++ +++++ +++++++ +++++++
| | | | | |
-------------------------------------------------------------------
| | | | | |
| INPUT, | Note A | new value | Note B | Note B |
 ++++++
| NOT DEFAULTED | | as keyed | | |
 +++++++++++++
| | | | | |
| INPUT, | Note A | new value |new value |new value |
 ++++++
| DEFAULTED | | as keyed |LOW-VALUES|LOW-VALUES|
 +++++++++
| | | | | |
| UPDATE, | Note A | new value |new value | Note B |
 +++++++
| NOT DEFAULTED | | as keyed |unchanged | |
 +++++++++++++
| | | | | |
| UPDATE, | Note A | new value |new value |new value |
 +++++++
| DEFAULTED | | as keyed |unchanged |LOW-VALUES|
 +++++++++
| | | | | |
-------------------------------------------------------------------
```

Note A
++++++
MAPIN applies reasonableness checks on all operator input, which must conform to the picture clause of the field affected and possibly satisfy additional attributes established by $FORM such as A (alpha-filled), D (date) or N (number-filled). If the input is not "reasonable" in this sense, the operator is prompted again for good input following the base line message:-

INVALID * xx...xx
///////

where xx...xx is the first 20 characters (or less) of the

/////////
previous submission.

Note B
++++++
MAPIN will not allow the operator to skip an input field, or delete an update field, which is not defaulted. If this is attempted the operator is prompted again for actual input following the base line message:-

INPUT REQUIRED

Table 5.4.4 - How MAPIN establishes the new value of an accepted field
++

In the simplest type of Screen Formatting operation a MAPOUT...
ALL (or MAPOUT...TEXT, MAPOUT...OUTPUT combination) is simply
followed by a MAPIN...ALL to obtain all the relevant new values
of input and update fields. MAPIN...RECORD is mainly of use in
data entry applications where the number of records (e.g the
number of order lines) is under operator control. After an
initial MAPIN...ALL with recno set to zero, individual records
/////
are processed one by one by MAPIN...RECORD.

If you set $$ESC to 2 the operator can indicate the end of record
data entry simply by keying <ESCAPE> which would cause the
------
outstanding MAPIN...RECORD statement to be terminated with
exception condition 2. Alternatively, as in the example program,
you can make all the input and update fields of the record
defaulted, and test a particular field for its default value to
see whether the operator has signalled the end of record data
entry by keying <CTRL A>.

The MAPIN...FIELD option is particularly useful in file updating
applications where the operator at most requires to change a few
update fields of the display. When you create the map using
$FORM you should clearly number each such field in the text, and
assign the field number as an attribute. After you have output
the map (using, say, MAPOUT...ALL) you prompt for the number of
the field the operator wishes to change using normal teletype
DISPLAY and ACCEPT statements which operate on the base line.
You then use MAPOUT...FIELD to change each indicated field. For
example:-

```
MAPOUT M-UPDATE ALL * SET UP SCREEN
DISP.
DISPLAY "KEY FIELD NUMBER YOU WISH TO CHANGE"
ACC.
ACCEPT FNUM NULL GO TO NEXT
MAPIN M-UPDATE FIELD FNUM
ON EXCEPTION
DISPLAY "KEY VALID NUMBER OR NULL"
GO TO ACC
END
GO TO DISP
NEXT.
```

This example assumes that none of the numbered fields belong to a
record. If this were the case it would be necessary to prompt
for the record number as well as the field number, and set up
recno correctly before executing the MAPIN...FIELD statement. It
/////
also assumes that invalid field number (1) is the only exception

++++++++++++++++++++
suffered by MAPIN...FIELD, implying that the System Manager or
MAPIN handles the operator's keying of <ESCAPE> (i.e. $$ESC is 0
------
or 1) and there is no possibility of your validation routine
terminating the statement with exception condition 3.

```
| | | | |
--------------------------------------------------------------------
| | | | |
| MAPCLEAR | CONSOLE ACTION | USE OF recno FIELD | EXCEPTION CONDITION|
 +++++++++ ++++++++++++++ +++++++///// ++++++ ++++++++++++++++++
| OPTION | | | ($$COND) |
 ++++++ ++++++++
| | | | |
--------------------------------------------------------------------
| | | | |
| ALL | Clear text, | You set recno to a | None |
 /////
| | output, update | value between 0 and| |
| | and input | n, to indicate how | |
 /
| | fields. | many records are to| |
| | | be processed. | |
| | | | |
| OUTPUT | Clear output, | Same as ALL option.| None |
| | update, and | | |
| | input fields. | | |
| | | | |
| TEXT | Clear text | Same as ALL option.| None |
| | fields. | | |
| | | | |
| RECORD | Clear text, | You set recno to a | None |
 /////
| | output, update | value between 1 and| |
| | and input | n to indicate which| |
 /
| | fields (for one| record is affected | |
| | selected record| for the appropriate| |
| | only). | record set. | |
| | | | |
| FIELD | Clear a single | If the field is | Invalid Field |
| | output, update | part of a record | Number (1) |
| | or input field,| you must set recno | |
 /////
| | identified by | to a value between | |
| | its field | 1 and n to indicate| |
 /
| | number. | which occurrence is| |
| | | affected for the | |
| | | appropriate record | |
| | | set. | |
| | | | |
--------------------------------------------------------------------
```

Table 5.5 - MAPCLEAR Statement Processing Summary
 +++

5.5 The MAPCLEAR Statement
+++ ++++++++++++++++++++++

The MAPCLEAR statement is used to set a selection of fields on
the screen to spaces. It does not affect the values of the
associated fields within the program. It is coded:-

MAPCLEAR mapname option [field]
////////////// /////

e.g.

MAPCLEAR M-DATA ALL

or:-

MAPCLEAR M-DATA FIELD 3

Here mapname labels the map definition on which the statement
///////
operates, option is one of ALL, TEXT, OUTPUT, RECORD or FIELD,
//////
and field, an integer literal or PIC 9(4) COMP variable, is the
/////
field number, which must be supplied if the FIELD option is
used. For the ALL, OUTPUT and RECORD options the field number
identifies the first field to be cleared: if absent the
operation will start with the first appropriate field in the map.

MAPCLEAR is only valid when a previous CLEAR statement has
established formatted mode.

5.5.1 The MAPCLEAR ... ALL Option
+++++ ++++++++++++++++++++++++++++

The ALL option causes all fields in the map to be displayed as
spaces, and so can be used to clear the area of the screen
occupied by the map. If the map contains records you must set
recno of the appropriate record set to the number of records to
/////
be processed, or to zero if none are to be modified. If a field
number is supplied only fields starting from that field will be
cleared.

5.5.2 The MAPCLEAR ... OUTPUT Option
+++++ +++++++++++++++++++++++++++++++

The OUTPUT option causes all input, update and output fields to
be displayed as spaces. If the map contains records, you must
set recno of the appropriate record set to indicate how many of
/////
them, if any, are to be affected by the operation, and any text
fields within these records will also be cleared. If you set
recno greater than the total number of records defined by the map

/////
your program will be terminated in error. If recno is zero, no
/////
records will be processed. If a field number is supplied only
fields starting from that field will be cleared.

5.5.3 The MAPCLEAR ... TEXT Option
+++++ +++++++++++++++++++++++++++++
The TEXT option causes all the text fields in the map, together
with the $$RECNO and $$DATE fields if defined, to be cleared from
the screen by displaying them as spaces. If the map contains
records you must set recno of the appropriate record set to the
/////
number to be processed, or to zero if none are to be modified.

5.5.4 The MAPCLEAR ... RECORD Option
+++++ ++++++++++++++++++++++++++++++
The RECORD option should only be used if the map contains
records. Before executing the statement you must set recno to
/////
the record number that you wish to process. Then MAPCLEAR ...
RECORD causes all fields of that record to be displayed as
spaces. Your program will be terminated in error if you supply
an invalid record number, or if the map does not contain
records. If a field number is supplied only fields starting from
that field will be cleared.

5.5.5 The MAPCLEAR...FIELD Option
+++++ +++++++++++++++++++++++++++
The FIELD option operates on input, output or update fields which
were assigned a unique field number when the map was created by
$FORM. The MAPCLEAR...FIELD statement causes a single field,
identified by the number supplied as its third operand, to be
displayed as spaces.

If the field is within a record, you must set recno of the
/////
appropriate record set to identify the occurrence involved before
executing the MAPCLEAR...FIELD statement. If you supply an
invalid record number your program will be terminated in error.

The invalid field number exception (1) will be signalled if the
++++++++++++++++++++
field number you supply is not defined in the map.

5.5.6 Programming Notes
+++++ +++++++++++++++++
The MAPCLEAR statement normally causes selected fields on the
screen to be cleared, although in some cases it may have no
effect. For example, MAPCLEAR...OUTPUT will not affect a screen
consisting entirely of text, and a MAPCLEAR...ALL, MAPCLEAR...
OUTPUT or MAPCLEAR...RECORD statement will be ignored if it uses
a field number or variant number which does not exist, or if it
specifies a field number excluded by the selected variant.

The MAPCLEAR...ALL option can be used to clear a particular map
from the screen, without affecting fields not defined within that
map. This is particularly useful in programs which use several
different maps simultaneously, each defining a different area of
the screen, in order construct complex screen layouts.

The other options are mainly employed when you want to use the
map currently displayed to input another set of data. For
example, you can use MAPCLEAR...OUTPUT to clear all the input,
output and update fields, leaving the text fields unchanged.
This will give a smoother, less jerky display than CLEAR followed

by MAPOUT TEXT, which will cause all the text fields to disappear briefly.

6. Advanced Screen Presentation

6.1 Validation Routines

A validation routine enables you to check each input and update field as it is entered, and optionally to display derived fields to confirm that the correct information has been keyed. It can be used to supply default values when an input field is skipped or an update field is deleted. It can terminate the current MAPIN operation by returning a special exit code, and you can also use it to display a field with the user conversion attribute, in a different format from that held in the record. You can also call a validation routine prior to an accept being issued.

You indicate that you wish to provide a validation routine by keying "Y" to the validation code option. You either indicate the validation routine by supplying its entry name to $FORM's validation prompt when the map is created, or by supplying its entry point in the map definition (MD). If the validation routine was supplied during map creation then the resulting map will contain a global referencing the routine, which will normally be included in any linkage edit involving the map.

In most applications, there will be some fields which need special validation as they are keyed. Also, you will often want to display derived fields as a confirmation, for example the account name corresponding to the account number just entered, or the total of an order line. You could, of course, input each field separately, but this not only is likely to be slower and require more code than using a validation routine, but also means that <LINE-FEED> cannot be employed to go back to correct a previous field, as this only applies to fields accepted by a single MAPIN statement.

You may also want to display some help information prior to a field being accepted to assist the user. This can also be done within a validation routine.

Your validation routine is called to validate a field from the mapping routine when a MAPIN request accepts a field which was assigned a validation code when the map was built and the reasonableness checks performed by the mapping routine itself are satisfied (if the V attribute is not used). Thus the routine can be called:

- when the operator supplies a new (reasonable) value for a field, or skips an update field indicating that its current value is to be used or;

- when the operator requests that a default value be established either by skipping a defaulted input field, or by deleting a defaulted update field;

- when the operator supplies an unreasonable value for the field provided the V attribute is set.

When the validation routine is called to validate an accept, the field validation block passed to your routine as a parameter indicates whether or not a default was requested and allows you to access the new value of the field tentatively established by MAPIN as defined in Table 5.4. You can either perform extra validation on the field or possibly update the standard default value, LOW-VALUES, by supplying your own special value.

If the new value is satisfactory you need only exit normally to the mapping routine to cause processing to continue. The new value is displayed on the screen and is eventually made available to the controlling application program. If it is LOW-VALUES it will be output as blanks unless the field involved is computational and the B (blank when zero) attribute has not been specified, in which case a zero value will be displayed.

If you find the new value to be invalid you must return control using the EXIT WITH condition statement to signal an exception and cause the operator to be prompted again for correct input. Depending on the condition you supply you can either cause mapping to output a standard error message, or you can provide your own message instead. Alternatively, you can cause the controlling MAPIN statement to be terminated with a validation exception.

When the validation routine exits normally, you can specify that another field should be displayed to allow final confirmation of the new value. Alternatively, you can display fields directly using MAPOUT statements. If the new value is incorrect then the operator may either key <ESCAPE> to cause an incomplete MAPIN of several fields to be terminated with an exception or re-select the particular field. In the first case, the controlling application should handle the exception and repeat the operation.

If you need to establish a special default you simply move your own value to the new value field before signalling normal completion. The value you have supplied will then be displayed and made available to the application in the normal way. When specifying a default for a numeric field you must ensure that it conforms with the picture clause of that field, otherwise your program will be terminated in error.

6.1.1 Coding the Entry Statement

The first executable instruction of your validation routine must be an ENTRY statement of the form:

ENTRY *entry-name* USING FV

The *entry-name* must be the same as that supplied to $FORM's validation prompt when the map or maps requiring to use the routine are created.

The parameter, FV, must be the name of a level 01 group item coded in the linkage section to define the field validation block supplied when the routine is called.

6.1.2 Using the Field Validation Block

The field validation block must be read-only as far as your routine is concerned and should be defined as follows:

```
01 FV
 02 FVCODE PIC 9(2) COMP * Field validation code
 02 FVDEF PIC 9(2) COMP * Default flag
 * 1 = Default Requested
 * 0 = Input supplied
 02 FVX PIC 9(4) COMP * Field index
 02 FVAREA PIC PTR * Area base
 02 FVFIELD PIC PTR * Field base
 02 FVPTR PIC PTR * Pointer to area list in MD
 02 FVCON PIC 9(2) COMP * Confirmation field number
 02 FVCVAR PIC 9(2) COMP * Confirmation variant number
 * Formatting validation
 02 FVVER PIC 9(2) COMP * User validation called for
```

* inputs that fail screen
* formaating validation (V
* attribute required

FVCODE, the validation code, is the value, between 1 and 99,
++++++
assigned to the field by $FORM when the map was created (the routine will not be called for fields for which no such code was assigned). FVCODE is used to identify the particular test to be performed by the routine and indicate which field is involved. If you allocate codes sequentially starting at 1 you can write:-

```
GO TO DEPENDING ON FVCODE
TO test 1
...
TO test n
/
```

to route control rapidly to the section responsible for a particular test.

FVDEF, the default flag, will be set to 1 to indicate that the
+++++
operator has requested that a default value be supplied, either by skipping a defaulted input field, or by deleting a defaulted update field. In other cases FVDEF will be zero and your routine will be expected to validate the new value with which it is supplied.

FVX, the field index, must be used when a multi-valued field is
+++
involved. When the field is within a record, it is set to the record number. If the field is to be referenced using its data name you must select the correct occurrence by treating it as an indexed variable coded as:-

```
data name (FVX)
/////////
```

There is an example of this below.

FVAREA, the area base address, must be employed when you need to
++++++
access the new value using the data name of the affected field. This is the procedure to follow when the validation code and corresponding test apply to only a single named field of the map. In this case the area containing that field should be described in the linkage section of your routine by including the same copy book that you supplied to $FORM when the map was created. You can then access the new value prepared by the mapping routine by basing this area on FVAREA and then

referencing the field in the normal way, using FVX where appropriate. For example, suppose the area described by copy book EX is:-

```
01 EX
02 EXNAME PIC X(16) * SHORT NAME
02 EXACCT PIC 9(8) COMP * ACCOUNT NUMBER
02 EXMSTA OCCURS 12 PIC X * MONTHLY STATUS CODE TABLE
```

Then if EXACCT was assigned validation code 1, test 1 might begin:-

```
BASE EX ON FVAREA
DIVIDE 11 INTO EXACCT GIVING QUOTIENT
```

```
| | |
------------------------------------------------------------
| | |
| STATEMENT | EFFECT |
+++++++++ ++++++
| | |
------------------------------------------------------------
| | |
| EXIT WITH 1 | Mapping outputs its standard base line error|
| | message:- |
| | |
| | INVALID * xx...xx |
///////
| | |
| | then accepts the field again. |
| | |
| EXIT WITH 2 | You have already output the base line error |
| | message so mapping simply accepts the field |
| | again. |
| | |
| EXIT WITH 3 | The current MAPIN operation is terminated |
| | with a validation exception (3). The field |
| | being accepted is not updated. |
| | |
| EXIT WITH 4 | The field is accepted, and then all |
| | remaining incoming fields are skipped as if |
| | <CTRL A> had been keyed. (Update fields |
| | retain their existing values, input fields |
| | are given default values if available.) |
| | |
| EXIT WITH 5 | The current MAPIN operation is terminated |
| | with a validation exception (3). The field |
| | being accepted is updated. |
| | |
------------------------------------------------------------
```

Table 6.1.3 - Exceptions Signalled from the Validation Routine
++

Test 2, on a selected occurrence of EXMSTA, might start:-

```
BASE EX ON FVAREA
IF EXMSTA(FVX) GREATER "E" GO TO ERROR
```

(The BASE statement is described in section 5 of the Global Cobol Language Manual.)

Note that only the new value of the field just input can be accessed in this way: any attempt to access other fields in the area, e.g. EXNAME in the above example, will give unpredictable results and may cause a program check.

FVFIELD, the field base address, points to the value just input
+++++++
and can be used as an alternative to FVAREA. It is particularly useful when a test is to be applied to a number of different fields rather than a single named field. You must describe the field format by means of a 77 level item in the linkage section. Then you can access the new value input by the operator by basing this item on the FVFIELD pointer. For instance, suppose in the previous example we had wished test 1 to apply to all 8 digit account numbers used by the application. The field format could be defined by a linkage section declaration:-

```
77 ACACCT PIC 9(8) COMP
```

and then test 1 would begin:-

```
BASE ACACCT ON FVFIELD
DIVIDE 11 INTO ACACCT GIVING QUOTIENT
```

FVPTR, the area list pointer, can be used in applications where
+++++
the validation routine is separate from the program containing the data areas to enable fields in these areas to be examined or updated. You should define a level 01 item in the linkage section with a pointer for each area defined in your MD. For example, if the MD contains the following clause:-

```
AREAS "AC" "EX"
```

then the 01 item in the linkage section should be coded:-

```
01 PT
02 PT-AC PIC PTR * POINTER TO AC AREA
02 PT-EX PIC PTR * POINTER TO EX AREA
```

If it were then desired to compare account numbers during input of EXACCT the comparison would be coded:-

```
BASE EX ON FVAREA
BASE PT ON FVPTR
BASE AC ON PT-AC
IF EXACCT NOT EQUAL ACACCT...
```

FVCON, the confirmation field number, can be used when you wish
+++++
to select a field to be displayed to enable confirmation of the
new value of the current field. The contents of the confirmation
field itself should be set using the area list pointer, FVPTR, to
access it as described above. For example, if we wish to use the
short name for confirmation of the new value for the account
number and the short name has been numbered 3 in the map then the
following code is required:-

```
MOVE 3 TO FVCON
```

Note that you could obtain the same effect by coding:-

```
MOVE FVX TO RECNO * if in a record
MAPOUT MAP FIELD 3
```

FVCVAR, the confirmation variant, is similar to the confirmation
++++++
field number described above, but instead identifies a group of
confirmation fields identified by a variant number.

FVVER, this flag is set to 1 if the field had the V attribute and
+++++
the input keyed by the user failed screen formatting internal
checking. For example, if the V attribute was set for a numeric
field and the user keyed in a character, then the validation
routine will be called for that field but FVVER will be set to 1.

$$LENG will be set to the length of the input keyed when the
++++++
validation routine is called. If it is zero this means that <CR>
(or a control response) has been keyed on its own.

$$EOF is established as for an ACCEPT operation when the
+++++
validation routine is called, and can be used to test what key
was used to terminate the response.

6.1.3 Returning Control to the Mapping Routine
+++

If the operator input is valid you simply code a normal EXIT
statement to return control to the mapping routine. If you wish
to establish a special default value when FVDEF = 1 you must
overwrite the new value provided by mapping with your own special
value prior to returning control.

If the operator input is incorrect, or if you wish to terminate
the current MAPIN operation, you should use the EXIT WITH
condition statement to signal one of four possible exception

conditions, as indicated in table 6.1.3.

Before issuing the EXIT WITH 2 statement you must have written your own error message to the base line using a normal DISPLAY statement. For example:-

```
DISPLAY "MONTHLY STATUS MUST BE A, B, C, D OR E"
EXIT WITH 2
```

Note that if you use EXIT WITH 3 to terminate the MAPIN operation but wish to update the field being updated, the validation routine must copy the new value to the field before exiting.

```
PROGRAM VA100
DATA DIVISION
* WORKING STORAGE CONTAINS ONLY INTERNAL WORK FIELDS
*
77 QUOTIENT PIC 9(7) COMP
77 MULTIPLE PIC 9(8) COMP
77 ACACCT PIC 9(8) COMP

LINKAGE SECTION
* FIRST DEFINE FIELD VALIDATION BLOCK PASSED AS PARAMETER
01 FV
 02 FVCODE PIC 9(2) COMP * Field validation code
 02 FVDEF PIC 9(2) COMP * 1 = Default, 0 = Input
 02 FVX PIC 9(4) COMP * Index
 02 FVAREA PIC PTR * Area base
 02 FVFIELD PIC PTR * Field base
 02 FVPTR PIC PTR * Pointer to areas list

* NEXT ANY AREAS CONTAINING FIELDS VALIDATED BY NAME
01 EX
COPY EX
 02 EXNAME PIC X(16) * Short name
 02 EXACCT PIC 9(8) * Account number
 02 EXMSTA OCCURS 12 PIC X * Monthly status table
* FINALLY ANY GROUPS FOR FIELDS VALIDATED BY TYPE
77 ACN PIC 9 (8) * Account number

PROCEDURE DIVISION
ENTRY CSTVAL USING FV
 GO TO DEPENDING ON FVCODE
 TO TEST-1
 TO TEST-2
*
*CHECK 8-DIGIT ACCOUNT NUMBERS ARE MULTIPLES OF 11
TEST-1.
 BASE ACN ON FVFIELD
 MOVE ACN TO ACACCT
 DIVIDE 11 INTO ACACCT GIVING QUOTIENT
 MULTIPLY QUOTIENT BY 11 GIVING MULTIPLE
 IF MULTIPLE NOT = ACACCT EXIT WITH 1
 EXIT
*
* CHECK MONTHLY STATUS BETWEEN "A" AND "E". ESTABLISH "E" AS DEFAULT
TEST-2.
 BASE EX ON FVAREA
 IF FVDEF POSITIVE
 MOVE "E" TO EXMSTA(FVX) * Establish default
 ELSE
 IF EXMSTA(FVX) > "E"
 DISPLAY "MONTHLY STATUS MUST BE A, B, C, D OR E"
 EXIT WITH 2
 END
 END
 EXIT
ENDPROG
```

Figure 6.1.4 - Fragment of Example validation program

++

6.1.4 An Example Validation Program
++++++++++++++++++++++++++++++++++++
Figure 6.1.4 shows a fragment of an example validation program.
Unfortunately there is only enough room to show two tests, but
they serve to illustrate most of the points that have been made.
There are further, more complex examples in Appendix A.

Test 1 applies to all 8-digit account numbers used by the
application, and is therefore a test by field type, rather than
by field name. Each account number is described by AC field
coded in the linkage section. When account number fields are
processed by $FORM they are not assigned the defaulted option, so
+++
the operator is forced to always supply a value. The test simply
checks that this value is a multiple of 11 and, if this is not
the case, returns to the mapping routine signalling exception
condition 1 so that the operator is prompted again following the
standard error message:-

INVALID * xx...xx
///////

which appears on the base line. It does allow the special
response of "*" to indicate account number 999999. This field
has the V attribute set.

Test 2 applies to a named multi-value field, EXMSTA, which forms
a status table, containing an entry for each month of the year.
The EX copy book defining EXMSTA is included in the linkage
section of the validation routine in order to describe the area.
(The same book will have been made available to the program
containing the map definition and to $FORM itself.) The example
assumes that when the map was built EXMSTA was established as an
input field and assigned the defaulted attribute in order that,
whenever the operator skips the field, the value "E" is set up as
a special default. The validation routine limits the range to
be A to E. If an erroneous value is input the routine displays
an error message on the base line and causes mapping to prompt
the operator again by signalling exception condition 2.

6.1.5 Programming Notes
++++++++++++++++++++++
A validation routine may use any MAPOUT or MAPCLEAR statements
except for MAPOUT ...PRINT, and may reference either the current
map or a different one. It may also use a MAPIN statement
provided that the field being mapped in the validation routine
does not itself have a validation routine as this would involve a
recursive call of the routine which processes MAPIN. (Your
program will be terminated in error if you attempt this.)
However, the validation routine may use ACCEPT or ACCEPT...LINE
statements to input information. You can alter the values of

recno and varno before performing a MAPOUT or MAPCLEAR on the map used by MAPIN; this will not affect the current MAPIN operation being processed, as it uses internal copies of these fields, containing the values originally supplied.

If your validation routine is part of the main program, you must be careful to use the new value pointed to by FVFIELD, rather than the old value in the associated data field, which is not updated with the new value until after the validation routine has relinquished control.

```
PROGRAM VAL200
DATA DIVISION
EXTERNAL SECTION SM$BUF
01 UVBUF
 02 UVBUFN PIC 9
 02 FILLER PIC X(79)
LINKAGE SECTION
01 FV
 02 FVCODE PIC 9(2) COMP * FIELD VALIDATION CODE
 02 FVDEF PIC 9(2) COMP * 1 = DEFAULT, 0 = INPUT
 02 FVX PIC 9(4) COMP * INDEX
 02 FVAREA PIC PTR * AREA BASE
 02 FVFIELD PIC PTR * FIELD BASE
 02 FVPTR PIC PTR * POINTER TO AREAS LIST
*
77 UVFIELD PIC PTR * USER VALIDATION. POINTER TO FIELD IN RECORD
77 UVH REDEFINES UVFIELD PIC 9(2) COMP
77 UVCODE PIC 9(2) COMP * USER VALIDATION CODE
77 L-COUNT PIC 9 COMP
*
01 L-UVX
 02 L-UVN1 PIC 9
 02 L-UVX2 PIC X(2)
77 L-UVC1 REDEFINES L-UVX PIC 9 COMP
*
PROCEDURE DIVISION
ENTRY VA-VAL USING FV
 ON OVERFLOW GOTO ENT2 *ENTRY WITH 2 PARAMETERS
 GOTO VA100
ENT2.
ENTRY USING UVFIELD UVCODE
 GOTO VA500
*
* CHECK THAT THE USER HAS KEYED EITHER A SINGLE DIGIT OR THE WORD
* "ALL". THE FIELD IS TO BE HELD IN THE RECORD AS A PIC 9 COMP
* FIELD, SO CONVERT IT TO A COMP FIELD. IF "ALL", CONVERT TO 0.

VA100.
 BASE L-UVX ON FVFIELD
 IF L-UVX = "ALL"
 MOVE 0 TO L-UVC1
 EXIT
 END
 IF L-UVN1 NOT NUMERIC EXIT WITH 1
 IF L-UVX2 NOT SPACES EXIT WITH 1
 MOVE L-UVN1 TO L-UVC1 * CONVERT TO COMP FORMAT
 EXIT
*
* ENTRY TO EITHER THE PRE-ACCEPT VALIDATION OR THE USER CONVERSION
* VALIDATION.
*
```

```
VA500.
 IF UVH = -1 * PRE ACCEPT VALIDATION
 DISPLAY "Key number or ALL for all" * DISPLAY HELP
 EXIT
 END
 *
```

Figure 6.1.5 - User Conversion Example

++

```
*
* TAKE THE NUMBER HELD IN THE RECORD FIELD AND BUILD THE DISPLAY
* FORMAT IN THE USER VALIDATION BUFFER. IF VALUE IS 0, DISPLAY
* "ALL", OTHERWISE DISPLAY THE SINGLE DIGIT NUMBER.
*
 BASE L-COUNT ON UVFIELD * GET RECORD FORMAT
 IF L-COUNT = ZERO * IF RECORD FORMAT = 0
 MOVE "ALL" TO UVBUF * DISPLAY AS "ALL"
 ELSE
 MOVE "" TO UVBUF
 MOVE L-COUNT TO UVBUFN * OTHERWISE DISPLAY NUMERIC COUNT
 END
 EXIT
ENDPROG
```

Figure 6.1.5 (cont.) - User Conversion Example
+++

Note that during program development you can legitimately omit validation routines when maps are linked if you need to start testing without them. The mapping routine sensibly avoids calling a missing routine. You must be careful, of course, to restrict your input to valid data and avoid skipping or deleting fields which require special default values to be set up.

6.1.5 User Conversion and pre-accept Validation

++

If you have given a field the user conversion attribute (U) to display the field in a different format from the original record format, the validation routine is used to do the conversion.

As for all fields, the validation routine is called from the mapping routine when a MAPIN request accepts a field. However, for user-validated fields, the mapping routine does no validation on the field but simply accepts characters up to the defined display length into the field area (the area based on pointer FVFIELD). It is up to the user to do all the validation on this field and to perform any necessary format conversion such as numeric conversion for fields to be held in the record in numeric form.

The validation routine is called with 2 parameters instead of 1 whenever the field is to be displayed. These parameters are a
++
pointer to the field in the record and a 1-byte validation code. The display format should be built in the global field SM$BUF, accessed via an external section. (The display length used is that which was supplied for this field when defining the map using $FORM.

The example opposite shows how a PIC 9 COMP field with the user conversion attribute and a display length of 3 can be used to display and accept a single digit number or the word "ALL" if the value is 0.

If you have given a field the pre-accept attribute (H) to perform pre-accept validation then the validation routine can be used to display extra information such as help information.

The validation routine is called by the mapping routine before a MAPIN request accepts a field. The validation routine is called with two parameters as with a user conversion call to the validation routine. The first parameter is a PIC 9(2) COMP field which is always set to -1 and the second parameter is the validation code.

The example shows how the pre-accept validation can be used to show some help information on the baseline.

6.2 Variants
+++ ++++++++

Often a map contains some fields which are not required by all the transactions which use it. Alternatively, there may be a group of fields within it which are always processed together. You can, when you define the map, identify each such group of fields as a variant, and either exclude or select particular variants on any MAPIN, MAPOUT or MAPCLEAR operation. This may be more convenient than introducing multiple maps or using field numbering. In particular, it allows you to input a selection of fields using a single MAPIN statement, allowing <LINE-FEED> to be
----
used to move back and correct fields. You can also, from within a validation routine, alter the fields to be input subsequently.

You must specify the variant numbering option when the map is defined. This will cause you to be prompted for a variant number, 0 to 99, whenever you supply the definition of a new field, and also for each new text field. Variant number zero, the default, means that the field is to be processed by all map processing statements. Other numbers indicate variants, which can be included or excluded as specified below. You must give all fields belonging to a particular variant the same variant number.

6.2.1 Using a Table of Selected Variants
++

Tables of selected variants provide a powerful and flexible method of handling variants. To use a table of selected variants you must set the variant number to -1 and supply a table of selected variants using the optional second parameter on the VARIANT statement. This second parameter is either the name in quotes of the table to be used, or, more usually, a symbol which labels a PIC PTR field within the MD. The program must initialise this pointer to address the table to be used before executing any screen formatting statements. If you always intend to supply a table, you can code the variant number as -1 in the VARIANT statement.

A table of selected variants is composed of any number of PIC S9(2) COMP entries, containing the numbers of variants to be processed, and is terminated by an entry set to -1. Entries containing values which do not correspond to a variant in the map are ignored, and so you can "remove" a variant from the table by setting the entry to an impossible value, such as 100. Note that fields with variant number zero are only processed if a zero entry is included in the table.

Thus, for example, if you want to select variants 1, 2, 7 and 23 you might define a table:-

```
01 TABLE
02 T-VAR OCCURS 5 PIC S9(2) COMP
VALUE 1
VALUE 2
VALUE 7
VALUE 23
VALUE -1
```

If the MD had been coded as:-

```
MD M-DATA MAP "ORDATA"
VARIANT -1 VARTAB
```

you could input these variants using the statements:-

```
POINT VARTAB AT TABLE
MAPIN M-DATA ALL
```

If you change the address of the variant table within a validation routine, this will not affect the remainder of the current MAPIN operation, as the address of the table is preserved inside the mapping routine. Thus you can supply different tables for use with MAPOUT and MAPCLEAR statements issued from within a validation routine.

If, however, a validation routine changes the contents of the table being used by the current MAPIN operation, this will affect subsequent processing. This feature is very useful when the map contains variants which are only to be processed for certain values of one of the earlier fields. The validation routine for this field should set the appropriate entry in the table to either the variant number or to some impossible value such as 100, according to whether the variant is to be processed or not.

Note, however, that you must not alter the table to include or exclude fields prior to the one currently being processed: if you do, <LINE-FEED> will not work correctly and other inconsistencies may arise.

6.2.2 Screen Overlays
++++++++++++++++++++

If you have a complex map with overlapping fields, the overlay facility of $FORM allows these to be displayed and amended independently. Variant numbers 1-39 are always displayed, but you can then additionally select any one of the 6 decades 40-49, 50-59, 60-69, 70-79, 80-89 or 90-99. Thus you should use 1-39 for fields that are not overlapped and use variant numbers from different decades in the range 40-99 for overlapped fields.

Note that the overlay facility is purely for convenience of editing; it is for you to select the "overlay" required at run time by using a suitable table of variants.

6.2.3 Selecting by Variant Number
++++++++++++++++++++++++++++++++

Note: this facility is included only for compatibility with
++++
earlier versions of $FORM. Selecting by variant number is not
+++

recommended programming practice and you should use the more powerful and flexible table of selected variants instead.

To select or exclude variants, you must include the VARIANT statement in the associated MD, so that you can establish the variant number associated with an operation.

If the number is zero, its initial value, all variants are processed by any map processing statement. If, however, the number is set to a value between 1 and 99 only selected variants, as defined in table 6.2, will be processed by a subsequent MAPOUT, MAPCLEAR or MAPIN statement which does not use the FIELD or QUERY option. (When the FIELD option is used, the specified field is always processed, irrespective of variant number, and the QUERY option always returns the total number of records in the map, whether or not variants are employed.)

Variant Number in MD (V)	Variants Selected
0	All
1 to 19 inclusive	0 and V
20 to 49 inclusive	0, 20 to V inclusive
50 to 99 inclusive	0, 1 to 19 inclusive, 50 to V inclusive

Table 6.2 - Selection of Variants

6.3 Separately Loaded Maps

```
+++ +++++++++++++++++++++
```

Normally a map definition is associated with a single map which is linked with the program containing the MD, and loaded as part of that program. The linking process causes the relevant pointer in the MD to be resolved to address the entry point of the map, the name of which is specified in quotes in the MAP clause.

By using the symbol option when coding the MAP clause you can delay specifying which map is to be used until run-time. In this case the map need not be part of the program containing the MD and can be loaded separately. If the map uses a validation routine which is part of the root program then the map must be linked as an overlay of the root so that the address of the validation routine in the map can be resolved. Otherwise, the linker must be used to combine the map file and, optionally, its validation routine, into a program file whose entry point is that of the map itself. For example the following dialogue could be used to create program file ORCUST containing the map file, C.ORCUST, and the sample validation program, C.ORVAL:-

```
GSM READY:$LINK
-----
$44 LINK:#3C00
-----
$44 LINK:ORCUST UNIT:101
------ ---
$44 LINK:ORVAL UNIT:<CR>
----- --
$44 LINK:<CR>
--
$44 PROGRAM:<CR> UNIT:<CR>
-- --
$44 MAP UNIT:<CR>
--
$44 LINK OPTION:<CR>
--
$44 LINKAGE EDITING
$44 LINKAGE EDIT COMPLETED
GSM READY:
```

Note how the map file is linked first so that its entry point becomes the entry point of the resulting program file.

An application containing an MD requiring such a map simply uses a normal Global Cobol LOAD statement to bring the program file into memory and then resolves the map-id field from the entry point address returned in $$EPT. For example, suppose the map definition to be:-

```
MD SEPARATE MAP P-MAP
```

```
RECORD R-MAP
AREAS "OR" "WK"
```

and assume the name of the map program file to be in the PIC X(8) field MAPPROG. Then the following code loads the map and introduces it to the map definition:-

```
LOAD MAPPROG
ON EXCEPTION
(Logic when the map program file is not available)
END
MOVE $$EPT TO P-MAP
```

The map will be brought into memory starting at the address specified when the program file was linkage edited (i.e. #3C00 in the example).

Separately loaded maps are employed when storage is critical or it is not known until run-time which of a series of maps will be required. Special applications may exploit the fact that such maps can be created or modified without the need to alter the controlling program. Normally, however, it is preferable to link the map with the program requiring it since the coding and preparation involved is simpler and the overhead of a separate load operation is avoided.

6.4 Fully Symbolic Map Definitions
+++ ++++++++++++++++++++++++++++++

Advanced applications using separately loaded maps may require to specify other information parameterising the mapping operation dynamically. In the most flexible scheme you may need to supply the addresses of the map and up to five data areas at run-time. To achieve this you need to code a fully symbolic map definition such as:-

```
MD SYMBOLIC MAP P-MAP
RECORD R-MAP
VARIANT V-MAP
AREAS P-A1 P-A2 P-A3 P-A4 P-A5
```

This example establishes pointer fields named P-MAP, P-A1...P-A5, together with a recno field named R-MAP, which you will be
/////
responsible for setting up prior to executing a MAPIN statement using this MD. The table below shows the initial values assigned to the symbolic fields when the map definition is compiled:-

```
| | | |
--------------------------------------------------
| | | |
| FIELD | USAGE | INITIALISED AS |
+++++ +++++ +++++++++++++
| | | |
-----------------------------------------------
| | | |
| P-MAP | Address of map | #FFFF |
| | | |
| R-MAP | Current record number | 0 |
| | | |
| V-MAP | Current variant number | 0 |
| | | |
| P-A1 | Address of first area | #FFFF |
| | | |
| . | | . |
| . | | . |
| | | |
| P-A5 | Address of fifth area | #FFFF |
| | | |
-----------------------------------------------
```

Table 6.4 - MD Initialisation when the Symbol Options are Used
++

These initial values have particular significance as far as the mapping routine is concerned. If the address of the map is still #FFFF when it is entered, the routine will realise you have failed to resolve the map pointer and terminate your program in error. Similarly if a map processing statement involves an output, update or input field, and the pointer to the area containing that field is set to #FFFF, your program will be terminated due to your failure to supply the necessary area.

A MAPOUT or MAPIN statement using the RECORD option will fail if you have not set up a valid record number. The same applies to the FIELD option, if the field involved is part of a record.

6.5 Special Global Data Areas
+++ ++++++++++++++++++++++++

The mapping routine contains several global data areas which can be modified by the user program. The end of field routine address allows an intercept routine to be established to analyse the terminators keyed, and to take special action if required. The remaining globals are used to alter the messages used by screen formatting; they are particularly useful when working in languages other than English.

To access one of these areas, you must include an external section in your program whose name is the global, and which contains a single 77 level definition of the field. For example, to access the PIC PTR field SM$EOF you could code:-

```
EXTERNAL SECTION SM$EOF
```

```
77 P-EOF PIC PTR
```

in a module which is linked with the mapping routine. You can then access P-EOF as if it were part of your own working storage.

6.5.1 SM$EOF, End of Field Intercept Routine

++

Global SM$EOF is the address of a PIC PTR field containing the address of a routine which is called after each field is keyed in response to a MAPIN statement, but before any special action is taken as a result of the terminator, and before the validation routine is called. It can therefore be used to change the meanings of the terminators, by translating the valued keyed, or to perform special processing when certain terminator keys are used. The field is initially set to point at an EXIT statement within the mapping routine, so that no special processing takes place.

The routine is not passed any parameters, and can be coded as a section within a program linked with the mapping routine. It must, of course, remain resident while the mapping routine is in use. When it is called the PIC 9(2) COMP system variable $$TER contains the value of the terminator byte, $$EOF PIC X contains the interpretation, and $$LENG PIC 9(4) COMP contains the number of characters keyed excluding the terminator, just as after a normal accept. The routine can, if required, modify the value of $$TER as indicated in table 6.5.1, to change the action taken by the mapping routine.

The routine may also set $$LENG to zero if it wants to cause the graphics characters keyed to be ignored. The value of $$EOF is not used by the mapping routine, although it may be tested by your validation routine.

The end of field routine can signal exception 3 to exit the MAPIN routine with a validation error or can signal exception 4 to cause the field to be re-input.

6.5.2 SM$Y-N, Yes/No Fields

+++++++++++++++++++++++++++

Global SM$Y-N is the address of a PIC X(3) field which contains the values corresponding to Yes and No which can be keyed for a field with the Y attribute. It is initially set to the value "Y/N". The "/" character is only used when displaying the help message.

$$TER	KEYSTROKE	ACTION
1	<CTRL A>	All subsequent fields are skipped.
2	<CTRL B>	Field is set to default value.
4	<CTRL D>	Go back to 1st field (but retain values keyed so far as defaults).
10	<LINE-FEED>	If $$LENG = 0, go back 1 field, current field unchanged; if $$LENG >0, ignore input, re-display and re-input.
13	<RETURN>	Accept current input.
27	<ESCAPE>	$$ESC=0 Terminate program; $$ESC=1 Treat as <RETURN>; $$ESC=2 Terminate MAPIN with exception.
other		Treat as <RETURN>.

Table 6.5.1 - Field Termination Action

6.5.3 SM$INV, "INVALID" Message

Global SM$INV is the address of a PIC X(9) field which contains the messages displayed when an invalid value is keyed. It is initially set to "INVALID *". When displayed it is followed by a space and the invalid value keyed.

6.5.4 SM$REQ, "INPUT REQUIRED" Message

Global SM$INV is the address of a PIC X(14) field which contains

the message displayed when <CR> is keyed to an input field which
--
does not have a default. It is initially set to "INPUT
REQUIRED".

6.5.5 SM$BUF, User Conversion
++++++++++++++++++++++++++++++
Global SM$BUF is the address of a PIC X(80) field which contains
the display format of the displayed field, for use with user
converted fields.

6.5.6 SM$XF, Current field entry

++++++++++++++++++++++++++++++++

Global SM$XF is the address of the current field entry block which contains the line and column of the field currently being processed together with the field number. The current field entry block is defined as follows:-

```
01 XF
02 FILLER PIC X * reserved
02 XFNO PIC 9(2) COMP * Field number
02 FILLER PIC X * Reserved
02 XFLI PIC 9(2) COMP * Screen Line
02 XFCO PIC 9(2) COMP * Screen Column
```

This block is a read only block which can be inspected during field validation routines and can be useful when using the same validation code for more than one field.

6.6 Window Management using WINDO$ and WRES$
+++ +++

WINDO$ and WRES$ are two routines which are provided to simplify the use of pop-up windows in conjunction with screen formats. They are used to save the data on a particular part of the screen, where a screen format is to be displayed, and to replace it subsequently.

6.6.1 Preparing a window using WINDO$
+++++++++++++++++++++++++++++++++++++

You call WINDO$ to prepare a window for a screen format before displaying it using a MAPOUT statement. The call is coded as follows:-

CALL WINDO$ USING md save-area
////////////

where md identifies a Map Definition for the screen format to be
//
displayed, and save-area is a data area for the information
/////////
currently on the screen to be saved in. The save area must be defined as follows:-

01 save-area
/////////
03 save-size PIC 9(4) COMP
/////////
VALUE size
////
03 save-data PIC X(size)
///////// ////

The size of the save area is specified in the control block itself. The area should be large enough to the screen information for the part of the screen which will be overlaid by the screen format. This will be the number of characters overlaid by the screen format (depth multiplied by width) multiplied by three to account for the colour and attribute information.

6.6.2 Save a window area using WINDO$
+++++++++++++++++++++++++++++++++++++

When it is called, WINDO$ saves all screen information from the area where the screen format specified in the MD will be displayed in the save area passed. It then clears the area where the screen format is to be displayed. WINDO$ uses CEOL$ to clear the lines in the window area if the overlaying screen format goes up to the right-hand edge of the screen Therefore, for this purpose WINDO$ both saves and clears the 80th column if a screen format is overlaid up to column 79 on an 80 column screen.

6.6.3 Exception conditions

++++++++++++++++++++++++++

WINDO$ returns exception condition 1 if it is run on a version of System Manager which does not support GTSCR$, either because it is too early a version or because there is no screen image kept.

Exception condition 2 is returned if the save area is not large enough to hold the information from the screen.

In both of these cases the screen information has not been saved, nor the appropriate parts of the screen cleared.

6.6.4 Restoring the display using WRES$
++
Once your pop-up screen format is no longer required, you restore
the original display by calling WRES$ as follows:-

```
CALL WRES$ USING md save-area
////////////
```

where both md and save-area reference the same parameters used by
// /////////
the original call to WINDO$.

WRES$ again determines the location of the screen format from the
MD, and then restores and re-displays the screen information from
the save area using PTSCR$.

6.6.5 Exception conditions from WRES$
++++++++++++++++++++++++++++++++++++++
WRES$ returns exception condition 1 if it is called on a version
of System Manager which does not support PTSCR$. As an earlier
call to WINDO$ would also have failed this is not likely to arise
in practice, and probably indicates a programming error.

6.6.6 Programming Notes
+++++ +++++++++++++++++
The way in which you would normally use WINDO$ and WRES$ is as
follows:-

```
CALL WINDO$ USING md save-area * Save and clear
MAPOUT md TEXT * display window
MAPIN md ALL * input values
. . .
further statements to manipulate data in pop-up window
. . .
CALL WRES$ USING md save-area * restore window
```

Essentially WINDO$ is called to save the window area, which is
then used by the screen format to do such processing as is
required. When processing is complete WRES$ is called to restore
the window area.

It is important that WRES$ is not called without a preceding call
to WINDO$, as this would attempt to restore whatever data
happened to be in the save area to the screen with unpredictable
results.

If your program calls WINDO$ from several unrelated places the
calls may share the same save area. However, if WINDO$ is called
to create a second window while the first is still in use then
the second, nested, call requires its own separate save area.
You may create further windows in this way, so long as each has

its own distinct save area, and the areas are restored, via WRES$, in the reverse order to that in which they were saved.

If after you have finished with a window you decide to abandon the whole process, and move to another screen entirely, then you may simply clear the screen without calling WRES$ to replace the original screen contents.

7. Advanced Console Handling Facilities

This chapter describes the various system variables and system subroutines which can be used in conjunction with the screen formatter or with ordinary ACCEPT and DISPLAY statements to provide extra facilities for screen handling.

7.1 System Variables for Console Management

+++ ++

A number of system variables are provided so that you can write programs which can readily adapt themselves to the type of terminal in use. Some of the variables serve as flags which modify the functioning of the normal console I/O operations. Two of them are set following an accept operation to indicate the number of characters that the operator actually keyed and the terminator used.

7.1.1 The Terminal Code, $$TERM

++++++++++++++++++++++++++++++++

System variable $$TERM is a PIC X(6) field containing a unique code identifying the terminal. This is the value which the operator keys in response to the terminal prompt during sign-on. It identifies the terminal attribute program (TAP) used to supply the System Manager with detailed information about the device: the TAP program-id is of the form:-

$.code
////

where code is the contents of $$TERM.
////

Applications which use hardware-dependent features of certain terminals can examine $$TERM to determine which, if any, of the range of terminals they support is currently being employed as the console, and adapt themselves accordingly. $$TERM should not, of course, be used in this way by portable applications which require to run on a wide variety of different devices.

7.1.2 The Auto-input Flag, $$AUTO

++++++++++++++++++++++++++++++++++

Usually each input keyed by the operator must be completed by a terminator, normally <CR> or a control sequence such as <CTRL A>
-- ------
or <ESCAPE>. However, when the system variable $$AUTO, a PIC 9

COMP field, is set to 0, input will be automatically terminated when the operator keys the last character of the field, the length of which is implied by the ACCEPT, ACCEPT...LINE or CALL ACCE$... statement responsible for the operation. The auto-input flag only applies to a single accept: you must set it to zero

before each statement to be affected.

Auto-input is useful in creating simple dialogues where, for example, the operator selects from a menu of actions by keying just a single character. It also allows for the support of terminals which cannot transmit <CR> or other control sequences, for example telephone key pads.

You should note that even when $$AUTO is set to zero the operator is still able to end the input early by keying a terminator, providing the terminal supports the necessary key stroke.

7.1.3 The Escape Key Suppress Flag, $$ESC
++
Normally, when the operator keys ESCAPE key in response to a
prompt, control is immediately returned to the monitor or to a
menu. You may, however, wish to prevent this happening in
certain applications where an inadvertent return to the monitor
could cause difficulties, such as files becoming inconsistent, or
updates being lost. The System Manager provides the system
variable $$ESC to allow you to inhibit the normal ESCAPE key
handling.

$$ESC is a PIC 9 COMP field which controls the way in which the
ESCAPE key is handled. On entry to an application $$ESC is zero,
++++
and this setting causes <ESCAPE> to return control to the monitor
------
in the normal way.

You may set $$ESC to 1 to cause the normal processing to be
suppressed. In this case when <ESCAPE> is keyed it is treated
------
exactly as though the operator had keyed <CR>. Thus, when $$ESC
--
is 1 it is no longer possible for the operator to obtain a ready
prompt by keying <ESCAPE> in response to an application prompt.

Note that $$ESC is automatically reset to zero at end of job in
order to restore normal ESCAPE key handling.

7.1.4 The Type-ahead Buffer Flush Flag, $$FLSH
++
Characters keyed at the console are accumulated in a type-ahead
buffer until requested by an ACCEPT statement or equivalent. An
accept operation normally inputs characters from the buffer and
then, if a terminator is not present, takes them directly from
the terminal as the operator keys them. By setting the PIC 9
COMP system variable $$FLSH to 1 before an accept takes place you
can cause any information currently in the buffer to be "flushed"
so that the new input comes directly from the keyboard. The
flush flag only applies to a single accept operation: if you wish
to clear the buffer again you must reset the flag to 1.

The BELL statement, which you are recommended to use when an
error is detected, sets the buffer flush flag to 1. The assump-
tion is that any information in the buffer is useless, since you
would not be expecting the error. Note that the System Manager
issues a BELL statement, and hence clears the buffer, whenever it
detects an error, such as an invalid format number being keyed,
or an irrecoverable I/O error.

7.1.5 The Prompt Character, $$PROM
++++++++++++++++++++++++++++++++++

System variable $$PROM is a PIC X field which you may set to change the prompt character used in the next accept operation. If you set it to LOW-VALUES no prompt character at all is displayed. Normally $$PROM contains a colon character, and it will be restored to this value following the accept. Thus you must reset $$PROM every time you wish to produce a special prompt not identified by the usual colon character.

7.1.6 The Decimal Point Character, $$DECP
+++

System variable $$DECP is a PIC X field which holds the character currently being used as the decimal point. This is the character set using $CUS, and may not be modified by the program.

7.1.6 The Job Management Flag, $$JCL
++++++++++++++++++++++++++++++++++++

When an application program runs under job management, input for ACCEPT, ACCEPT...LINE, CALL ACCE$ or ACDEF$, CALL PASS$ and MAPIN statements is normally supplied from a previously prepared dialogue table held in memory. The operator is no longer prompted, and instead the necessary responses are supplied from the table. There is also an option of job management which allows you to suppress the output of information by DISPLAY, DISPLAY...LINE, CALL VIDEO$, and CALL DISP$, so that programs can be run in a "quiet" mode, with no console displays appearing.

System variable $$JCL is a PIC 9(2) COMP field which you can set to 1 to temporarily bypass job management, so that the next accept operation takes input directly from the operator, ignoring the stored dialogue. Any displays that take place while $$JCL is 1 are always output to the screen, irrespective of whether the general dialogue has been suppressed. $$JCL is reset to a value other than 1 by any accept operation, so that it only modifies one group of displays terminated by an accept at a time.

You can set $$JCL to 0 to cause the program to be terminated with an error if it is running under job management. Any displays that take place after setting $$JCL to 0 are always output to the screen, and the program will be terminated on the next accept operation. In practice, you will normally use the BELL statement which has the same effect, since it causes $$JCL to be set to zero.

This description of $$JCL is included in this chapter on advanced console management for completeness only, because the system variable affects the operation of a number of console I/O statements. You should refer to the Global Cobol Job Management Manual for a fuller explanation of the coding of application programs to run under job management.

7.1.7 The Input Field Length, $$LENG
++++++++++++++++++++++++++++++++++++

Following an ACCEPT statement, or equivalent, the PIC 9(2) COMP system variable $$LENG is set to contain the length in bytes of the input field actually keyed by the operator. This may be smaller, of course, than the input variable supplied by your program. the System Manager automatically packs a short character input with rightmost blanks, so you can use $$LENG to determine the significant part of the input without the need to write special code to detect and count any trailing blanks.

The following statements set $$LENG as described above:-

ACCEPT CALL ACCE$, ACDEF$...
ACCEPT...LINE CALL PASS$...

7.1.8 The End of Field Code, $$EOF

++++++++++++++++++++++++++++++++++++

System variable $$EOF is a PIC X field which indicates which terminator the operator keyed in order to complete a field input by an ACCEPT, ACCEPT...LINE, CALL ACCE$ or ACDEF$... or CALL PASS$... statement. The byte value of each terminator is in the range #01 to #1F, and in consequence #40 is added to it in calculating $$EOF, so that the relationship shown in table 4.1.8A is established:-

TERMINATOR	BYTE VALUE	$$EOF
<CTRL A>	#01	#41 or "A"
<CTRL B>	#02	#42 or "B"
<CTRL C>	#03	#43 or "C"
<LINE FEED>	#0A	#4A or "J"
<CR>	#0D	#4D or "M"
<ESCAPE>	#1B	#5B or "["

Table 7.1.8A - Special Terminators

BYTE	USUAL KEY NAME	SYSTEM MANAGER ACTION
#07	<CTRL G>	Sets an internal flag which can be checked by the TEST$ routine to cause a planned program interrupt
#08	<BACKSPACE>	Erases the last character keyed
#09	<TAB>	Moves the cursor to the next tab position
#11	<CTRL Q>	Activates transmission on a terminal controlled by X-on, X-off protocol
#13	<CTRL S>	Suspends transmission on a terminal controlled by X-on, X-off protocol
#15	<CTRL U>	Clears the type-ahead buffer
#16	<CTRL V>	Changes the software caps lock setting

#17	<CTRL W>	Schedules a break interrupt to stop a looping program
#18	<CTRL X>	This is often in use as the System Request key on a serial screen

----

Table 7.1.8B - Special Key Strokes Intercepted by the System Manager
+++

Note that if the auto-input flag, $$AUTO, is 0 when the accept
operation takes place and if the input is then completed by the
keying of the last character of the field, rather than a termin-
ator, $$EOF will be set to "M", just as if <CR> had been keyed.
--

The example below shows part of a program which tests $$EOF to
see if the operator has replied normally to its quantity prompt:-

```
KEY QUANTITY:number
******
```

by keying a number, followed conventionally by <CR> or whether he
--
or she has used the special terminator <CTRL C>:-

```
KEY QUANTITY:number<CTRL C>
****** ------
```

to request an extra report of how many items remain in stock:-

```
DISPLAY "KEY QUANTITY"
ACCEPT QTY
SUBTRACT QTY FROM STOCK
IF $$EOF = "C"
DISPLAY STOCK
DISPLAY " ITEMS REMAINING" SAMELINE
END
* COMMON ADDITIONAL PROCESSING BEGINS HERE
```

With some exceptions, any key stroke transmitting a byte value in
the range #01 to #1F will be treated as a terminator. However,
if you require to develop portable applications to run on as many
different terminal devices as possible, you should limit the
special terminators you use to those listed in Table 7.1.8A, and
treat any others as though <CR> had been keyed. This is the
--
convention adopted throughout Global Cobol.

If you must use terminators other than those listed in Table
7.1.8A, you should at any rate avoid the byte values shown in
Table 7.1.8B, since these are invariably intercepted by the
System Manager and processed specially, and are never returned in
$$EOF.

There is a problem with <ESCAPE> transmissions from certain term-
inals. Frequently special function keys are provided which send
groups of bytes starting with <ESCAPE>. This means that if one
of these is keyed in a field being input under the control of an
ACCEPT, ACCEPT...LINE, or CALL ACCE$ or ACDEF$, the input will be

terminated by the <ESCAPE> character, and the subsequent byte will be stored in the type-ahead buffer, and presented as the first character of the next input. Operators should thus normally avoid keying such characters, except when a program using the CHAR$ routine, which is able to process them properly, is in control.

7.1.9 The Version 5.0 ACCEPT Simulation Flag, $$AC-5

++

The system variable $$AC-5 is a PIC 9 COMP field that allows you to simulate the V5.0 ACCEPT...LINE...[COL...] statement. By setting $$AC-5 to 1, the ACCEPT...LINE...[COL...] statement will cause the input area immediately to the right of the colon to be erased by writing blanks to it. Once set, this condition will remain in force until the end of job when the field will be set back to 0.

7.1.10 The Tap Selection Byte, $$BYTE
++++++++++++++++++++++++++++++++++++++
The system variable $$BYTE is a PIC X field used in creating
TAPs. It is mentioned here only for completeness.

7.1.11 The Field Editing Flag, $$FED
+++++++++++++++++++++++++++++++++++++
The system variable $$FED is a PIC 9 COMP field used to indicate
whether field editing is enabled. It is set to 1 if field
editing is enabled and to 0 if not. It is always set to 0 in
pre-V6.0 systems.

7.1.12 The Field Editing Start Character Flag, $$FESC
+++
The Field Editing Start Character Flag is a PIC 9(2) COMP field
used to control the enabling of field editing in ACCEPT
statements. The following values are permitted:-

0 Field editing is not entered automatically, but may be
enabled during the ACCEPT as described in the Global
Operating manual (section 3.1).

-1 Field editing is not permitted. (Cursor editing keys
are returned as termination characters to ACCEPT.)

n (Where n is greater than 0) This causes field editing
/ /
to be enabled automatically, and the cursor to be
positioned under the nth character of the ACCEPT
/
field. Setting n to 1 means that the ACCEPT starts in
/
field editing mode.

Special Value
+++++++++++++

-2 This causes the cursor to be placed at the first
trailing space in the field.

On completion of the ACCEPT the flag is reset to 0.

7.1.13 The Accept Time Out, $$ATIM
+++++++++++++++++++++++++++++++++++
This is a PIC 9 (2) COMP field used to cause the subsequent
ACCEPT statement to "Time Out" if no operator input is keyed
within a specific time. It can be set to any period in seconds
between 1 and 60 (if set to 0 then "Time Out" will not occur).
The time-out is cancelled as soon as the operator keys any
character, or if there were any characters in the type-ahead
buffer when the ACCEPT was invoked. A NULL clause must be

coded. A null response is indicated by $$COND = 1. Time-out is indicated by $$COND = 2. See section 7.8.2 for use with CHAR$.

7.2 Special Video Operations, VIDEO$
++++++++++++++++++++++++++++++++++++

The VIDEO$ system routine can be used to highlight or underline individual fields, to display a field in reverse video, to display a field brighter or dimmer than normal, to make a field flash, or to turn the cursor on and off. These operations are only supported on certain display terminals, so the routine signals an exception if it is called for a function which is not provided.

7.2.1 Invocation
++++++++++++++++

To perform a special video operation, you use a CALL statement of the form:-

CALL VIDEO$ USING opcd
////

where opcd, the operation code, is the name of a PIC 9(4) COMP
////
variable, or integer literal, which selects the function to be performed. Possible operation codes are:-

1 Turn highlighting on, so that the next field or fields
+
displayed on this line will be highlighted. The cursor advances one character position. Colour screens whose TAP does not contain a highlighting sequence will use colour combination 7 instead.

101 As 1, but the position of the cursor remains unchanged.
+++

2 Turn highlighting off, so that subsequent fields are
+
displayed normally. The cursor advances one character position.

102 As 2, but the position of the cursor remains unchanged.
+++

3 Turn the cursor display on. The position of the cursor
+
remains unchanged;

4 Turn the cursor display off. The position of the cursor
+
remains unchanged.

5 Turn underlining on, so that subsequent fields are
+

underlined. The cursor advances one character position.

105 As 5, but the position of the cursor remains unchanged.
+++

6 Turn underlining off, so that subsequent fields are
+
displayed normally. The cursor advances one character
position.

106 As 6, but the position of the cursor remains unchanged.
+++

7 Turn reverse video on, so that subsequent fields are
+
displayed in reverse video. The cursor advances one
character position.

107 As 7, but the position of the cursor remains unchanged.
+++

8 Turn reverse video off, so that subsequent fields are
+
displayed normally. The cursor advances one character
position.

108 As 8, but the position of the cursor remains unchanged.
+++

9 Turn brightness on, so that subsequent fields are displayed
+
brighter than normal. The cursor advances one character
position.

109 As 9, but the position of the cursor remains unchanged.
+++

10 Turn brightness off, so that subsequent fields are displayed
++
at normal brightness. The cursor advances one character
position.

110 As 10, but the position of the cursor remains unchanged.
+++

11 Turn flashing on, so that subsequent fields flash. The
++
cursor advances one character position.

111 As 11, but the position of the cursor remains unchanged.
+++

12 Turn flashing off, so that subsequent fields are displayed
++
normally. The cursor advances one character position.

112 As 12, but the position of the cursor remains unchanged.
+++

13 Turn dimness on, so that subsequent fields are displayed
++
dimmer than normal. The cursor advances one character
position.

113 As 13, but the position of the cursor remains unchanged.
+++

14 Turn dimness off, so that subsequent fields are displayed
++
normally. The cursor advances one character position.

114 As 14, but the position of the cursor remains unchanged.
+++

7.2.2 Exceptions
++++++++++++++++
Exception condition 1 is signalled if the video operation you

have requested is not supported. This depends on the sequences defined for the console's terminal attribute program, which in turn depend on the capability of the device in question. For example, on some terminals special characters controlling high-lighting, reverse video, underlining, brightness, flashing and dimness always occupy a space on the screen, and so for these devices functions 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 14 may be supported but 101, 102, 105, 106, 107, 108 109, 110, 111. 112, 112, 113 and 114 cannot be.

7.2.3 Programming Notes

+++++++++++++++++++++++

In terminal attribute programs supplied with the System Manager the highlighting function is supported whenever possible, using brightening, or failing that, flashing characters or reverse video. The cursor display control, underlining, reverse video and associated operations are available if and only if the hard-ware supports them. You can use the $T command's TEST option to determine which operations are available on a particular terminal.

To emphasise information, whether by highlighting, reverse video, underlining etc, you select the appropriate pair of on/off operations defined above. You turn the hardware function on, display the information in the normal way, then turn the function off. For example, to highlight a group of fields separated from others by spaces, use operations 1 and 2:-

* CALL VIDEO$ USING 1 to output a single space preceding the group;

* Display the group of fields;

* CALL VIDEO$ USING 2 to suppress the highlighting and output a single space to the right of the group.

Each highlighted area must begin and end on the same line of the screen, so if a number of lines containing emphasised information are displayed you will have to use a similar sequence of operations on each line. If a single line contains two or more emphasised areas you will need to create each of them in the same way.

If you are writing portable applications to run on all types of terminals you should ensure that any emphasis you use serves merely as an enhancement and can be dispensed with when the hardware does not support it. For instance, in the highlighting example you can code the following calling sequence to ensure that the cursor is moved to the same screen position, to the right of a newly displayed space, whether or not highlighting is provided:-

```
CALL VIDEO$ USING 1
ON EXCEPTION
DISPLAY " " SAMELINE
END
```

The code for operation 101 which does not output a space simply ignores the exception:-

```
CALL VIDEO$ USING 101
ON EXCEPTION
END
```

Similar sequences should of course be coded for the operations which turn the hardware function off.

If you are using formatted screen working you must position the cursor at the start of the area to be highlighted (or underlined, or reversed) before calling VIDEO$. The field should be displayed using DISPLAY...SAMELINE or CALL DISP$, and not DISPLAY...LINE or MAPOUT. For example, to display XYZ in reverse video on line

6 column 20 you might write:-

```
CALL POSI$ USING 6 19
CALL VIDEO$ USING 7
ON EXCEPTION
DISPLAY " " SAMELINE
END
DISPLAY "XYZ" SAMELINE
CALL VIDEO$ USING 8
ON EXCEPTION
DISPLAY " " SAMELINE

END
```

Similarly, if you want to display highlighted characters on the baseline, you must first position the cursor on the baseline. This can be done by a DISPLAY SPACE statement, which positions the cursor at the start of the baseline.

On terminals which support more than one special function you should avoid turning two functions on at the same time, since the effect will differ from terminal to terminal.

7.3 The Screen Clearing Routine, CLEAR$, CEOL$ and CEOS$
+++ +++

The CLEAR$ system routine can be used to clear the screen of a display terminal and position the cursor in the top left-hand corner. This is the same function as that performed by the Global Cobol CLEAR statement, except that CLEAR always
++++++
establishes the formatted mode of terminal working, whereas CLEAR$ leaves the mode unchanged. Thus you can use the routine in teletype mode to erase the screen and continue with a scrolled dialogue: messages produced by subsequent DISPLAY statements will then begin to fill the blank screen, starting from the top.

The CEOL$ routine allows you to blank the remainder of the line on which the cursor is currently located (starting with the character currently under the cursor).

The CEOS$ routine is similar to the CEOL$ routine, except that it clears the remainder of the screen, starting with the character under the cursor.

7.3.1 Invocation
++++++++++++++++
The Clear Screen routine, positioning the cursor at the top left-hand corner, by simply coding:-

CALL CLEAR$

The Line Clearing routine is invoked by a call of the form:-

CALL CEOL$

The Clear Screen from Cursor routine is invoked by a call of the form:-

CALL CEOS$

After the call the position of the cursor is undefined, and it must be explicitly re-positioned before any further displays take place.

7.3.2 Exception
+++++++++++++++
For CLEAR$, exception condition 1 is signalled if the console is not type 1 or greater, a display with cursor control. In this case the screen will not be modified in any way by CLEAR$ and the position of the cursor will remain unchanged.

For CEOL$ and CEOS$, exception condition 1 is signalled if attempted on a pre-V6.0 system.

7.3.3 Programming Notes

++++++++++++++++++++++

A CLEAR statement followed by a SCROLL statement erases the screen (providing it supports cursor control) and re-establishes teletype mode. However, the cursor is then positioned at the bottom left-hand corner, at the start of the base line.

++++++

On most screens the CEOL$ and CEOS$ operations are supported by
the screen and hence are faster than displaying spaces. If they
are not supported under V6.0 then spaces are automatically
displayed instead.

7.3.4 Memory Paged Version

++++++++++++++++++++++++++

A memory paged version of these routines is available. The
memory paged version contains most of the code within the System
Manager so that the subroutine size is smaller. Please see the
System Subroutines Manual section on memory pages.

7.4 The Cursor Positioning Routine, POSI$

+++ ++++++++++++++++++++++++++++++++++++++

Providing the console is type 1 or greater, a display with cursor control, the POSI$ system routine can be used to position the cursor at any X,Y co-ordinate on the screen, including the base line. Before using the routine for the first time in a program you should either have checked that $$TYPE is positive, or have used the CLEAR statement to check that the terminal is of the appropriate type and clear it for formatted working. If you attempt to use POSI$ on a basic display without cursor control your job will be terminated with a stop code.

7.4.1 Invocation

++++++++++++++++

The cursor is positioned using a call of the form:-

CALL POSI$ USING Y X
/ /

where Y and X are integer literals or PIC 9(4) COMP variables
/ /
specifying the required line number and column number, respect-ively. As in formatted display working, line and column co-ordinates are counted starting at 1. That is, the top left-hand character position of the screen has X=Y=1.
/ /

An ACCEPT (or call on ACCE$) issued following cursor positioning will cause the colon prompt character to be output at the X,Y
/ /
co-ordinate you supplied to POSI$. A DISPLAY SAMELINE (or call on DISP$) will cause the first character of the output string to appear at that position.

7.4.2 Programming Notes

+++++++++++++++++++++++

The POSI$ routine can be used in conjunction with the ACCE$ and DISP$ routines described in the next two sections to allow the flexibility of variable character string working with formatted displays. An obvious application in which POSI$ and DISP$ might feature is in the production of histograms and bar charts at the console.

Normally, when the attributes of a field you require to accept or display can be determined when the program is coded, you do not require POSI$. You use the ACCEPT...LINE or DISPLAY...LINE statement described in the Global Cobol Language Manual to input or output the field in question to the application area of the screen.

POSI$ is slightly more powerful than ACCEPT...LINE and

DISPLAY...LINE inasmuch as it can position the cursor on the base line. However, this feature must be handled with great care, if it is used at all, since messages from the System Manager may appear on this line during formatted display working.

7.5 The Variable Length Accept Routine, ACCE$
+++ +++

The ACCE$ system routine outputs the prompt character (a colon) at the current character position of the current line and then inputs a character string whose length is given by a variable passed to the routine as a parameter displaying the current value of the string as a field editable default. If <CR> is keyed
--
exception condition 1 is signalled. If the input is shorter than the maximum length specified, the string is padded to that length with rightmost spaces.

7.5.1 Invocation
++++++++++++++++

A character string is accepted using a CALL statement of the form:-

CALL ACCE$ USING area length
//// //////

The first parameter, area, is the name of the area into which the
////
string is to be input, and which must contain the current value (or spaces if this has not yet been defined).

The second parameter, length, is the name of a PIC 9(4) COMP
//////
variable containing the maximum length of the string to be input. This must be in the range 1 to 80 inclusive, otherwise the program will be terminated in error.

7.5.2 Exceptions
++++++++++++++++

If <CR> is keyed then the input area will remain unchanged and
--
exception condition 1 will be signalled. The CALL statement should, therefore, always be immediately followed by an ON EXCEPTION statement.

7.5.3 Programming Notes
+++++++++++++++++++++++

You can use ACCE$ to accept a variable length numeric field, but you will have to check that the field is of the correct format, and if necessary convert it yourself, since ACCE$ only process character strings. It is best to use the intermediate code instructions defined in Chapter 6 of the Language Manual to validate and convert a field whose attributes are only known at run-time.

You may use ACCE$ for formatted display working but in this case you must ensure that the cursor is positioned correctly before

the routine is called, since any previous accept operation will
have left it at an indeterminate location. The cursor is pos-
itioned at the top left-hand corner of the screen by the CLEAR
statement or CLEAR$ system routine, or at a specific X, Y co-ord-
inate by the POSI$ routine.

When you call ACCE$ from within your program, the linker includes
parts of the ACCEPT...LINE processing, as indicated in Appendix
B, in case you use ACCE$ for formatted display working. If you
are sure that you are not using ACCE$ for formatted display
++++
working, you may avoid including this extra code by defining a
GLOBAL label Q$ACCE in the procedure division of your program.
/////
Not that you should exercise great caution with this, as if you
do attempt to call ACCE$ for formatted display working, control
will be passed to label Q$ACCE with unpredictable results.
/////

7.6 The Variable Length Display Routine, DISP$

+++ ++

The DISP$ system routine outputs a character string of a specified length at the current character position of the current line. The string to be displayed and its length are passed to the routine as parameters.

7.6.1 Invocation

++++++++++++++++

A character string is displayed using a CALL statement of the form:-

CALL DISP$ USING area length
///////////

The first parameter, area, is the name of the area containing the
////
string to be displayed.

The second parameter, length, is the name of a PIC 9(4) COMP
//////
variable containing the length of the string to be displayed, which must be in the range 1 to 80 inclusive, otherwise the program will be terminated in error.

7.6.2 Programming Notes

++++++++++++++++++++++

If you need to output variable length computational items you will need to convert them to numeric characters before using DISP$. The conversion of data items whose attributes are only known at run-time is best accomplished using the intermediate code instructions described in Chapter 6 of the Language Manual.

You may use DISP$ for formatted display working but in this case you must ensure that the cursor is positioned correctly before the routine is called, since any previous accept operation will have left it at an indeterminate location. The cursor is positioned at the top left-hand corner of the screen by the CLEAR statement or CLEAR$ system routine, or at a specific X, Y co-ordinate by the POSI$ routine.

7.7 The Password Routine, PASS$
+++ ++++++++++++++++++++++++++

The PASS$ system routine outputs the prompt character (a colon) at the current character position of the current line and then inputs a character string whose maximum length is given by a variable passed to the routine as a parameter. On full duplex terminals (the great majority) the echoing of characters is suppressed, making PASS$ ideal for accepting a secret password. On half duplex terminals with local echoing the hardware displays the characters automatically as they are keyed but PASS$ erases the password once it is complete.

If the input supplied by the operator is shorter than the maximum length specified, the string is padded to that length with rightmost spaces. If the operator keys <CR> the string will consist
--
entirely of spaces. If the operator keys in the full length of the string then the accept will be automatically terminated, that is <CR> will not have to be keyed.

7.7.1 Invocation
++++++++++++++++

A password is accepted using a CALL statement of the form:-

CALL PASS$ USING area length
///////////

The first parameter, area, is the name of the area into which the
////
password string is to be input.

The second parameter, length, is the name of a PIC 9(4) COMP
//////
variable, or integer literal, containing the maximum length of the string to be input. This must be in the range 1 to 80 inclusive, otherwise the program will be terminated in error.

7.7.2 Exceptions
+++++++++++++++++

Exception condition 1 will be signalled if the ACCEPT used in PASS$ has timed out when $$ATIM has set prior to the PASS$ call.

7.7.3 Programming Notes
+++++++++++++++++++++++

You may use PASS$ during formatted display working but in this case you must ensure that the cursor is positioned correctly before the routine is called, since any previous accept operation will have left it at an indeterminate location. The cursor is positioned at the top left-hand corner of the screen by the CLEAR statement or CLEAR$ system routine, or at a specific X, Y co-ordinate by the POSI$ routine.

7.8 The Character I/O Routine, CHAR$, CHARX$, ECHO$ and EOFCH$

The character I/O routine provides four entry points, named CHAR$, CHARX$, ECHO$ and EOFCH$. It can only be used on terminals with extended control functions ($$TYPE >1). The CHAR$ function is used to input single graphic characters from a terminal, without echoing them on the screen. In addition, it allows for the processing of special keys, such as CURSOR LEFT or ERASE, in a way which does not depend on the actual values transmitted by the terminal. The CHARX$ function can be used to optimise the handling of cursor movement keys as well as providing the functions of CHAR$. The ECHO$ function causes a single character to be output. The EOFCH$ function allows a program to analyse a special key used to terminate an ordinary accept operation. The prompt character, colon, is not automatically displayed when CHAR$ is used, nor is any operator input echoed. Any information that you wish to appear must be explicitly written to the screen by ECHO$.

By using CHAR$, CHARX$ and ECHO$ extremely sophisticated display handling programs, such as full screen editors, can be constructed in Global Cobol to run on any System Manager configuration with a suitable terminal.

7.8.1 Graphics and Control Functions

CHAR$ recognises two types of input, graphics characters and control functions. The graphics are the ASCII characters whose hexadecimal values range from #20 to #7D (or #7E, providing this is not used as a lead-in character). These values correspond to the printable subset of ASCII which, apart from a few slight national differences, usually has an identical representation on any terminal. For example, ASCII A is always represented by a byte containing the hexadecimal value #41.

The meaning of the other characters, which are used for control purposes, varies from terminal to terminal because the ASCII standard was introduced before display devices became commonplace. For example, to request the CURSOR UP function from a Lear Siegler ADM 3A terminal you hit <CTRL K> which transmits a
------
single byte with the hexadecimal value #0B. However, a CURSOR UP request on a Hazeltine 1510 results in two bytes being sent with the hexadecimal values #7E and #0C. The first byte of this two-character sequence is termed a lead-in character, and on a number of devices one or more such characters are used to introduce control functions.

The CHAR$ routine completes normally when a graphic character has been input, but signals an exception condition in response to any control function request. The exception number returned in

$$COND indicates the particular type of request that has been made. There are no less than 18 different types, so normally the ON EXCEPTION logic following the CALL will begin with a GO TO DEPENDING ON $$COND to route control quickly to the appropriate service routine.

The V6.2 and later versions of CHAR$ provides some ability to analyse for further function keys, as explained later.

7.8.2 Inputting a Character

++++++++++++++++++++++++++

You input a graphic character, or route control to the appropriate control function processing routine, by invoking CHAR$ with a call of the form:-

```
CALL CHAR$ USING inbyte
//////
ON EXCEPTION
GO TO DEPENDING ON $$COND
TO ... * PROGRAM FUNCTION 1
TO ... * PROGRAM FUNCTION 2
TO ... * PROGRAM FUNCTION 3
TO ... * PROGRAM FUNCTION 4
TO ... * PROGRAM FUNCTION 5
TO ... * RETURN
TO ... * ESCAPE
TO ... * CLEAR
TO ... * CURSOR HOME
TO ... * PAGE
TO ... * CURSOR LEFT
TO ... * CURSOR RIGHT
TO ... * CURSOR UP
TO ... * CURSOR DOWN
TO ... * TAB RIGHT
TO ... * TAB LEFT
TO ... * RUBOUT
TO ... * UNDEFINED
TO ... * TIMEOUT
END
```

All 19 TO statements should be present, although if a particular control sequence is not supported by the application, all that is necessary is to ignore it by transferring control to the initial CALL statement to obtain the next input.

The quantity inbyte is the name of a PIC X field in which a
//////
graphic character is returned when normal completion is signalled. The field is also updated when an exception takes place to contain a binary representation of the character, or characters, that make up the control function. The values returned for the particular functions are device-dependent, and can be obtained by running the $T command, described in the Operating Manual, on the terminal in question. If a byte value is transmitted which corresponds neither to a graphic value, nor to one of the 17 values listed by $T, exception condition 18 is signalled.

Note that if you are writing a program which is to be device-independent so as to operate on a number of different terminals, you should not test or otherwise use the value returned in inbyte when an exception occurs, since such values differ from device to device. In particular, you should not attempt to display the inbyte value as the result will be unpredictable and may not even be related in any obvious way to the original key stroke.

Table 7.8.4 - Control functions and their typical implementation

+++

7.8.3 Using the CHARX$ function
+++++++++++++++++++++++++++++++
You can optimise the handling of cursor movement keys by invoking CHARX$ with a call of the form:-

CALL CHARX$ USING ch count
////////

where ch is a PIC X variable as for CHAR$, and count is a PIC 9
// /////
(4) COMP count of the number of times the indicated function occurred consecutively. The count field will only be returned as
/////
other than 1 for the cursor movement functions, as it has been provided to speed up movement of the cursor by allowing the calling program to 'accept' many cursor movement key strokes at once.

7.8.4 Outputting a Character
++++++++++++++++++++++++++++
You can output a graphic character by invoking ECHO$ with a call of the form:-

CALL ECHO$ USING outbyte
///////

where outbyte is the name of a PIC X field, or a single character
///////
literal, containing the character to be transmitted.

7.8.5 Programming Notes
+++++++++++++++++++++++
If CHAR$ or CHARX$ is called by a program executing under job management, the stored dialogue is not used, but the input is obtained directly from the terminal. To ensure that any explanatory text displayed prior to calling the routine continues to appear even when job management has suppressed the output of dialogue, you must set system variable $$JCL to 1 immediately before the DISPLAY, DISPLAY...LINE, or CALL DISP$ statements involved, as explained in 7.1.5. You should note that characters output by ECHO$ are always displayed, irrespective of the status of job management.

If you use CHAR$ or CHARX$ on a terminal with $$TYPE < 2 your program will be terminated with a stop code.

You have complete program control of the way in which the 17 different control functions and the undefined condition are handled. However, it will be most helpful for the operator working with a variety of applications if similar keys are used for similar purposes. Table 7.8.4 lists each exception condition

generated by CHAR$, the name of the control function associated with it, and the usual way in which the function is implemented.
It is recommended that an application using CHAR$ should produce a help screen listing the terminal key names and the functions for which they are employed. The key names themselves, which differ, of course, from one type of console to another, can be obtained using the KEYS$ system routine described in the next section.

7.8.6 Interpreting an accept terminator using EOFCH$

++

In some situations you will wish to input data using conventional accept operations (using the ACCEPT verb, Screen Formatting, CALL ACCE$ or one of the other accept using routines), but desire to be able to determine if the operator has terminated the reply using one of the function keys, so that you may take some special action.

You would typically use a call to EOFCH$ in the following way:-

```
ACCEPT REPLY NULL * or some other accept operation
CALL EOFCH$ * interpret function response
ON EXCEPTION
GO TO DEPENDING ON $$COND
TO ... * PROGRAM FUNCTION 1
. . .
. . .
TO ... * UNDEFINED
END
END
```

Here EOFCH$ is only called when the function response is keyed without any other data (as a 'null' response). If you always wished to detect a function response you would place the call to EOFCH$ after the NULL clause on the accept.

EOFCH$ returns the same exception conditions as CHAR$ and CHARX$ (see 7.8.2), and they should be handled in the same way. Exception 19 (time-out) will never be returned, so no entry for this needs to be coded.

If the operator terminates the reply with <CR> in the usual way, EOFCH$ will signal exception 6 (for the RETURN function). If the operator terminates the reply with <ESC> this will in general signal exception 7. However, if <ESC> is the lead in for the function keys on the terminal then their function values will not
+++
be returned by EOFCH$, as the <ESC> value is returned by preference. With V6.2 the handling of the most common terminals using <ESC> as the function lead in (the VT100 family of terminals) has been altered so that this should not present a problem.

If the terminal uses some other lead in for the function keys EOFCH$ will automatically input the remainder of the characters generated by the key, interpreting the functions in the usual way.

Function keys used by the System Manager field editing (F1, F2, cursor movement keys, CLEAR, HOME, TAB and RUBOUT) will not cause an accept to be terminated and hence cannot be detected by EOFCH$. If field editing is disabled for a particular accept (using $$FESC) then F1, F2, CLEAR and HOME may be returned.

Probably the most common way of using EOFCH$ will be in the validation routine of a screen format for a particular field, where you wish to use some standard function key to cause a special action (such as displaying a list of options for the field).

7.8.7 Detecting extra function keys

V6.2 and later CHAR$, CHARX$ and EOFCH$ attempt to recognise function key values returned in excess of the five defined functions returned by earlier versions. Where a key can be determined to be a function key, all these routines place the function key number in a data field which may be examined by making an EXTERNAL SECTION declaration as follows:-

```
EXTERNAL SECTION QT$FNC
77 QT-FNC PIC 9(2) COMP * function number
```

The field QT-FNC contains the number of the function key, or zero if the key was not a function key.

Up to 16 function keys can be detected in this way. For a portable application you should probably not use more than 9 function keys (common keyboards only contain 9 usable function keys with the 10th used as SYSREQ), or you might wish to make your determination dependent on the terminal code contained in $$TERM.

The detection of function keys is independent of the exception value which might be returned by CHAR$ et al, so for example functions 1 to 5 would return exceptions 1 to 5 and also set QT-FNC to a value in the range 1 to 5 when input via CHAR$, but if the F6 key were used as the CLEAR function it would return exception 8 and QT-FNC would contain 6. For consistency you should either only check the function number if CHAR$ returns exception condition 18 (which is the practice we would recommend, allowing extra function keys to extend the range of values detectable by CHAR$), or you should check the function number instead of using the exception condition returned by CHAR$.

Important note: values of QT-FNC greater than 16 may be used in
++++++++++++++
future versions of the System Manager, so your program should validate the range of the value in QT-FNC before using it to index a table or perform a GO TO DEPENDING construct.

7.8.8 Memory Paged Version

Memory paged versions of these routines are available. These versions have most of the subroutine code within System Manager making the subroutines smaller. Please refer to the System

Subroutines Manual on how to link memory paged subroutines.

7.9 The Key Names Routine, KEYS$

+++ ++++++++++++++++++++++++++++

The KEYS$ system routine supplies you with the short 30-character
console description, the names of the keys used for the extended
control functions, and, optionally, a full page of explanatory
text detailing any special features. The information is obtained
from the terminal attribute program, (TAP), which must be on-line
when the routine is called.

7.9.1 Invocation

++++++++++++++++

You can obtain the key names and other information using a CALL
statement of the form:-

CALL KEYS$ USING KE [area]
/////////

The parameter KE identifies a 200-byte area in which KEYS$
//
returns the console description and key names. The format is:-

```
01 KE
02 KEDESC PIC X(30) * CONSOLE
02 KENAME OCCURS 17 PIC X(10) * KEYNAMES
```

The seventeen 10-byte key names define the key strokes needed for
control functions 1 to 17. Each will be set to spaces if KEYS$
is used on a terminal which does not support extended control
functions (ie where $$TYPE < 2)

The area parameter is optional. When present it labels a
////
1500-byte area in which a screen of explanatory text will be
returned. The information consists of a PIC 9(4) COMP control
word, followed by an optional line to be displayed, followed by
another control word, another optional line, and so on, until a
special terminator control word is met. Control word values and
their meanings are as follows:-

```
| | |
-------------------------------------------------------
| | |
| CONTROL | MEANING |
+++++++ +++++++
| WORD | |
++++
| | |
-------------------------------------------------------
| | |
| >0 | The length of the subsequent text line to |
| | be displayed |
```

0	Display a blank text line
-1	Terminates the text area

Table 7.9.1 - Control Word Values and their Meanings
++

The text lines themselves are in a form suitable for immediate display, with no embedded tabs or other control characters. There will be at most 20 lines to be displayed, including blank ones.

7.9.2 Exceptions
++++++++++++++++
Exception condition 1 will be signalled if an irrecoverable I/O
error arose when KEYS$ attempted to load the terminal attribute
program. Exception condition 2 will be generated if the TAP was
not on-line when KEYS$ was called.

7.9.3 Programming Notes
+++++++++++++++++++++++
KEYS$ obtains the information to return to you from the terminal
attribute program which must be on-line at the time the routine
is called. The name of the terminal attribute program is $.code,
////
where code is the number in $$TERM, established during sign-on in
////
response to the terminal prompt or permanently fixed, depending
on what is specified in the $CUS Sign-on Customisation Terminal
Type option. The TAP must either be stored as a stand-alone
program file, or must be a member of library P.$TAP.

KEYS$ searches for the TAP on the device from which the last
program was loaded. Thus, when you build an application which
uses KEYS$, you should copy P.$TAP and any stand-alone TAPs from
SYSRES, where they are installed, to the volume containing your
application library.

The system variable $$SRKT is a PIC X(10) field that holds the
key top description of the System Request key. This is always
available to the program.

7.10 Check for Console Input Routines, CHECK$ and CHKPR$
++

The CHECK$ system routine allows you to determine whether or not the operator has keyed any characters without the need to stop your program (in an ACCEPT statement or ACCE$ or CHAR$ call) until he or she does so. The routine can therefore be used to develop applications which are sensitive to console interrupts caused by the operator keying unsolicited characters. The CHKPR$ routine performs a similar function, except that it checks for characters awaiting input in another partition of the screen.
Both routines work by examining the type-ahead buffer. They complete normally if there is information present in the buffer, indicating that the operator has keyed one or more characters. Alternatively, if the buffer is empty or the terminal does not support the type-ahead feature, an exception is signalled.

7.10.1 Invocation
+++++++++++++++++

You check for console input by coding:-

```
CALL CHECK$ [USING count]
                   /////
```

to check in the current partition, or by coding:-

```
CALL CHKPR$ USING partition [count]
                  ///////// /////
```

to check in another partition. The routines complete normally if the type-ahead buffer of the indicated partition contains information. However, exception condition 1 is signalled if the terminal does not possess a type-ahead buffer, and exception condition 2 if the buffer is present, but empty. The parameter partition is a PIC 9(4) COMP variable or integer literal
/////////
containing the partition number to be examined.

If the optional PIC 9(4) COMP parameter count is supplied then if
/////
the routine completes normally then the position of the first <CR> or <ESC> character within the buffer will be returned (or zero if neither of these characters are present). This allows a program to determine whether the reply to the next ACCEPT is already available, enabling it to avoid displaying menus or prompts unnecessarily, thus improving performance.

7.10.2 Programming Notes
+++++++++++++++++++++++

You may want to ignore any characters already typed ahead by setting the flush flag, $$FLSH, before the first call of CHECK$.

In a multi-user system, each partition has its own type-ahead buffer. Characters keyed by the operator are placed into the type-ahead buffer of the partition which currently has control of the console.

Normally programs using CHECK$ should not be run under job management, since they will behave unpredictably depending on whether on not the operator types ahead at any time. CHECK$ is purely concerned with the status of the type-ahead buffer, and ignores the dialogue table established by job management.

If your application only needs to check for one type of
unsolicited operator input, meaning, for example, "stop printing
the current report" you should consider using the <CTRL G> key
----
stroke for this purpose and employing the TEST$ routine described
next to see whether it has been keyed. By avoiding the use of
CHECK$ in this case you allow the operator still to make use of
type-ahead in the normal way.

7.10.3 Memory Paged Version
++++++++++++++++++++++++++
Memory paged versions of these subroutines are available. In
these versions most of the subroutine code is held within System
Manager so making the subroutine smaller. Please refer to the
section on memory paged routines in the System Subroutines
Manual.

7.11 Test for <CTRL G> Keyed Routine, TEST$
+++
The TEST$ system routines enables you to test whether the
operator has keyed <CTRL G> (or <PR> G if the terminal supports a
------ -- -
prefix key in place of CTRL) since the previous TEST$ call or the
last accept operation. (An accept operation results from a MAPIN
or ACCEPT statement, or ACCE$ or PASS$ call.)

7.11.1 Invocation
+++++++++++++++++
You check for unsolicited keying of <CTRL G> by coding:-

```
CALL TEST$
```

The routine completes normally if <CTRL G> has not been keyed
------
since the previous TEST$ call or the last accept operation. If
the key stroke has been made exception condition 1 is signalled.

7.11.2 Programming Notes
++++++++++++++++++++++++
If <CTRL G> is keyed and an accept operation occurs before a

TEST$ call, the console's audible alarm will sound.

You should note that the keying of <CTRL G> will clear the
console executives type ahead buffer.

7.12 Colour Working with the System Manager
++

7.12.1 Colours and Combinations
++++++++++++++++++++++++++++++
On colour screens the System Manager supports 8 colours, numbered
1 to 8 as follows:-

1 = Black 5 = Red
2 = Blue 6 = Magenta
3 = Green 7 = Yellow
4 = Cyan 8 = White

You can select any combination of 'paper' (the background colour)
and 'ink' (the colour of the characters displayed) when you
display information on the screen.

Rather than having each software module select its own colour
combinations, the System Manager provides 8 standard colour
++++++++++++++++
combinations. These combinations can be altered, using
+++++++++++++
customisation, to suit the user's preferences. These colour
combinations have special uses, as follows:-

1 = scrolled working, baseline
2 = line and boxes
3 = text and headings
4 = variable information
5 = not used
6 = help colour
7 = highlighting colour
8 = field input colour

Colour 1 is used by default. The field input colour is used in
formatted mode while a field is being entered, so that the
position of the field on the screen and its length are clear to
the user. The highlighting colour is used to draw attention to
an anomalous value or error condition. Fields in maps set up
using screen formatting can be in any of the eight standard
colour combinations.

7.12.2 Monochrome Screens
+++++++++++++++++++++++++
Screens with 8 monochrome shades available are treated as if they
are in full colour, with the colour sequence selecting the 8
shades.

For other monochrome screens reverse video is used for field
input (colour 8) and highlighting for colour 7 when possible.
Such substitution can only be done when the selection sequences

do not occupy a space on the screen, and when they only affect subsequent displays. Substitution of reverse video, etc for colour in this way is referred to as pseudo-colour. When you use
+++++++++++++
pseudo-colour the ink colour is ignored; only the paper colour is used. If pseudo-colour is in use then colours 2 and 3 will be DIM, if this is available. It is possible to use other representations for the remaining standard colours by setting up a special TAP for screens where similar sequences are used for all the attributes.

7.12.3 Field Input

+++++++++++++++++
When the screen is in formatted mode (ie after a CLEAR statement) any accepts except those done using CHAR$ will display the input area in the input colour before starting the accept, and will re-display it in the current colour at the end of the accept. This makes the current input field easy to see, and also shows its length. When colour is used for input colon prompts are automatically suppressed.

If you do not want accepts to be done in this way you should replace any CLEAR statements by a:-

CALL CLR-N$

statement, which clears the screen but does not enable colour working.

When the input field is displayed in colour, it is shown as either spaces or the current value of the field according to the type of accept:-

Default spaces: ACCEPT X
ACCEPT X NULL
ACCEPT X LINE Y [COL Z]
CALL PASS$
MAPIN input field
///////////

Default current value: ACCEPT X LINE Y [COL Z] NULL
CALL ACCE$
MAPIN update field
////////////
(CALL ACDEF$ - now obsolete)
CALL BASEL$

Note in particular that this means that the default value must be
++++
in X before you issue an ACCEPT X LINE Y COL Z NULL statement. In most cases the field will already contain the default since this is a natural way to write the code, but you must check any accepts of this form to make sure they do have the correct defaults established. Note also that the code generated for this statement is changed by $LINK when the program is linked: this means that the linked program will not correspond precisely with the code shown on the compiler's binary list (the size of the program is not, however, altered).

This may be affected if the $$AC-5 flag has been set (see 7.1.9).

7.12.4 The COLOR$ System Routine
++++++++++++++++++++++++++++++++
The COLOR$ routine selects a colour combination to override the colours that would otherwise be used, but does not affect the input colour. The selected remains in force until the next call of COLOR$, or end-of-job is reached.

Note colours selected by COLOR$ override colours used by Screen Formatting until a parameterless call of COLOR$ is issued. This is so that you may mapout a selection of fields in the same colour (to draw attention to an illegal situation, for example) without needing to define a separate screen format.

To change colour you use a call of the form:-

```
CALL COLOR$ USING CC
//
```

where CC is either a PIC 9(4) COMP variable or an integer
//
literal. It must contain either a value in the range 1-8, or a block of the form:-

```
01 CC
02 INK PIC 9 COMP
02 PAPER PIC 9 COMP
```

The first form selects one of the standard colours, 1 to 8. The second form selects a specific colour combination, as given by the paper and ink colours, each in the range 1 to 8. The select-ed colour overrides colours specified in the map when used with screen formatting.

To return to using the standard colour, or the colours defined in the map if used with screen formatting, you simply call COLOR$ without any parameters:-

```
CALL COLOR$
```

7.12.5 Highlighting
++++++++++++++++++
On colour screen a VIDEO$ call to do highlighting will cause colour combination 7 to be selected, unless the TAP specified some other sequence to be used.

Note that this colour will not override the colours used by screen formatting. To produce highlighting with screen format-ting you should call COLOR$ and select colour 7; this will also work on pseudo-colour screens by enabling highlighting.

7.12.6 Change Standard Colours Routine, SCOLR$

++
The SCOLR$ routine allows a program to change the 8 standard
colours, as used by screen formatting, to different combina-
tions. It has no effect when the "colours" are represented by
pseudo colours only.

The colours are changed by a call of the form:-

CALL SCOLR$ USING table
/////

where table consists of 16 PIC 9 COMP entries, the first two of
/////
which are the ink and paper colours for colour combination 1
(standard screen colour), the second two for colour combination
2, etc. A call of SCOLR$ with no parameters resets the colours
to the customized colours: you should reset the colours on exit
from the program unless you want them to be used by subsequent
programs. The colours are reset automatically at end of job.

7.12.7 Memory Paged Versions
++++++++++++++++++++++++++++
Memory paged versions of these subroutines are available. These versions have most of the code within System Manager so the subroutine in smaller. Please refer to the section on memory pages routines in the Systems Subroutines Manual.

7.12.8 THE $$CLR System Variable
++++++++++++++++++++++++++++++++
A new system variable, $$CLR PIC 9 COMP, indicates the type of screen, as follows:-

-1 = no colour (or pre-V5.2)
0 = pseudo-colour (reverse/highlighting)
+ve= full colour

7.13 Box and Line Drawing with the System Manager
++

The BOX$ system routine enables you to draw a box or a line on the screen, using the terminal's own graphic characters (if available) or a set you can optionally define yourself.

The LINE$ routine enables you to draw a horizontal line of up to 80 characters using the line drawing characters if they are available, or otherwise using minus signs. It is intended for use in programs that do not require the full generality of the BOX$ routine, and is about 1K smaller than BOX$.

If a program contains the BOX$ routine and you wish to draw a line, then you should use this routine instead of LINE$ to avoid including two separate routines when one will do.

7.13.1 Invocation
+++++++++++++++++

The BOX$ routine is invoked by a call of the form:-

CALL BOX$ USING bc [table]
//////////

where bc is a control block of the form defined in 7.13.2 below
//
and table is an optional second parameter, a table in which you
/////
can define the characters to be used if your terminal has no graphic characters defined; see 7.13.3 below.

The LINE$ routine is invoked by a call of the form:-

CALL LINE$ USING length
//////

where length is a PIC 9(4) COMP variable or integer literal
//////
containing the length of the line to be displayed.

The line is displayed starting from the current cursor position.

7.13.2 The Box/Line Definition Control Block
++

The first parameter passed to BOX$ is a record of the format shown below, which defines the starting point on the screen, the width and depth of the box (if either is 0 a line will be drawn), and (in the case of a line) the characters to be used for each end or (in the case of a box) whether it is to be filled with spaces or not. You should define a box as space-filled if it is not in the same paper colour as the rest of the screen.

```
01 BC
02 BCLINE PIC 9(4) COMP * START LINE NO.
02 BCCOL PIC 9(4) COMP * START COLUMN NO.
02 BCWID PIC 9(4) COMP * WIDTH OF LINE/BOX
02 BCDEP PIC 9(4) COMP * DEPTH OF LINE/BOX
02 BCLTI PIC 9(2) COMP * INDEX FOR LEFT/TOP
02 BCRBI PIC 9(2) COMP * INDEX FOR RIGHT/BTM
02 BCSOLI PIC 9(2) COMP * IF 1 = SPACE FILLED
* IF 0 = OUTLINE
```

If either BCWID or BCDEP is set to zero then a line will be drawn. Zero width defines a vertical line. Zero depth defines a horizontal line.

BCLTI defines the index (taken from the table below) of the leftmost character of a horizontal line or the uppermost character of a vertical line. Similarly, BCRBI defines the index of the rightmost or bottom character. If a box is to be drawn then these two fields are ignored.

BCSOLI is ignored if a line is to be drawn as it defines whether the box is to be filled or not.

The System Manager table of graphic characters are defined as shown in table 7.13.2.

INDEX	CHARACTER DESCRIPTION	HEX	CHARACTER
1	Vertical bar	#80	
2	Horizontal bar	#81	
3	Bottom lefthand corner	#82	
4	Top lefthand corner	#83	
5	Top right-hand corner	#84	
6	Bottom right-hand corner	#85	
7	Vertical bar joined from right	#86	
8	Horizontal bar joined from below	#87	
9	Vertical bar joined from left	#88	
10	Horizontal bar joined from above	#89	
11	Four way junction	#8A	
-	Full height block	#8B	
-	Half height block	#8C	
-	RESERVED	#8D	
-	RESERVED	#8E	
-	Shaded block	#8F	

Table 7.13.2 - The System Manager table of graphic characters

7.13.3 The Optional Table of Characters

You may pass the routine a second parameter - a table which contains the 11 characters you want to use - if no System Manager table of graphic characters is defined for your screen. For example:-

```
77 TABLE PIC X(11)
VALUE "|-++++|-|-+"
```

If no table of graphic characters is defined for the screen by the System Manager or by your program exception condition 1 will be signalled.

7.13.4 Exception conditions
++++++++++++++++++++++++++

Exception condition 1 will be returned if an attempt is made to run BOX$ on a version of System Manager which does not have suitable box drawing characters.

7.13.5 Line drawing characters and the System Manager
++
The System Manager uses the characters in the range #80 to #8F as line and block drawing characters. These are as shown in table 7.13.2 Programs running under System Manager V6.0 or later may display these characters directly to the screen. The must not be
+++
used on a pre-V6.0 system.

Note that characters #80 to #8A are displayed as vertical bars, hyphens and plus signs as appropriate if the screen does not support line drawing characters. They are also handled appropriately by the printer.

Characters #8B to #8F will be displayed as spaces if the screen does not support them and so should be used with caution.

```
...............................................
. .
. .
. .
. .
. .
. .
. .
. .
. .
. PHOTO .
. .
. .
. .
. .
. .
. .
. .
. .
. .
...............................................
```

Figure 7.14A - The Example Menu, As Displayed

```
++++++++++++++++++++++++++++++++++++++++++++++++
```

```
01 ME * MENU DEFINITION
03 MEFUNC PIC 9(2) COMP * RETURNED FUNCTION
03 METEXT PIC X(?) * TEXT
VALUE "Distribution."
VALUE "Label Printer."
VALUE "Today's Orders."
VALUE "Specific Orders."
VALUE "Software Generation Masters."
VALUE "Configuration Volumes."
VALUE "Production Volumes."
VALUE "Preformatted Diskette Masters."
VALUE "Duplicator Program Diskettes."
VALUE "Software Stock Items."
VALUE "Preformatted Diskette Stock."
VALUE "Exit."
VALUE "Please select a Function.."
```

Figure 7.14B - The Menu Definition Area for the Example
+++

7.14 Standard Menu Handling Routine, MENU$
++
The MENU$ system routine provides you with a simple means of displaying a menu conforming to the System Manager conventions. You provide the routine with a table of menu lines; it displays the menu, accepts a reply, and returns you the number of the function selected.

When MENU$ is used on standard screens it clears the screen first, then displays the menu. On simpler terminals where only scroll mode is supported it leaves a single blank line, then outputs the menu.

Note that MENU$ provides only a subset of the complete facilities available with the V6.2 and later menu handling system.

7.14.1 Standard Menu Conventions
++++++++++++++++++++++++++++++++
Menus displayed by Global software are usually similar in format to the example shown in figure 7.14A opposite. They are centred horizontally in an otherwise clear screen in an area which is 33 characters wide and f + 10 lines deep, where f is the total of
/ /
numbered functions supported. The maximum value of f is
/
restricted to 13 by the MENU$ routine in order that any menu displayed will fit within the 23-line formatted area of the normal 24-line screen.

The menu lines, reading from top to bottom, are as follows:-

* top line of screen, underline, blank line;

* the title, centred;

* the title underlined;

* the subtitle, centred;

* the first separator line;

* the f numbered function lines, each of which consists of the
/
function name, zero or more double spaced leader dots, and then the function number. The dots are so arranged that the rightmost one appears in what would be the hundreds position of the number, which is right justified so that its units position occupies column 33 of the menu area;

* the default function line, similar to a numbered function line except that instead of a number it ends with <CR> to

indicate that you select it by hitting the <RETURN> key by
itself. Normally the default function is used to exit from
the program displaying the menu in order to return to a
higher level of control, such as your system menu;

* the second separator line, 33 spaces;

* the prompt line, consisting of the prompt text followed by a
colon in column 31, then 2 spaces.

7.14.2 Invocation
+++++++++++++++++
To invoke the MENU$ routine you call it passing the menu
definition area. For example:-

CALL MENU$ USING ME [title]
///////

The menu definition area consists of a PIC 9(2) COMP function
number field in which the routine returns the operator's
selection, followed by a PIC X(?) text field in which you list
the various text elements used in the menu in the order they are
to be displayed:-

* the title;

* the subtitle;

* the names of the f (0-16) numbered functions;
/

* the name of the default function;

* the text to be used for the selection prompt.

Each separate item must be terminated by a period (full stop) and
the selection prompt text which ends the list must be followed by
2 periods. See the example in figure 7.14B. On displaying the
menu, as in 7.14A, the routine will perform the necessary
centering, and add the underline, separator lines, leader dots,
function numbers, <CR> at the end of the default function, and a
: character at the end of the prompt text.

The optional parameter title defines a PIC X(50) title area to be
/////
displayed underlined on the top line of the screen, with the date
and time at the right hand end of the title line.

Once the menu is displayed the operator will be prompted
(repeatedly if necessary) until he or she replies with a valid
function number, corresponding to one of the numbered function
lines displayed, or keys <CR> to select the default function.
--
The function number, or 0 in the case of <CR> , will be returned
--
in the PIC 9(2) COMP field at the start of the menu definition
area. If <ESCAPE> is keyed, the result depends on the value of
------
the $$ESC system variable on entry to MENU$. If this contained
zero (its default value) then the calling program will lose
control and the ready prompt or system menu will be displayed.
If $$ESC is 1, then <ESCAPE> is treated as though the operator
------
had selected the default function.

7.14.3 Programming Notes
+++++++++++++++++++++++
Because the period is used as a separator, it obviously cannot be
part of one of the text strings appearing in the list, and there-

fore cannot feature in the title, subtitle, function names or prompt text.

The maximum length of each separate text item is 30 characters, or 28 in the case of the default function name. If more characters appear before the terminating period the extra ones will simply be ignored.

The minimum acceptable text list consists of just 4 items; the title, subtitle, default function name, and prompt text. The maximum list has 20 items, ie those of the minimum list together with 16 additional ones naming the numbered functions. If you supply a list with less than 4 or more than 20 items, or one that does not end with a double period, your program will be terminated with a stop code.

7.15 The Screen Width Routines, WIDE$ and NAR$

+++

The WIDE$ routine is used to place the screen into wide mode. If the screen is capable of physically changing width then it will do so, otherwise the 79/80 column screen will be a window onto the logical screen. The <SYSREQ>L and R keys can be used to select the window displayed, and the window will be changed automatically if data is accepted from a position outside the current window. If the width requested is greater than 132 then the two overlapped 132 character windows onto the wide screen will be used if possible, otherwise three overlapped 80 character windows will be used.

7.15.1 Setting Narrow Mode

+++++++++++++++++++++++++++

The NAR$ routine restores the screen width to narrow mode (79/80 characters) if the system was in wide mode.

The NAR$ routine is invoked by a call of the form:-

```
CALL NAR$
```

7.15.2 Setting Wide Mode

+++++++++++++++++++++++++

The WIDE$ routine is invoked by a call of the form:-

```
CALL WIDE$ [USING WB]
```

Where WB is a control block in the following format:-

```
01 WB
02 WBWID PIC 9(4) COMP * new screen width
02 WBWMOD PIC 9 COMP * enable wide mode flag
02 WBLINE OCCURS 4 PIC 9 COMP * fixed position lines
```

If WB is omitted the screen is switched into 131/132 wide mode if this is supported, otherwise the narrow screen is used as a window onto a 132 wide screen but with the bottom line always showing from its beginning (as, in Global software, this line is usually the baseline prompt (see BASEL$, section 7.23)).

WBWID is the width you require, up to a maximum of 192 characters.

WBWMOD is a flag to control whether windowing via a narrower screen is allowed:-

* If WBWMOD is set to 1 windowing is not acceptable and an exception will be signalled if the screen cannot be switched into wide mode.

* If WBWMOD is set to -1, the screen will be switched to wide mode if possible; otherwise windowing will be used.

* If WBWMOD is set to 0 then windowing is always used with the screen in narrow mode.

WBLINE allows up to 4 lines of a windowed screen to be defined as fixed lines. (For example, on a Global Writer screen the top line is a fixed status line showing the current editing mode and so on, and the bottom line is used for commands; therefore only lines 2 to 23 are used for displaying the document and so only these lines need to be windowed. In this case, WBLINE is set to 1, $$LINE (to allow for 25 screen lines), 0, and 0. The zeros indicate unused table entries.)

7.15.2 Exceptions
+++++++++++++++++
Exception 1 is returned if it is attempted on a pre-V6.0 system, or a V6.0 or later system that does not have a wide enough screen image.

7.15.3 Programming Notes
++++++++++++++++++++++++
$$WIDE is set to the maximum value of WBWID and the new physical screen width (which is either $$RWID or $$RWID - 52 if the screen is capable of wide mode but has not been enabled. Lines specified in the line table (WBLINE) are not shifted when the screen is shifted. You will normally want the baseline left in its normal position, but you may also want others (such as the top line) not to shift either. Zero entries in the table are ignored.

Note that the screen width specified in the configuration file or via $CUS must be at least as wide as the width requested, otherwise an exception will be signalled. Systems are standardly distributed with a width of 132.

7.16 The Read Screen Routines, RDSCR$, GTSCR$ and PTSCR$

The screen reading routines allows you to read back information currently displayed on the screen. Although they should not be needed in a simple application they can be very useful in more general purpose pieces of software.

7.16.1 The SI control block

The SI control block is used to control the processing of screen information. It has the following layout:

```
01    SI
 02   SILENG PIC 9(4) COMP          * Number of characters read
 02   SILINE PIC 9(4) COMP          * Line to be read
 02   SICOL PIC 9(4) COMP           * Start column number
 02   SIATT PIC 9(2) COMP           * No. of bytes per character
 02   SIRFSH PIC 9 COMP                     * Refresh required? ***
```

SILINE and SICOL specify the start line and column to be processed. If either SILINE or SICOL is zero then it will be replaced by the corresponding value for the current cursor position before the operation is performed.

SILENG specifies the number of characters on the screen which are to be processed. If SILENG is greater than the number of characters remaining on the line then it will be reduced to that number. If SILENG is zero, no data is processed (this may be used in conjunction with SILINE and SICOL of zero to find the current cursor position).

7.16.2 The read screen data routine, RDSCR$
++
This is invoked by a call of the form:-

```
CALL RDSCR$ USING SI area [partition]
////////////////
```

where area defines the place where the data read from the screen
////
is to be placed, partition is the partition number to be read (if
/////////
omitted then the current partition is assumed) which must be a
PIC 9(4) COMP field, and SI is the control block described
above. The SIATT and SIRFSH fields are not used with this
routine and therefore can be omitted.

7.16.3 Get screen attribute data, GTSCR$

This is invoked by a call of the form:

```
CALL GTSCR$ USING SI area [partition]
```

where area must be at least three times as long as SILENG, as the number of characters returned will be SILENG multiplied by SIATT. The first SILENG characters are screen data and the following characters (0, 1 or 2 x SILENG) are attribute information held by the System Manager.

7.16.4 Replace screen attribute data, PTSCR$
+++
This is invoked by a call of the form:-

```
CALL PTSCR$ USING SI area [partition]
////////////////
```

This routine puts data and attributes from area to the indicated
////
place on the screen. IF SIRFSH is 0 the screen is not refreshed
to show the change. If SIRFSH is set to 1 then it is refreshed
after the operation.

7.16.5 Exceptions

For RDSCR$, exception 1 will be returned on a pre-V6.0 system, or a system where no screen image is kept, to indicate that the request cannot be honoured.

7.16.6 Programming Notes

Note that any help text displayed is not "on the screen" as far as the system is concerned and hence cannot be read from the screen. If you read an area of the screen containing help text, the "original" overwritten information will be returned instead, that is, the information that will be restored when the screen is refreshed.

Data returned by GTSCR$ should only be replaced to the same screen by PTSCR$ as the format of the attribute information is terminal and implementation dependent.

When replacing an area of the screen, we recommend that all but the last line be replaced with SIRFSH set to zero so that the screen refresh only occurs on the last line, minimising display overhead.

7.17 The Check Screen Routine, SCRN$
++++++++++++++++++++++++++++++++++++

This routine allows you to determine if the partition from which
the program is running is in direct communication with the
physical screen (as opposed to merely having its displays stored
in the screen image buffer, or being in background on a pre-V6.0
system).

7.17.1 Invocation
+++++++++++++++++

The Check Screen routine is invoked by a call of the form:-

CALL SCRN$ [partition number]
////////////////

where partition number is the returned partition number of the
///////////////
partition on which the program is running if it is in direct
communication with the physical screen.

7.17.2 Exceptions
+++++++++++++++++

Exception 1 is returned if the partition is not in direct
communication with the physical screen.

7.17.3 Programming notes
++++++++++++++++++++++++

The use of SCRN$ is intended to completely replace the testing of
$$TASK, which was used on pre-V6.0 systems for essentially the
same purpose. Under V6.0 systems $$TASK is always set to zero,
as displays will not cause the partition to be suspended. Users
should note, however, that on V5.1 and V5.2 concurrent systems it
is not possible to determine if the partition is in direct
communication with the screen, and such systems will always
return without an exception.

The state of communication reported is only valid at the instant
of the check, so if it is important to guarantee that the
partition retains control of the screen the sequence to disable
concurrent switching (#1B35) should be displayed before SCRN$ is
++++++

called. Obviously in such a case it would be necessary to re-enable concurrency at some later point (using the #1B36 sequence).

If the partition number parameter is passed to SCRN$ then SCRN$
////////////////
will return the partition number of the partition on which the program is running provided it is in direct communication with the physical screen (that is no exception is returned).

7.18 The Message Passing Routine, MSG$
++++++++++++++++++++++++++++++++++++++

The message passing routine allows you to send a message to another user of the computer or network on which you are working, or to the status line of your own screen.

7.18.1 Invocation
+++++++++++++++++

The message passing routine is invoked by a call of the form:-

```
CALL MSG$ USING [area [TG]]
//// //
```

where the optional area parameter identifies a PIC X(30) area
////
containing the message to be sent, and the optional TG parameter
//
identifies an area in the following format:-

```
01 TG
02 TGSCNN PIC 9(2) COMP * Screen number (from $$SCNN)
02 TGNID PIC X * Computer-id (from $$LNID)
02 FILLER PIC X(2) * RESERVED, set to low-values
```

The screen number and computer-id identify the target of the message. The appropriate values from $$SCNN and $$LNID must have been saved by some other process so that they are available to the calling program.

If both parameters are omitted the call clears the message area of the screen on which the calling program is running.

If only the area parameter is supplied then the message will be
////
displayed in the message area of the screen from which the calling program is running. This, for example, allows an error message to be sent to the operator irrespective of whether the current partition is selected for display

If the message cannot be sent for some reason (such as there being no computer on the network with a computer-id equal to TGNID) then the operation will nevertheless complete normally.

7.18.2 Exceptions
+++++++++++++++++

Exit 1 will occur if used on a pre-V6.0 system.

Exit 1 will also occur when detectably incorrect parameters are used, such as a non-existent screen number.

7.19 The Supply Key strokes Routine, TYP$/TYPF$
+++

The TYP$ routine allows you to place characters into the console's type-ahead buffer, from where they will be retrieved by subsequent console input commands (ACCEPT, CALL CHAR$, CALL ACCE$, screen formatting, etc.). The TYP$ routine inserts characters inserted at the front of the buffer (before any outstanding type-ahead from the operator) and thus crudely simulate a form of Job Management. The TYPF$ routine inserts the characters after any outstanding type-ahead from the operator.

7.19.1 Invocation
+++++++++++++++++

The TYP$ routine is invoked by a call of the form:-

CALL TYP$ USING area length [partition]
//////////////////////

and the TYPF$ routine is invoked by a call of the form:-

CALL TYPF$ USING area length [partition]
//////////////////////

where the area parameter identifies the characters which are to
////
be inserted into the type-ahead buffer, the length parameter
//////
identifies a PIC 9(4) COMP variable containing the number of characters to be inserted. and partition identifies the target
/////////
partition of your screen (1-9).

7.19.2 Exceptions
+++++++++++++++++

Exception 1 is returned if the operation is not supported, as would be the case on any pre-V6.0 system, or if the partition is half-way through a reply (under V6.2 and later of System Manager, however, an exception is not generated half-way through a reply).
+++

Exception 2 is returned if there is insufficient free space in the buffer for the characters to be inserted - the operation could be retried after setting $$FLSH to 1 to indicate that type-ahead should be cleared, at the expense of losing all typed-ahead input from the operator.

Exception 3 is returned if the number of characters was greater than the length of the type-ahead buffer, and hence could never be inserted. Most systems distributed by Global Software have the type-ahead buffer set to a length of at least 40 characters, but this can be changed using $CUS or Global Configurator.

7.19.3 Programming Notes

++++++++++++++++++++++++

TYPF$ will not work under pre-V8.1 System Manager and the System Manager version flag ($$VERS) must be checked specially for this.

The TYP$ routine will cancel any outstanding type-ahead keyed by the operation or any previous TYP$ call. If you want the operator dialogue to be completed before calling TYP$ you must use the CHECK$/CHKPR$ routines to check that the type-ahead buffer is empty or you must use TYPF$. Note however that TYPF$ adds any type ahead to type ahead already existing in the buffer and so the space available for the TYPF$ string is dependant on any information still there.

7.20 The Text Editing Routine, TXEDT$, TXVAL$ and TXSML$
++

The text editing routine allows you to accept and amend a set of associated lines of text. It behaves as a limited word processor, handling word-wrapping and paragraph management. There are also various special functions available to ease maintenance of the text area. It is functionally equivalent to the $TED command.

7.20.1 Invocation
+++++++++++++++++

The text editing routine is invoked by a call of the form:-

```
CALL TXEDT$ USING text
////
```

where text indicates the text block to be edited by the routine,
////
whose size is indicated in the EE control block which is used by
//
the routine. To optimise memory usage this control block is passed via a COMMON SECTION data area, rather than as a parameter.

The EE block must be contained in a COMMON SECTION called BX$EE
//
located within the calling program. It has the following format:-

```
COMMON SECTION BX$EE
01 EE
02 EELLEN PIC 9(4) COMP * length of lines in text area
02 EELINE PIC 9(4) COMP * start line of text window
02 EECOL PIC 9(4) COMP * start column of text window
02 EEDEEP PIC 9(4) COMP * total number of text lines
02 EEWDEP PIC 9(4) COMP * text window depth
02 EEWUP PIC 9(4) COMP * no./lines to scroll up
02 EEWDWN PIC 9(4) COMP * no./lines to scroll down
02 EELNO PIC 9 COMP * if = 1, display line No.s
02 EEWTOP PIC 9(4) COMP * line number of window top
02 EEDISP PIC 9 COMP * if = 1 window already shown
02 EECLIN PIC 9(4) COMP * current line number
02 EECCOL PIC 9(4) COMP * current column number
02 EEEXIT PIC 9 COMP * highest user exit (max = 9)
02 EEFUNC PIC 9 COMP * -1 = no function allowed
02 FILLER PIC X(8) * RESERVED - set to low-values
02 EEUPDT PIC 9 COMP * if > 0 text has been updated
02 EEEND PIC 9(4) COMP * highest non-blank line
02 FILLER PIC X(5) * RESERVED - set to low-values
```

The total size of the text area is defined by EELLEN and EEDEEP. Note that the actual area in memory must be one line longer than

+++++++++++++++
that implied by the above values (the extra line is used during
text overflow conditions), and hence the text area should be at
least EELLEN * (EEDEEP + 1) bytes long.

EELLEN This is the length of a single line of the text area
++++++
which you must set before calling TXEDT$. This line
length must be between 30 and 191 characters and must
not be wider than the current screen width allowing
space for the start column and any line numbering.
That is for no line numbering EELLEN must not exceed
(screen width +1) - EECOL and for line numbered text
EELLEN must not exceed (screen width - 1) - EECOL - 3.

EELINE This is the start line number of the text area on the
++++++
screen and must be supplied before calling TXEDT$.
EELINE must not exceed the current depth of the screen.

EECOL This is the start column of the text. If line
+++++
numbering is used then this is the start column number
of the first character of the line number. The value
of EECOL must be such that EELLEN does not exceed its
value range as described above.

EEDEEP This is the actual number of lines in the text block
++++++
passed to TXEDT$ and must be set before the routine is
called. This must not be greater than 99.

EEWDEP This is the number of lines in the screen window area
++++++
and this field must be set before calling the TXEDT$
routine. The window depth must be such that (EELINE -
1) + EEWDEP does not exceed the screen depth.

EEWUP This is the number of lines the routine will scroll
+++++
upwards if CURSOR UP is keyed at the top of the window
area. You must set this value before calling TXEDT$
and EEWUP must not exceed the number of lines in the
window area. If the number of lines in the text area is
the same as the number of lines in the window area then
no scrolling is possible and the value of this field
will be ignored.

EEWDWN This is the number of lines the routine will scroll
++++++
downwards if CURSOR DOWN is keyed at the bottom of the
window. You must set this value before calling TXEDT$
and EEWDWN must not exceed the number of lines in the
window area If the number of lines in the text area is
the same as the number of lines in the window area then
no scrolling is possible and the value in this field
will be ignored.

EELNO If this flag is set to 1 before calling TXEDT$ then the
+++++
line number of each line will be displayed prior to the
text of the line. The line number will take up two
characters and an intervening SPACE, 3 characters in
total. If no line numbering is required then this flag
must be set to 0.

EEWTOP This field specifies which line of the text area passed
++++++
should be set as the first line of the window when the
TXEDT$ routine is first entered. This value must not
exceed EEDEEP. This field is also returned from the
TXEDT$ routine and indicates the line in the text area
which was at the top of the window when TXEDT$ was
exited.

EEDISP If this flag is set to 1 before calling TXEDT$, then
++++++
TXEDT$ will not display the text in the window area on
entry. This can be useful if TXEDT$ has exited so that
the program can process a special command as explained
later and then needs to be re-entered without
re-displaying the window. In this case EEWTOP would
also need to be set. If you required the text to be
displayed on entry then this flag must be set to 0.

EECLIN This is the line number within the text area passed to
++++++
the TXEDT$ routine at which you want the cursor to be
positioned on entry to TXEDT$. This value is also
returned by the routine and will then indicated the
line number within the text area of the character under
the cursor on exit.

EECCOL This is the column number within the text area passed
++++++
to the TXEDT$ routine, at which you want the cursor to
be positioned on entry to TXEDT$. This value does not
must not include the three places required for any line
numbering. This value is also returned by the routine
and will then indicate the column number within the
text area of the character under the cursor on exit.

EEEXIT This field must be set to a value between 0 and 9
++++++
indicating the number of special command functions you
might want to process within the TXEDT$ routine. If
this field is set to a n (a number other than 0), when
/
the user keys COMMAND (program function 3 as described
later) 1 to n then TXEDT$ will exit with $$COND = 1 to
/
n respectively. This allows you to defines the
/
processing of these commands within your program,
returning to TXEDT$ if required.

EEFUNC If this flag is set to -1 then none of the special
++++++
TXEDT$ functions will be allowed and only simple
editing will be available. This flag should be set to
0 if functions are to be allowed.

EEUPDT If this field is set to a value greater than one then
++++++
the operator has changed the text (as opposed to merely
inspecting it) and any permanent copy of the text needs
to be updated.

EEEND This field is returned by TXEDT$ and indicated the
+++++
highest line number which has any non-blank text
associated with it, and may be used to avoid keeping
blank lines at the end of the text area.

7.20.2 TXEDT$ Parameter Checking using TXVAL$
++

The TXEDT$ routine assumes that all appropriate parameters will have been established in the EE block before it is called, and performs no checking of the values itself. Consequently if any value is missing or incorrect the routine may fail with a stop code or program check. You should normally validate the parameters yourself, but if you are planning to call TXEDT$ with a fixed set of parameters you may wish to use the TXVAL$ subroutine to check the control block during initial testing. It is invoked by a call of the form:-

```
CALL TXVAL$
```

again requiring the EE block in the COMMON SECTION BX$EE, and will signal a stop code if there is an error in the parameters.

7.20.3 Editing when running TXEDT$

When you invoke the text editing routine the indicated text is displayed in the editing window. You may then over type existing text, delete and insert text, and type new text at the end of the current area, much like a simple word-processor. The following functions are recognised by the routine:-

Function	Action
Program function 1	Insert a space under the cursor
Program function 2	Flip the case of the character
Program function 3	Special editing command (see below)
Program function 4	Next window of text
Program function 5	Previous window of text
ESCAPE	Exit from text editing
RETURN	Move to a new line
RUBOUT	Delete character (or line if blank)

The special editing commands introduced by Program function 3 are as follows:-

Command	Action
B	Draw a box
C	Copy lines
D	Delete lines
E	Go to end of text area
F	Find text string
G	Go to selected line number
H	Display Help
I	Insert a blank line
L	Draw a line
M	Move lines
S	Go to start of text area
U	Wrap up next line

The text editing routine automatically adjusts typed-in text for word-wrapping at end of line. It also recognises and preserves paragraph breaks in the original text. To add a paragraph break press RETURN at the end of a line, and if necessary delete any blank line resulting. To remove a paragraph break use the Wrap Up command.

The line and box drawing commands automatically insert the correct junction characters if lines and boxes intersect.

Commands 1 to 9 are special user exits, and are indicated be setting EEEXIT, as described above, and are for your own special processing.

7.20.4 Small Editing Facility, TXSML$

++++++++++++++++++++++++++++++++++++++
TXSML$ is a cut-down version of TXEDT$ in which no commands or user exits are allowed. To invoke the reduced text editing facility you code:-

CALL TXSML$ USING text
////

where text indicates the text block to be edited by the routine.
////

When editing using the TXSML$ routine function 3 is equivalent to wrap-up

7.20.5 Exception Conditions

++++++++++++++++++++++++++++
Exception Conditions 1 - 9 can occur if the user exception field EEEXIT is set to a number in the range 1 - 9 but the special user command exceptions are not trapped.
Function 3 is equivalent to wrap-up.

7.21 Baseline Handling using BASEL$, BASEX$ and KCR$
+++

The baseline prompt routine, BASEL$, is used to handle multi-reply prompts on the baseline of the screen, conforming to Global Software internal standards. These are documented in full in Appendix D. It is a compact routine with an interface that involves an absolute minimum of code in the calling program. It forces the baseline prompt standards, but for speed and space reasons does not syntax check the prompt with which it is provided. BASEX$ is an extended version of the routine capable of using cursor up and down as selection responses. The KCR$ routine outputs a simple "Key <CR>:" prompt.

All interfacing is done via External Section EE$ZD for BASEL$ (or EQ$ZD for BASEX$), which has the following structure:-

```
EXTERNAL SECTION EE$ZD
01 ZD
02 ZDINX PIC 9(4) COMP
02 ZDPR1 PIC PTR * 10 pointers to display
02 ZDPR2 PIC PTR * areas
. . . *
. . . *
02 ZDPR10 PIC PTR *
02 ZDR
03 FILLER PIC X
03 ZDREP PIC X(30)
03 FILLER PIC X
VALUE ":"
02 ZDTREP PIC S9 COMP * Reply type
* 0 = PIC 9, zero filled
* 1 = PIC X
02 ZDLREP PIC 9(4) COMP * length of reply
02 ZDERR PIC 9 COMP * 0 = normal
* 1 = display same line
* and BELL
* -1 = display same line
* no bell
02 ZDFUNC PIC 9 COMP * function returned
02 ZDFNUL PIC 9 COMP * null function returned
02 ZDFESC PIC 9 COMP * <ESC> function returned
02 ZDDOWN PIC 9 COMP * cursor down function
02 ZDUP PIC 9 COMP * cursor up function
```

7.21.1 Simple use of Options
++++++++++++++++++++++++++++

The simplest interface to the routine, when a fixed prompt is to appear, and the user is to choose between various options is coded thus:-

```
CALL BASEL$ USING prompt
```

//////

The field prompt should correspond to the standards, in
//////
particular:-

* options must be separated by a comma and a space;

* each option should start with the capital letter identifying
it, or with the 5 characters "<ESC>" or the 4 characters
"<CR>". The prompt should not include "Key" or the
following space as these are supplied automatically;

* there can be a maximum of 10 options.

The prompt must be terminated by a colon eg:-

```
CALL BASEL$ USING "Amend, Back, <CR> to continue, <ESC> to
exit:"
GO TO DEPENDING ON ZDFUNC
TO AMEND
TO BACK
TO CONTINUE
TO EXIT
```

BASEL$ returns no errors - if an invalid reply is keyed this is
handled within BASEL$. BASEL$ allows upper or lower case
replies. If a function is chosen then, on exit, ZDFUNC contains
the number of the function.

7.21.2 Prompting for Information

++++++++++++++++++++++++++++++++

If the prompt allows a value to be input as well as a set of
options, then the baseline prompt should start with a request for
this information beginning with a lower case letter. You must
set up ZDTREP to define the type of the reply required (1 for PIC
X, 0 for numeric) and ZDLREP to the length of the reply (up to
30) before the routine is called. If the user keys something
other than one of the specified options then this is considered
to be "information" and is returned in character or display
numeric form in ZDREP. For character information ZDREP is space
filled to the right. For numeric fields ZDREP is 0 filled to the
left and extra bytes at the end are set to spaces. Note that the
user can only enter the number of characters specified in
ZDLREP. For example:-

```
MOVE 5 to ZDLREP
MOVE 1 TO ZDTREP
CALL BASEL$ USING "product code, Amend, <CR> to continue,
<ESC> to exit:"
GO TO DEPENDING ON ZDFUNC
TO PRODUCT
TO AMEND
TO CONTINUE
TO EXIT
```

If the user keys "123" to the prompt then control will pass to the label PRODUCT and ZDREP will contain "00123" followed by 25 spaces.

7.21.3 <ESC> and <CR>
++++++++++++++++++++
Occasionally you will not want the <ESC> option to appear in the baseline prompt. For example, it may have the same meaning as <CR> and would therefore be superfluous. If you do not include an <ESC> option in the prompt then you must tell BASEL$ what to
++++
do if <ESC> is keyed. This is done by setting ZDFESC to the number to be returned (in ZDFUNC) if escape is keyed. This number may or may not correspond to an option that can be keyed some other way. The value of ZDFESC is ignored if you specified an <ESC> option in the prompt.

Note. You must not set ZDFESC to be the number of the
+++++
information reply.

For example, suppose you want <ESC> to have the same effect as <CR>:-

```
MOVE 2 TO ZDFESC
CALL BASEL$ USING "Amend, <CR> to continue:"
GO TO DEPENDING ON ZDFUNC
TO AMEND
TO CONTINUE
```

If the user keys <ESC> to the prompt then control will pass to CONTINUE.

Similarly, if you do not have a <CR> option in your prompt you must set ZDFNUL to the number to be returned if <CR> is keyed. This number must be from 1 to n, where n is the number of options. It cannot be the number of an invalid reply.

This operates in a similar way to ZDFESC except when the number in ZDFNUL correspond to the information option. In this case some extra processing must be done to handle the default information. The first ZDLREP bytes of ZDREP should be set to the default reply beforehand. For numeric replies do not worry about leading spaces. BASEL$ will display the default after the colon just as defaults in maps are displayed, allowing it to be field edited. If <CR> is keyed the default reply is returned in ZDREP - untouched for character fields, zero filled and normalized for numeric fields. For example:-

```
MOVE 5 TO ZDLREP
MOVE 1 TO ZDTREP
MOVE "789" TO ZDREP
MOVE 1 TO ZDFNUL
CALL BASEL$ USING "product code, Amend, <ESC> to exit:"
GO TO DEPENDING ON ZDFUNC
```

TO PRODUCT
TO AMEND
TO EXIT

If the user keys null to the prompt:-

Key product code, Amend, <ESC> to exit:789

00789 will be returned in ZDREP and ZDFUNC will be set to 1.

7.21.4 Baseline Prompts after Errors
++++++++++++++++++++++++++++++++++++

After an error message on the baseline the prompt needs to be
re-displayed. If it will not fit on the same line as the error
message, the KCR$ routine (7.21.7) should be used followed by a
call to BASEL$ as usual to re-display the prompt. However, if
the baseline prompt will fit on the same line, it can be
displayed using BASEL$ with ZDERR set to 1 if a BELL is required
or -1 if not.

7.21.5 Constructed baseline prompts
++++++ ++++++++++++++++++++++++++++

Occasionally certain replies to a particular baseline prompt are
not valid according to the context. For example, the option to
go back a page is invalid when this is the first page of
information. In this case we want to omit the option from the
prompt and forbid the keying of the option.

This could be done by having 2 different prompts for each case,
but this is wasteful in space and becomes much worse if more than
one option is only sometimes present. You can get BASEL$ to
build a baseline prompt by specifying the individual options
which make up a prompt by replacing the line:-

```
CALL BASEL$ USING "prompt"
//////
```

with a number of lines of the form:-

```
POINT ZDPRn AT "part n of the prompt:"
/ ///////////////////
followed by:-
CALL BASEL$
```

Note that "part n of the prompt:" is terminated by a colon not by
/////////////////// +++
a comma and space.

For example, suppose we have the prompt:-

Key Amend, Back, Next, Warehouse, <CR> to confirm, <ESC>:

However, the user may only key B if there is a previous screen, N
if there is a next screen and W if multiple warehousing is in
use. The following code will be used:-

```
POINT ZDPR1 AT "Amend:"
IF there is a previous screen
POINT ZDPR2 AT "Back:"
END
IF there is a next screen
```

```
POINT ZDPR3 AT "Next:"
END
IF multiple warehousing in use
PRINT ZDPR4 AT "Warehouse:"
END
POINT ZDPR5 AT "<CR> to confirm:"
POINT ZDPR6 AT "<ESC>:"
CALL BASEL$
GO TO DEPENDING ON ZDFUNC
TO AMEND
TO BACK
TO NEXT
TO WAREHOUSE
TO CONFIRM
TO                                                          ESCAPE
```

Note that at label BACK there would be no need to re-check that there is a previous screen as BASEL$ would only have returned ZDFUNC=2 if ZDPR2 had been set.

When not in use, pointer fields ZDPR1 to ZDPR10 are set to #FFFF. If you point one of these fields at an option but decide, before calling BASEL$, that the option is not to appear, simply set the appropriate pointer to #FFFF before calling BASEL$ to suppress that option from the baseline.

7.21.6 Programming notes
++++++ +++++++++++++++++
All input fields to BASEL$ are reset each time it is called,
++++
after processing is completed, thus:-

ZDPRn to FFFF
/
ZDTREP to 0
ZDLREP to 1
ZDERR to 0
ZDFESC to 1
ZDFNUL to 1

The baseline is cleared by BASEL$ just before returning to the calling routine.

BASEL$ functions by breaking up a reply pointed to by ZDPRn and
/
pointing the pointers ZDPRn to ZDPRn+m-1 at its m elements. Such
/ ///// /
breaking down takes preference over any pointers that may have been specifically set by the caller. Thus, for instance, in the above example the lines setting up ZDPR5 and ZDPR6 could be replaced by:-

POINT ZDPR5 AT "<CR> to confirm, <ESC>;"

and BASEL$ will set up ZDPR6 automatically.

7.21.7 KCR$
+++++++++++
Also included within the subroutine is the entry point KCR$.
This is invoked by the call:-

CALL KCR$

and returns no exceptions. It displays "Key <CR>:" on the same line as a previous message, and is used to complete error messages or process termination messages.

The display of "Key <CR>:" will be preceded by a BELL. The EE$ZD section need not be used for KCR$ unless the BELL is not required, in which case it may be switched off by setting ZDERR to -1.

7.21.8 Selection processing using BASEX$
++
The BASEX$ routine has exactly the same programming interface as
BASEL$, except that it supports a special selection option using
cursor up and down. To indicate that you wish to use cursor up
and down for selection you must include a prompt segment of the
form "up/down to select", and set ZDDOWN to the correct number
for it. ZDDOWN will be this function number, and ZDUP the next
one (so the single element uses up two functions). Other
processing is exactly as for BASEL$ (including use of information
prompts).

7.21.9 Time-outs
++++++++++++++++
If you set up a time-out using $$ATIM, BASEL$, BASEX$ and KCR$
will return exception 2 to indicate that the prompt has timed
out.

7.22 The Screen Escape Sequences and DSYSR$/ESYSR$
++

The following escape sequences are interpreted specially by System Manager V6.0 and later systems. They may be used to cause particular effects of interest to systems programmers. These sequences should not be displayed on a pre-V6.0 system ($$VERS NOT > 2) as they will have unpredictable effects on your screen.

To display a sequence, use the SVC 25 interface documented in section 7.23.1 or DISPLAY... SAMELINE.

7.22.1 Bypassing the executive
++++++ ++++++++++++++++++++++

Sometimes you will wish to send a particular escape sequence to the screen for some effect not supported in a general way by the System Manager. If you just send the characters they will likely be interpreted by the executive, and unpredictable things will happen. To avoid such problems, you should use one of the following special System Manager sequences:-

#1B27 This turns on bypass mode. All following characters sent to the screen will be sent directly, without interpretation by the executive until a Bypass Off sequence is encountered. Note that after bypass mode has been turned on in this way the screen image may be cleared. On V6.0 systems this may also disable concurrency and system requests;

#1B1B This turns off bypass mode;

#1B7n This turns on bypass mode for the following n
/ /
characters only. This should be used to transmit particular attribute sequences etc to the screen. The characters are assumed to take no space on the screen;

#1B6n As sequence above, except that the characters are
/
assumed to take up one space on the screen;

Setting bypass mode stops the console executive from interpreting all sequences except the sequence for the system request key. To disable interpretation of this key you must call the disable system request key subroutine as follows:-

CALL DSYSR$

The system request key will be disabled and interpretation of the system request sequence by the console executive will now not take place. To enable the system request key again you must call the enable system request key subroutine as follows:-

```
CALL ESYSR$
```

The system request key will now be enabled.

7.22.2 Help Mode
++++++++++++++++
The following sequences are used to control help displays:-

#1B25 Start Help mode, subsequent displays go to the screen
but are not stored in the memory image. Note that
while in help mode concurrency is disabled;

#1B26 Stop Help mode, subsequent displays are treated
normally;

#1B2C Stop help and refresh, as above sequence but the screen
is refreshed to remove Help text. This sequence may be
used to remove Help text from the screen even if not
currently in Help mode.

7.22.3 Non-display Mode
+++++++++++++++++++++++
This is the logical counterpart of Help mode, where displays go
into the memory image but are not displayed on the screen until
it is refreshed. This may be useful if you are building a
complicated display, and do not wish the operator to see it in a
partially formed state. The following sequences are used:-

#1B2A Start non-display mode;

#1B2B Stop non-display mode;

#1B28 As above but refresh the screen, to show the hidden
characters.

7.22.4 Concurrency control
++++++++++++++++++++++++++
The following sequences control the use of concurrency:-

#1B35 Disable concurrency. The screen is fixed in this
partition (only effective if the screen is currently
attached to this partition, so this should be used in
conjunction with SCRN$);

#1B36 Re-enable concurrency.

7.22.5 Window display control
+++++++++++++++++++++++++++++
To start the screen display from a particular character position
when windowing onto a wide screen, use the sequence:-

#1B24yy
//

where the 79/80 column window will be displayed starting at

column indicated by yy, where yy = #20 indicates column 1, yy =
// // //
#21 indicates column 2, etc.

7.22.6 Line insertion and deletion

++++++++++++++++++++++++++++++++++
The following sequences are used for insertion and deletion of lines on the screen:-

#1B39 Insert blank line above current line, scroll the rest of the screen down and lose the last line;

#1B3A Delete current line and scroll the rest of the screen up.

These sequences can be used to manage a scrolled area that is only part of the screen.

Note that operation of these sequences is only guaranteed under System Manager V6.1 and later ($$VERS > 3). Under most System Manager V6.0 systems the sequences will be ignored.

7.23 SVC Statements for Screen Presentation
++

Before using the following SVC statements you should refer to chapter 6 of the Global Cobol Language manual for a general description of these calls.

7.23.1 SVC 25 - Display
+++++++++++++++++++++++

The SVC 25 call is used to speed up displays in certain circumstances. The statement:-

SVC 25 USING area length
//////////

is equivalent to:-

CALL DISP$ USING area length
///////////

Except that it does not check that the length supplied is valid and should not, therefore, be used on the baseline in formatted working. Not only is it significantly faster than the DISP$ call but is also faster than:-

DISPLAY area SAMELINE
////

for fixed length displays that are not on the baseline.

7.23.2 SVC 48 - Calculate Cursor Positioning Sequence
+++

SVC 48 is the positioning algorithm that is loaded as part of the Terminal Attribute Program. It takes a column number supplied in system variable $$X and a line number supplied in $$Y and establishes the corresponding cursor positioning sequence in the PIC X(10) system variable $$XY. The routine is invoked by the statement:-

SVC 48

without any parameters. The length of the positioning sequence, which is fixed for any particular terminal, is given by PIC 9(4) system variable $$L. The maximum possible length is ten bytes. Thus you can use SVC to position the cursor at the position last calculated by using the statement:-

SVC 25 USING $$XY $$L

The main use of this routine is when you frequently display a message at a particular screen location in a performance-critical program, such as some form of text editor. Typically you would calculate the positioning initially and move it to a work area immediately preceding the message you will be displaying. In this way, you not only avoid re-calculating the sequence each time, but you can display both the sequence and the message in a single operation. There is another, PIC 9(4) COMP, system variable called $$11-L which contains eleven minus the length of the positioning sequence; you may find this useful if you attempt to concatenate the sequence with a message area.

7.24 The Print File Viewing Routine, VIEW$

++

The print file viewing routine allows you to enter an inspection mode on your print file at the end of printing the report. The operator can use the inspection facility to move around the report, gathering whatever data he needs, before finally exiting and allowing the report to be completed.

The report inspection facility used ($VIEW) is the same as that used by Global Business software, and the print spooler. Complete documentation of it can be found in the Utilities manual.

7.24.1 Invocation

+++++++++++++++++

You should call the print file viewing routine on completion of your print report, but before you close the print file. It is
++++++
invoked by a call of the form:-

CALL VIEW$ USING fd
//

where fd indicates the Relative-Sequential print file FD used to
//
create your report. The print file FD should still be open, and will remain open when VIEW$ returns control.

In addition to the passed FD there is a complicated interface block which is located in an EXTERNAL SECTION within VIEW$. Much of the information passed in this interface block is constructed by VIEW$ when it is called, and passed on to the inspection facility, but you may wish to establish certain fields within your program to condition the functioning of the inspect.

7.24.2 VIEW$ Interface Block

++++++++++++++++++++++++++++

The PR control block, used to interface with the inspection
//
facility, is laid out as follows:-

```
EXTERNAL SECTION IY$PR
01 PR
03 FILLER PIC X(13) * Reserved
03 PRTITL PIC X(67) * Report title
03 FILLER PIC X(2)
03 PRSTRC PIC X(20) * Stripe control
03 FILLER PIC X(44)
03 PRPHMA PIC 9 COMP * Size of page headings
03 PRSHMA PIC 9 COMP * Size of sub-headings
03 FILLER PIC X(4)
```

```
03 PRSTA PIC 9 COMP * Special stationery flag
03 FILLER PIC X(54)
03 PRPGE1 PIC 9(2) COMP * No. lines to ignore
03 FILLER PIC X
03 PRAVPG PIC 9(4) COMP * Average no. lines/page
03 FILLER PIC X(9)
03 PRPGM PIC X(8) * Name of VIEW request
```

In most cases a program will be able to make use of the default values for the majority of the fields in PR control block.

The fields are used as follows:-

PRTITL used by the inspection facility as the report title on the
++++++
screen. This field should always be established by the calling
program.

PRSTRC stripe control fields, used by the Column display feature
++++++ //////
of the inspection facility. A particular application might wish
to save a sensible stripe pattern and to use this as a default
for displays by the inspection facility.

PRPHMA the number of lines used as a page heading in the report
++++++
(set to 0 if not amended by the calling program). A non-zero
value indicates that this many lines at the start of each page
are not to be displayed on the screen, and that the first such
line contains a page number preceded by the word "PAGE" (or
"Page").

PRSHMA the number of lines used by each sub-heading in the report
++++++
(set to 0 if not amended by the calling program). The
sub-headings are the column headings, and will appear at the top
of the screen (under the title line) at all times.

PRSTA a flag, which should be set to 1 if the report is printing
+++++
using special stationery. This causes the inspection facility to
stop displaying information when the end of a page is reached
(Next will move to the next page), and is done to prevent undue
////
confusion when viewing a report intended to print on a special
form.

PRPGE1 the number of lines to ignore at the start of the file.
++++++
These lines will never be displayed or processed by the
inspection facility. Typically this is used to remove the
alignment pattern from the displayed part of the file for a
special stationery report.

PRAVPG the average number of lines per page in the report. If
++++++
this is left set to zero (the default value) then the inspection
will assume a value of ($$PAGE - 6). The field is only used
during certain page search operations within the inspection
facility, so its value is not crucial, but a sensible value will
improve performance in some situations.

PRPGM the name of the view request program. If this is left set
+++++
to spaces then the $VIEW command from SYSRES will be used as the
view request (but this is only present under V6.2 and later
systems). Alternatively it may be set to the name of some other
program, normally a copy of $VIEW which has been incorporated
into your own program suite.

WARNING: if the value of PRPGM is set to identify a program which
++++++++
has not been linked as a system request then unpredictable
corruption of your program will occur when VIEW$ is called. We
recommend that you only set PRPGM to identify a renamed copy of
the $VIEW command, otherwise the results of a call to VIEW$ are
unpredictable.

7.24.3 Exception Conditions

+++++++++++++++++++++++++++

Exception condition 1 will be signalled if an irrecoverable I/O error arises while attempting to build the list of file segments for a report file on a spool unit.

Exception condition 2 will be signalled if VIEW$ cannot initiate the inspection facility (due to the report being sent directly to a printer, or to VIEW$ being run on a pre-V6.0 system).

Note that if VIEW$ cannot load the view request then control will return normally, as if the operator had exited from the inspection facility.

7.24.4 Programming Notes

++++++++++++++++++++++++

The inspection facility is a user system request overlay (the view request program) which is invoked by VIEW$ (the System Manager command $VIEW is normally the program invoked). Both the FD passed to the system request, and the PR control block, need to be outside the system request area, and VIEW$ will handle this by exchanging these blocks with areas in the first #200 bytes of the user area if necessary. If there is an intercept routine on the print file FD, this must also be outside the system request area, and in addition must not occupy the first #200 bytes of the user area in case VIEW$ needs to exchange it. In practice we would recommend that you remove any intercept routine on the print file FD over the call to VIEW$.

As the VIEW$ request loads a program to be the view request (normally the $VIEW command) any program library attached by the calling program will be closed until the next module is loaded from it.

Note that the view request does not manipulate the width of the screen. You should establish the screen width you wish to use (by calling WIDE$) before invoking the inspection facility by means of a VIEW$ call, and should reset it subsequently.

7.25 The Display Buffer Check Routine, DBUF$
+++

The display check routine can be used to determine if a partition has finished displaying characters from its display buffer to the screen. This is particularly useful if the program is intending to read data from the partition in question, as if there are still characters in the buffer they will not be retrieved by a RDSCR$ or GTSCR$ call.

7.25.1 Invocation
+++++++++++++++++

The display check routine is invoked with a call of the form:-

```
CALL DBUF$ USING part
////
```

where part is the partition number whose display buffer is to be
////
checked.

7.25.2 Exceptions Returned
+++++++++++++++++++++++++++

The routine returns exception condition 1 if it has been run on pre V8.0 System Manager.

It returns exception condition 2 if there are still characters in the display buffer for the partition in question. A typical action in this case would be to SUSPEND for a short time and then call DBUF$ again.

If no exception is returned then the display buffer for the partition in question is currently empty, and data may be read from the screen in relative safety.

7.25.3 Example
++++++++++++++

A typical example of a piece of code using DBUF$ can be described as follows:

```
AA100.
CALL DBUF$ USING partition
/////////
ON EXCEPTION
IF $$COND = 1 GOTO AA200 * Read the screen anyway
IF $$COND = 2 * Screen display has not
* completed
SUSPEND 1
GOTO AA100
END
END
```

```
AA200.
CALL RDSCR$ USING SI area partition
//////////////
```

7.26 Set Screen to Formatted Mode Without Clearing, FMODE$

++

A screen is normally set to formatted mode by the use of the CLEAR verb. This clears the screen and sets formatted mode. There may be occasions where you may not want the screen to be cleared when setting formatted mode. In order to do this you will need to use the FMODE$ subroutine.

7.26.1 Invocation

+++++++++++++++++

The subroutine is invoked by a call of the form:-

```
CALL FMODE$
```

7.26.2 Exception conditions

+++++++++++++++++++++++++++

Exception condition 1 will be returned if the screen cannot be put into formatted mode as on a teletype screen.

2

```
+ +

Global Software Friday 25/11/88 11.01

*******************************************************************************


 Distribution

 ***************
 Label Printer

 ********************************
 Today's Orders. . . . . . . . . 1
 Specific Orders . . . . . . . . 2
 Software Generation Masters . . 3
 Configuration Volumes . . . . . 4
 Production Volumes. . . . . . . 5
 Preformatted Diskette Masters . 6
 Duplicator Program Diskettes. . 7
 Software Stock Items. . . . . . 8
 Preformatted Diskette Stock . . 9
 Exit. . . . . . . . . . . . .<CR>

 Please select a Function :_
```

8. AutoGuide and Help Screens

8.1 Introduction

8.1.1 Overview
AutoGuide is used to create context-sensitive Help guides for Global applications software, as well as producing demonstration and training aids for use on all System Manager systems. Each "guide" consists of a number of linked display pages or "books". The user can follow his or her own path through the guide by selecting from options offered on each page.

The dialogue is designed to be extremely simple to use; all replies are made by pressing a single key, and no special conventions have to be learnt. The pages can also be arranged to follow each other automatically after pre-set time intervals, so that a continuous presentation can be given without requiring any intervention from the user.

AutoGuide has its own page editor which allows guides to be created and amended interactively; the cursor can be moved around the screen at will, and text can be over keyed, inserted and deleted. Pages can be windows (useful for creating Help screens), full screen or print pages 132 characters in width (allowing sample printouts to be produced during a demonstration).

AutoGuide has important applications in the design of computer systems, since it can be used to create very quickly a prototype of a proposed system which demonstrates the displays and reports to prospective users. These displays can then be changed easily to reflect users' suggestions.

8.1.2 Installation
AutoGuide is distributed as part of the Common Programming Utilities, and consists of the following files:

AG	AutoGuide Program
AGEDIT	AutoGuide Editor
AGED01	Overlay 1
AGED02	Overlay 2

8.2 Guide Structure

8.2.1 Pages
A guide can contain up to 500 pages (or 250 pages if it is a Help guide). Each page is identified by a one or two character name, which must be unique within the guide. Pages can be Print Screens, Windows or Full Screens. A guide may consist of any combination of these.

Print Screens	Print screens are useful for creating sample print outs for use during demonstrations or mock-ups of proposed systems. Print screens are 132 columns across and can therefore only be created and edited on screens that allow wide mode.
Windows	A window is an area superimposed on the screen that can be any size up to the size of the screen and can be positioned anywhere on the screen. A window can also be enclosed within a box to set it

apart from any text already on the screen. Windows are most useful in Help guides where just a few lines of context-sensitive information need to be displayed in an otherwise unused area of the screen.

Full Screens	Full screens are most useful when proposed screen displays or proposed programs and systems need to be simulated.

8.2.2 Page Linkages

When AutoGuide is started the first page of the guide is displayed automatically. From this point on, the path of the dialogue is determined by the user's reply in conjunction with the page linkages on the currently displayed page.

As many linkages can be set up for a page as will fit on the .END line at the bottom of the screen. The order in which page linkages are specified is immaterial.

Each page linkage can take one of the following forms:

c/dd	c is any single character reply except '?', '+', '-', '$' or '.' and dd is the name of the page to be displayed when that reply is keyed;
?/dd	causes page dd to be displayed when any reply not explicitly defined by other page linkages is keyed;
$/ddnn	causes page dd to be displayed if no reply is keyed within nn seconds of the previous display; nn can be in the range 1-99. This linkage will not work with Help screens;
+/dd	causes page dd to be displayed immediately, without waiting for a reply. The new page scrolls up from the bottom of the screen so that a continuous long display can be generated.

A number of reserved page names have special meanings in the above linkages:

$Bn	redisplay the nth page displayed before the current one. (n = 1 if not supplied);
$E	exit from AutoGuide;
$N	exit from AutoGuide but do not clear screen until after a reply is made to the outstanding prompt (used when creating Help screens).
$Pdd	print page dd leaving the display unchanged;
$Cdd	execute a sub-dialogue of pages starting at page dd.
$R	return from a sub-dialogue to the page from which it was initiated.

8.2.3 Examples of Linkages

Page AA has the following linkages specified:-

```
1/A3 2/B1 X/$E ?/BB
```

If the user keys 1, page A3 is displayed; if he keys 2, page B1 is displayed; if he keys X, AutoGuide terminates; and any other reply causes page BB to be displayed.

Page BB has the following linkages:

```
?/$PCC $/DE5
```

If any reply is received within 5 seconds, page CC is printed, the screen remains unchanged, and timing starts again. After 5 seconds with no reply, page DE is displayed.

Page CC has the following linkages:

```
B/BB N/DD E/$E
```

If B is keyed page BB is displayed; if N page DD is displayed and if E is keyed then the guide terminates.

8.3 Creating and Amending Guides

8.3.1 Starting the AutoGuide Editor

The AutoGuide editor is named AGEDIT. Together with the associated overlays AGED01 and AGED02, it should already have been installed on a diskette or on the program unit of a hard disk. After mounting the diskette or disk, you key AGEDIT in response to the GSM ready prompt or a System Manager menu selection prompt in order to start the editor.

You are then prompted for the name and unit address of the guide to be created or amended. If the guide is present on the unit specified, the page index is displayed as explained below.

If the named guide is not present, you are asked to confirm that you wish to create a new guide. You key Y to confirm this, or <CR> or any other single character to return to the file name prompt.

You can key <ESCAPE> to the file name prompt to exit from the editor.

8.3.2 The Page Index

When you start to create a new guide, or to edit an existing guide which contains no pages, you are only given the options of inserting a new page, abandoning the edit or ending the edit.

If you choose to edit an existing guide of one or more pages, the page index is displayed. This lists the names of the pages within the guide, and invites you to select the next editing operation.

The options available are as follows:

I	insert a new page
D	delete a page
U	update a page
A	abandon the edit (i.e. leave guide in the state it was before the edit was started)
E	end the edit

These options are described in the following sections. After completing each operation (other than A or E, which return to the ready prompt), the page index is re-displayed to reflect the change and you are prompted for the next operation.

8.3.3 Inserting a New Page

If you key I to the instruction prompt, you are asked to specify the name of the new page to be inserted. If the name you key is already present in the page index, you are asked to re-key it. You can also key <CR> to return to the instruction prompt.

You are then asked for the Book Title. This can be up to 30 characters in length or you can key <CR> to leave it blank. This title will be displayed on the .BOOK line if you print the guide using $PRINT.

Unless you are inserting the first page into a guide, you are asked to specify the position of the new page within the guide, by keying the name of the page which the new page is to precede, or <CR> if the new page is to be added to the end of the guide. This allows you to preserve a logical sequence of pages in the guide if you so wish.

Next you are asked for the screen colour combination. This can be 1 - 8. For help screens and windows we recommend that you use colour combination 6.

The next prompt asks you whether the page is to be a Print screen, a Window or a Full screen.

Print Screen. If you key P you are asked if you want to copy an existing page or the screen being displayed in another partition. (If the screen you are working on does not allow wide mode then you will be told at this point that you CANNOT EDIT PRINT FORMATS.) You are then asked:-

Do you want to start a new print page?

Type Y if you want the printer to form feed before printing this page, N or <CR> if not. The screen is cleared allowing you to create and edit the new page, as described in paragraph 8.3.6.

Window. If you key W you are asked if you want to copy a previous page (useful if the page you are creating is only slightly different) or if you want to copy the screen being displayed in another partition (allowing you to design a Help screen that will not obscure vital information on the screen it will be called from). Keying <CR> will give a blank screen.

Next you will be asked to indicate the top left hand and bottom right hand corners of the window by positioning the cursor and keying W. You are asked to confirm this and then you are asked if you want a box drawn around the window or not. After this you are asked to indicate the text area by positioning the cursor in the appropriate corners and keying W.

When these have been confirmed you are asked whether you want to use "top/bottom exchange". This is useful when creating Help windows as it allows the window to occupy the half of the screen opposite to that in which the cursor is located when the guide is called.

Finally you are asked whether you want the text area cleared or not. The screen is then cleared (if required) to allow you to create and edit the new page.

Full Screen. If you key F you are asked if you want to copy a previous page (useful, for example, if the screen you are creating is only slightly different) or if you want to copy the screen being displayed in another partition. Keying <CR> will give a blank screen.

Next you are asked to enter the screen depth. You can key <CR> to use the default of the current screen's depth.

The screen is cleared to allow you to create and edit the new page.

8.3.4 Updating a Page

If you key U to the instruction prompt, you are asked to specify the name of the page to be updated. If the name you key is not present in the page index, you are asked to re-key it. You can also key <CR> to return to the instruction prompt. The name of the page is displayed and can be amended.

The screen is cleared and you are asked if you want to amend the options. The screen colour and screen depth can be amended if the page is a full screen. If it is a window then the screen colour, position of the window, position of the text and size of the window can be changed.

The selected page is then displayed and you can edit it. The <ESCAPE> function is used to return to the page index when the amendments are completed.

8.3.5 Deleting a Page

If you key D to the instruction prompt, you are asked to specify the name of the page to be deleted. If the name you key is not present in the page index, you are asked to re-key it. You can also key <CR> to return to the instruction prompt.

The selected page is then displayed for confirmation that you really want to delete it. If you key D to the subsequent prompt it is deleted from the guide and the page index is re-displayed without the deleted page. Key <CR> to avoid the deletion and return to the instruction prompt.

8.3.6 Editing a Page

Pages are created and amended by keying the required text directly onto the screen, using the cursor control keys to move around the screen as necessary. Individual characters or complete lines can be inserted, deleted, copied or moved and 132 character wide print pages can be edited by switching the display 'window' from one half of the page to the other.

A number of commands are available when editing pages. These are called by pressing the Program Function 3 key followed by the appropriate letter:-

B	Draw (or delete) a box;
C	Copy lines;
D	Delete lines;
F	Search the screen for a specified piece of text;
G	Go to. Places required line at the top of the screen;
I	Insert a blank line;
L	Draw (or delete) a line;
M	Move lines;
S	Go to the start;

U	Wrap up the next line to current cursor position.

Keying <ESCAPE> will take you out of edit mode to the prompt:-

Key U to Update, A to Abandon, E to edit .END line, or <CR> to continue

Type U if you wish to continue editing, A if you wish to abandon this particular screen, E if you want to amend the page linkages specified on the .END line or <CR> if you want to go back to the page index.

The AutoGuide editor is basically the same as $TED, so you may find it useful to refer to the section on this command in the Global Operating Manual.

8.3.7 Specifying Page Linkages

Each page ends with a line which shows its page linkages. This line is identified by the characters '.END' in columns 1-4. This line is displayed when <ESCAPE> is keyed and is edited by typing E to the appropriate baseline prompt. The remainder of the line up to column 132 contains the page linkages in the form described in Sections 8.2.2 and 8.2.3, separated from each other by one or more spaces. The sequence is immaterial, unless two linkages conflict in which case the leftmost one will be used.

The editor does not check the validity of the linkages specified, and so each linkage should be checked by running the guide after completing the edit.

8.3.8 Ending the Edit

If you key E to the instruction prompt, the amendments made during the session are consolidated into a new copy of the guide. This consolidation process may take several minutes for a large guide. The old copy is renamed as a back-up by changing the file prefix from "S." to "B.". Thus the back-up for S.DEMO would be B.DEMO. Any previous back-up of the same guide is automatically deleted.

If there is insufficient space on the same volume to hold the new guide, you are asked to specify a new unit address. This prompt is repeated until you provide a volume with sufficient space for the new guide.

8.3.9 Printing the Guide

The System Manager utility $PRINT (described in the Global Operating and Utilities Manuals) can be used to print the guide. Pages are printed in index sequence separated by '.BOOK' and '.END' lines. The page name is shown on the '.BOOK' line. If any of the pages employ boxes or lines then the appropriate printer customization must be carried out before they can be printed.

8.4 Help Within Programs

8.4.1 Associating Help Guides with a Menu Entry

Help guides can be associated with menu entries during menu customisation using the menu maintenance utility (MN).

You need to specify the help library and book name associated with a particular menu line.

If you type $F or key <CR> to the help book prompt, then the first page displayed when the guide is called will be the first page in the page index.

The specified Help guide will now be entered when <SYSREQ> H is keyed.

8.4.2 Associating Help Guides with a Program

To attach a help guide that can be entered using <SYSREQ> H from a specific program the following two system variables need to be set up:

$$HFIL	PIC X(8)	* Name of help guide file
$$HUN	PIC X(3)	* Unit of help

These variables are only available under V6.0 and later systems, so you should check that $$VERS is greater than 2 before using them.

8.4.3 Help Book Name

The system variable $$HBK is a PIC X(2) variable that contains the name of the first help book to be displayed if <SYSREQ> H is keyed. It must be established if a help guide is going to be used as its initial contents are unpredictable. By altering the value of $$HBK according to context, the program can vary the help displayed.

Note that this system variable is available under all versions of the System Manager, although it only has effect under V6.0 or later. Therefore there is no need to test the value of $$VERS before setting this up.

8.4.4 Help Routines

The major limitation of the help guides is that the text is fixed. Often you will want to display help that contains variable information about data or parameter values. For example, the spooler's help screen of options shows the currently selected defaults. Such displays can be achieved by having a help routine in the program.

The address of the routine must be set up in $$HPTR, a PIC PTR variable (available in V6.0 and later versions only). When the special $H help book is selected within a help path, this routine is called and is a passed a single PIC X(2) parameter containing the name of the previous book (this allows the routine to display different information depending on the context). When the routine exits, the control passes to the help book now in the parameter, so the routine can alter the flow by updating it. If $H was the first book selected, the parameter will contain $E on entry.

Because the routine is called from the help system request, it is essential that the routine and all its data areas are outside the area occupied by the requests, i.e. either below #1000 or above #7000. It must also be permanently resident. In practice, it may be easiest to load it onto the user stack at the start of the program. If the routine is made unavailable for some reason, e.g. overwriting it in a space critical overlay, the pointer should be set to #FFFF.

Note that if a $H help book is encountered and there is no help routine set up ($$HPTR is #FFFF) then the help system will look for a book called $N in the guide. This allows you to set up a "shadow" book to be used when the help routine is not available. This is also used by the spooler.

8.4.5 Programmed Help Invocation

Rather than the user having to key <SYSREQ> H, the program can invoke help automatically using the CMND$ routine (described in the System Subroutines Manual section 5.5) by a call of the form:

```
        CALL CMND$ USING "H"
```

Thus you might program a response of H to select help or bring up help automatically if an error is detected.

8.4.6 Direct Help Displays

It is quite possible for a program running under V6.0 or later to display help directly without using a help guide at all. A program can select help mode by a display of the form:-

```
        DISPLAY #1B25 SAMELINE                  * Help on
```

This causes all subsequent displays not to be recorded in the screen image, e.g. that the previous state of the screen can be restored at the end by refreshing the screen. When in help mode, concurrency and system requests are disabled.

To exit from the help mode and restore the original screen, you perform a display of the form:

```
        DISPLAY #1B2C SAMELINE                  * Help off and refresh
```

Alternatively, you can leave the help information on the screen until the user changes partition or refreshes the screen by a display of the form:

```
        DISPLAY #1B26 SAMELINE                  * Help off, no refresh
```

Very often you will want to leave the help on the screen until the user has replied to the current prompt; this is done by setting the PIC 9 COMP variable, $$HCLR, to 1, displaying #1B26 (help off, no refresh) and then repeating the accept.

If you select colours in help mode, then on exit from help mode you should reset the colours to standard by a call of:-

```
        CALL COLOR$                         * Standard colours
```

to resynchronise the System Manager with the colour restored by the refresh. You may then need to re-select the desired colour.

If you use Help off and then Help on to clear the current help from the screen in the middle of displays, you will need to make a parameterless call of COLOR$ followed by re-selecting the required colour.

If the help is displayed within a system request on V7.0 and later System Manager, there should be no need to switch help on unless you do not want the user to be able to switch partitions or run another system request. With the screen preservation handling under V7.0 and later systems, a value of 0 in $$HCLR causes the screen to be replaced as soon as the system request is exited from and a value of 1 causes the screen to be replaced after the next accept.

Under V8.0 and later systems a value of -1 in $$HCLR causes the screen not to be replaced after exiting from a system request.

8.5 AutoGuide for Demonstration Purposes

AutoGuide can be used for demonstration purposes, such as producing a series of connected screens giving an approximation of what a proposed suite of programs can do. It can also be used as a teaching aid and as a way of testing help libraries.

8.5.1 Setting Options

The AutoGuide program is initially named AG, and you should copy it back to this name, if you have renamed it, before establishing a new guide. When the program is first used with a particular guide, the guide name and unit must be typed in. If the named guide is not found on the unit supplied, you are told so and asked to type in the guide name and unit again.

You are then prompted for the time delay before returning to the start. You can increase or decrease the delay by keying a larger or smaller value, respectively, or key <CR> to leave the value unchanged. You are also asked for the name of the book you want to start with. This is defaulted to the first book.

Once the guide has been found, the start page of the guide is displayed and the operation continues as described below.

8.5.2 Selecting Pages

When the AutoGuide program is started the first page of the selected guide is displayed automatically.

The remainder of the operation should be self-explanatory, provided that the guide has been designed to show the options available on each page. It is only necessary to press the specified key (without pressing <CR>) to select the option required. If the page has an automatic linkage, no action is necessary; the next page will be displayed after a suitable time delay.

Several keys are reserved for special purposes, as follows:

- <ESCAPE> exits from the guide to the ready prompt, with a message explaining how to restart the guide.

- '.' returns you to the page displayed before the current one. Up to ten steps backwards can be made in this way.

- '-' holds a page with an automatic linkage. Timing is restarted when any other key is pressed.

9. Interfacing to the Menu Handling System

9.1 Introduction
+++ ++++++++++++
As part of System Manager V6.2 a new menu handling system has been produced. This system is intended to deal with all aspects of displaying the menu on the screen and accepting the operator choice of function, permitting you to isolate the menu handling from the rest of your application. The menu handler can also be run in a mode where it simulates the previous $MENU program, providing system level menu facilities. Since both system and application menus are now controlled by the same software, it is easier to maintain consistency of the user interface.

A detailed discussion of the creation and amendment of menu systems is to be found in the Utilities manual, and you are recommended to familiarise yourself with this before attempting to construct application menus using the new menu handling system.

This chapter begins by discussing how you would write an application program to make use of the menu handling system, detailing the control blocks and programming interfaces required, and finishes by covering some special purpose system routines which enable you to construct more complex interfaces into the menu handling system.

9.2 Interfacing to the Menu Handler, $MH
+++ ++++++++++++++++++++++++++++++++++++
The basic interface between an application and the menu handling system is a piece of software known as the menu driver. The menu
///////////
driver is responsible for invoking the menu handling system, and for reacting to the information returned from the menu system.

Communication between the driver and the menu system is handled via an MI interface block, a linkage section item which overlays data within the data division of the menu handler. The first action of the driver program must be to establish the location of this data block, by loading and executing the menu handler in a special way. Once this has been achieved the driver moves appropriate values into fields within the interface block, and then invokes the menu handler again to elicit a response from the operator.

Control will remain within the menu handler until the operator successfully selects a legitimate option from the menu. Where a selection from the menu indicates a subsidiary menu, then control remains within the menu handling system until a legitimate option is selected from that menu, and so on.

When control returns to the menu driver a valid option from a menu will have been chosen, and various data relating to that option will have been retrieved from the menu file and placed into the interface block. The driver must now direct control to the appropriate portion of the application, which will usually mean chaining to the appropriate program.

Overleaf is a brief example driver program together with a description of the various interface fields returned from the menu handler.

```
PROGRAM DRIVER
DATA DIVISION
*
77 MIPTR PIC PTR * Pointer to MI area
77 MIPTRX REDEFINES MIPTR PIC X(2)
*
LINKAGE SECTION
01 MI BASED MIPTR
03 MIVERS PIC 9(2,2) COMP * Version, must be 6.0
03 MIMFIL PIC X(8) * Menu file name
03 MIMUNI PIC X(3) * and unit
03 MICNAM PIC X(30) * Company name
03 MIANAM PIC X(30) * Application name
03 MITITL PIC X(30) * Menu title
03 MISVSR PIC X(8) * Supervisor program
03 MIPROG PIC X(8) * Program to CHAIN
03 MIACCE PIC PTR * Accept routine
03 MIEXIT PIC 9 COMP * Accept control
03 MICLEA PIC 9 COMP * Clear screen control
03 MIHELP PIC 9 COMP * Help mode control
03 MITOP5 PIC 9 COMP * Display top lines
03 MITYPE PIC 9 COMP * Menu type
03 MIFUNC PIC 9(2) COMP * Returned function
03 FILLER PIC X * Reserved
03 MIAP6B PIC X(6) * Saved application data
03 MISACR PIC PTR * System access control
03 MICLOS PIC PTR * Close down routine
03 MISECR PIC PTR * Read security code
03 MISECW PIC PTR * Write security code
03 MIVALI PIC PTR * Response validation
03 MIMAIN PIC 9(4) COMP * Main menu?
03 FILLER PIC X(14) * Reserved for expansion
*
77 MXSACR REDEFINES MISACR PIC X(2)
77 MXCLOS REDEFINES MICLOS PIC X(2)
77 MXSECR REDEFINES MISECR PIC X(2)
77 MXSECW REDEFINES MISECW PIC X(2)
*
PROCEDURE DIVISION
*
MOVE #0000 TO MIPTRX * Set special value
EXEC "*MH" USING MIPTR * Run menu handler
* points MIPTR at MI block
* or "$MH"
MOVE #0000 TO MXSACR MXCLOS MXSECR MXSECW
. . .
* code to establish fields in MI
. . .
CALL $$EPT USING MIPTR * Invoke handler to display
* and get response from menu
```

```
ON EXCEPTION
* See section 9.2.3 for exception handling
END
. . .
* code to process results from handler
. . .
CHAIN MIPROG * Execute selected program
*
ENDPROG
```

9.2.1 The MI interface block
++++++++++++++++++++++++++++
Many of the fields in the MI block are established by the menu
driver to condition the actions of the menu system. Others are
returned by the menu system in response to the function chosen by
the operator, and these are identified in the list by having a
preceding asterisk (*). The use of the various fields is
explained below:-

MIVERS is a version number field, used to ensure integrity of
++++++
data between different versions of the menu handling system. The
menu driver should set the value of MIVERS to be 6.0, and the
menu handler will check it to ensure that it is a version with
which it is compatible. The value of MIVERS will only change if
significant differences in the menu system files appear between
two versions of the software.

MIMFIL and MIMUNI are set to identify the name and unit of the
++++++ ++++++
menu file to be used by the menu system. These must be
++++
established by the driver before the menu handler is invoked to
display the menu.

MICNAM and MIANAM are used to provide a company name and
++++++ ++++++
application name in the menu headings. If MICNAM is not set up
then $$SNAM is used to provide this element of the menu ("GLOBAL
SOFTWARE" on pre-V6.0 systems) and MIANAM is ignored.

MISVSR is the name of a temporary supervisor program to regain
++++++
control if system commands are run from the application menu,
which should be set up by the driver if required. Normally you
would set this to be the start-up program for the application so
that the application would be correctly reloaded. Note that when
the supervisor is returned control LOGOF$ is called allowing the
user to log off. (See Systems Subroutines Manual)

* MIPROG is the name of the program identified by the selected menu
++++++
function. The final action of the menu driver is typically to
CHAIN or EXEC this program. However, this field is not used when
+++
a menu line of type 'S' is selected. A stand-alone program is
always run directly.

* MIACCE is the accept continuation routine, returned by the menu
++++++
driver only if MIEXIT indicates that you wish to regain control

until the operator keys something.

MIEXIT is a flag, which the menu driver should set to 1 if it
++++++
wishes to perform background processing while awaiting a response to the menu from the operator. The menu handler will exit with exception condition 1 when background processing can be performed. The driver may perform simple housekeeping functions (taking care not to corrupt the menu handler code or data) and must periodically check for type-ahead using CHECK$. If type-ahead is present control must be returned into the menu handler by calling MIACCE, and routing subsequent control to the same handling performed after the original call to the menu handler.

MICLEA is a flag which should be set to -1 if you do not wish the
++++++
screen cleared before the menu is displayed. You would use this
if you wished to leave some other information on the screen
during the display of the menu, eg the READ MAIL box in
Organiser.

MIHELP is a flag which should be set to 1 to cause the menu to be
++++++
displayed in help mode. Help mode will be turned off, and the
screen refreshed, before control returns after the response is
keyed. You would use this to display a pop-up menu using the
menu handler.

MITOP5 is a flag used to control the display of the menu
++++++
heading. If it is set to zero (the default) then the whole menu
heading is displayed. If it is set to -1 then none of the menu
heading is displayed. If it is set to 1 then only the fourth and
fifth lines of the menu heading are displayed.

MITYPE is a special flag used by the menu maintenance software.
++++++
It should not be used by an application.

* MIFUNC is the function number keyed by the operator, returned in
++++++
case it is more sensible to perform subsidiary processing on the
basis of line number rather than program name.

* MIAP6B is 6 bytes of application specific data, returned to you
++++++
by the menu handler when the menu line is selected.
Conventionally these 6 bytes are organised as two PIC 9(2) COMP
fields followed by two PIC X(2) fields. Data returned in these
fields can be used to select a particular function or for
security checking etc.

MISACR is a pointer to a routine within the menu driver which is
++++++
called by the menu handler to determine which system access
control codes are in use. See section 9.2.4.

MICLOS is a pointer to a routine called by the menu handler
++++++
before it executes a system command or stand-alone program. This
routine is responsible for taking any appropriate termination
action (such as closing the security file, or saving system
defaults). See section 9.2.5.

MISECR and MISECW are two routines which are called respectively

++++++ ++++++
to read and to write the security sequence number whenever the
menu file has been amended. See section 9.2.6.

MIVALI is a routine called before the menu handler validates the
++++++
response to the menu prompt, to permit you to take special action
if certain responses are keyed (you might use this to ensure that
system command are disabled, or to implement a hidden special
password prompt). See section 9.2.7.

MIMAIN is the starting menu number, the number of the menu in the
++++++
file to be used as the main menu. If not set a value of 1 is
assumed (ie the first menu in the file is the main menu). Use of
this field enables you to have menus for a number of separate
applications with a shared data unit in a single menu file.

The routines indicated by MISACR, MICLOS, MISECR, MISECW and MIVALI do not have to be set up by the menu driver. If the pointers to these routines are not set up (or are set to low-values) then the menu handler will make no attempt to call them.

9.2.2 Using the returned information
++++++++++++++++++++++++++++++++++++

The menu handler essentially returns three pieces of information to the driver, the name of the program to CHAIN or EXEC, the line number selected by the operator, and six bytes of application specific information attached to the selected function.

A typical application would therefore perform such special processing as was indicated by the application specific information (loading one of a number of services overlays on the basis of a two character identifier for example), possibly perform some processing based on the line number and then CHAIN or EXEC the indicated program.

If it is necessary to load service modules or other programs over the area occupied by the menu handler (the menu handler is linked to start at #4700, and occupies memory locations from there onwards), then the values returned via the MI block must first be saved in some local data area (as they lie within the menu handler, and might well be overwritten by loading programs).

Note: It is not possible to run a next menu (line type N) from
+++++
within an application menu an all sub-menus must be of type "M". Also a return line from the top level application menu (line type "R") will always return to the top level system menu an not to any subsidiary menus that it might have been called from. It is preferable to have application menus run from the top level system menu.

9.2.3 Exceptions signalled by the Menu Handler
++

There are three exception conditions which can be signalled by the menu handler.

Exception condition 1 is signalled to indicate that a background activity may take place, while awaiting operator input. Such an exception will only be signalled if MIEXIT was set to 1. After performing the background activity, or if operator input is detected during a lengthy process, the background processing should call MIACCE to continue the menu processing. The call to MIACCE can return the same three exceptions as the EXEC of the menu handler.

Exception condition 2 is signalled if the menu handler detects data corruption in the menu file. The driver should indicate to the operator that a restore of the data is required, and possibly initiate such a process.

Exception condition 3 is signalled if a serious program error has been detected by the menu handler, which will not be helped by restoring the data. Such errors include an incorrect version number in MIVERS, the alteration or deletion of a critical assignment between two invocations of the menu handler, the user area being too small to permit the running of the menu handler, and serious I/O errors on the menu file. An explanatory message will have been displayed, so the driver should probably simply produce a 'Key <CR>' prompt and terminate.

9.2.4 Usage of System Access Control Codes
+++

When you define your menu, you may associate system access control codes with particular lines of the menu. The two character codes are used in conjunction with the routine pointed at by MISACR to cause the menu handler to prohibit ('star out') certain menu functions under specific conditions.

For example, you might have a system parameter which defines whether picking lists are in use. If they are not you would wish to prevent entry to functions which deal with picking lists. Similarly you might wish to prohibit certain end of session activities unless a recent back-up has been taken.

For each situation where you wish to control access to a function, you allocate a system access control code and associate it with the appropriate menu entries. In the menu driver you point MISACR at an access control routine, of the following general format:-

```
LINKAGE SECTION
77 L-SACR PIC PTR * to menu handler routine
*
PROCEDURE DIVISION
ENTRY UA-ACCESS-CONTROL USING L-SACR
*
IF condition for refusal of access
/////////////////////////////
CALL L-SACR USING access-code
//////////
END
. . .
* possibly further access code checking
. . .
EXIT
```

For each system access control code in use by the application you determine whether associated functions are appropriate. If they are not then you disable them by calling the menu handler routine, passing it the two character access-code for the control
//////////

code.

The menu handler routine will return exception condition 1 if the access code you specify is not defined within the menu file. This indicates a serious error in setting up the menu file, and should not just simply be ignored.

Note that although this handling should prevent access to
inappropriate functions within the application, it is still
prudent to check at the start of each function, and to reject it
with a polite message if it is in fact not appropriate to select
it.

The access control routine will be called whenever the menu
handler is invoked to display the main menu, or whenever control
is returned back to the menu from a function.

9.2.5 Saving application defaults over system commands
+++

If your menus are set up so that system commands may be run from
them, your application will need to be reloaded once the system
command has completed (MISVSR contains the name of the program to
be loaded to manage reloading of the application).

Before the system command is run, the menu handler calls the
routine indicated by MICLOS, and this may be used to save up to
32 bytes of default information within the menu handler. The MI
block is extended in the following way for this purpose:-

```
03 FILLER PIC X(14) * Reserved for expansion
* end of basic MI block
03 FILLER PIC X(1556) * Handler data area
03 MISPTR PIC PTR * Pointer to stack item
03 FILLER PIC X(11)
03 MIS32 PIC X(32) * Saved application data
```

The default information may be retrieved by the menu driver after
the MI block has been located by the first call to the menu
handler. If no default information has been saved then MIS32
will contain 32 bytes of spaces.

9.2.6 Use of Security Sequence Numbers
++++++++++++++++++++++++++++++++++++++

The security sequence numbers provide a method of tying the menu
file to a particular set of data files, to prevent someone simply
copying a new menu file onto the data unit and avoiding any
security which may have been set up.

The menu file has a two-byte security value embedded within it,
which is changed whenever amendments are made to the file. If
you establish MISECR and MISECW, then these routines will be
called when the menu file is opened by the menu handler (which
happens whenever the handler is invoked by the driver to display
the menu). The MISECR routine will be called first, to return
the security value, and the MISECW routine will be called
afterwards, to write the new security value away, if the menu
file has been amended. (The very first time you invoke a menu
you will not know what the correct security value to return is.

Consequently a value of zero is always regarded as valid by the menu handler.)

The simplest action to take with the security value is to write it away to a system header record, but this is not at all secure. It is much better to take a little trouble to disguise the security value, making it more difficult to tamper with.

We would recommend that you use some important but infrequently modified data field as an encryption key to 'scramble' the security value, and save the scrambled value on one of the application data files. Your method of scrambling the information must be such that you can reconstitute the security value from the scrambled value and the encryption key. Obviously you will need to create a new scrambled value if the value of the encryption key changes.

The general form of the security routines is illustrated below (MISECR points at UB-READ and MISECW at UC-WRITE):-

```
LINKAGE SECTION
77 L-SEC PIC 9(4) COMP * security value
PROCEDURE DIVISION
*
ENTRY UB-READ USING L-SEC
* retrieve scrambled security value from files and
* unscramble it (value must be zero very first time)
* place it in L-SEC
EXIT
*
ENTRY UC-WRITE USING L-SEC
* calculate new scrambled value of security field
* and write it to the data file
EXIT
*
SECTION UPDATE-ENCRYPTION-KEY
* retrieve scrambled security value from files and
* unscramble it
* alter encryption key value
* calculate new scrambled value of security field
* and write it to the data file
EXIT
```

A number of methods of actually encoding the security value are possible - the simplest would be to add the encryption key to it to produce the encoded value, but a more complicated method would clearly be more secure.

9.2.7 Using a Special Validation Routine
++

A special validation routine is indicated by MIVALI. If one is established it will be called by the menu handler immediately after the operator has selected a function, or keyed a program or command name. The validation routine is passed no parameters. The response just keyed by the operator (or the line number of the selected function if using SAA-style menus) is placed into MIPROG in the MI block before the validation routine is called.

The two most obvious uses for a validation routine are to restrict the commands which may be run from a menu (only certain responses are permitted), or to accept a concealed password which will release certain application functions.

The validation routine should exit normally if the menu handler is to proceed with the selected function (eg to load the selected command program). It should signal exception 1 (via an EXIT WITH 1 statement) if the selected function is not to be actioned, and the menu handler will re-prompt the operator (you would always do this after a concealed password prompt).

9.3 Amending menus under program control using MHRW$
+++ ++

The MHRW$ routine is used by an application to read and write menu entries. You would normally use this to build a menu entry for display where the contents of the menu are determined at run-time, or by system configuration (first reading the template, then updating appropriate fields and writing the amended version back to the menu file).

The routine is called as follows:-

CALL MHRW$ USING mc me
/////

where mc identifies a menu control block, and me is an area
// //
suitable for reading a single menu definition.

9.3.1 Menu control block
+++++++++++++++++++++++

The menu control block, MC, is laid out as follows:-

```
01 MC
03 MCMENU PIC 9(4) COMP * Menu number
03 MCPMST PIC PTR * Pointer to menu stack
03 MCOPCD PIC 9(4) COMP * Opcode
* 1 = read and lock
* 2 = write and unlock
* 3 = unlock
```

MCMENU must be set to the number of the menu to be read before MHRW$ is called to read a menu, and should not be changed before the following write or unlock is issued. MCPMST is a pointer to the menu stack data. This should either be established from the value placed in MISPTR by the menu handler (see section 9.2.5), or as the location of stack item "MENURETN" as determined by a call to ENTRY$. MISPTR can only be determined by the menu handler after invoking the menu handler from the menu driver. Otherwise the ENTRY$ routine must be used to find the stack item MENURETN. MCOPCD indicates the operation to be performed.

MHRW$ should be used to read a menu first (this will also lock the file to prevent other users from updating the file). The returned data should be amended as appropriate, and then written back using an opcode of 2. If no amendment of the menu is required, then the file should be unlocked using an opcode of 3, to permit other users to amend the file.

9.3.2 Menu definition block
++++++++++++++++++++++++++

The menu definition block, ME, is laid out as follows:-

```
01 ME
* Menu Header Area
03 FILLER PIC X(4)
03 MEDEP PIC 9(2) COMP * Menu depth
03 FILLER PIC X(3) * Reserved
03 MEDLAY PIC 9(3,1) COMP * Logoff delay
03 MELOGF PIC 9 COMP * +ve = call LOGOF$
03 MESCMD PIC 9 COMP * +ve = allow commands
* to be run from menu
03 FILLER PIC X * Reserved
03 MEEVLG PIC 9 COMP * 1 = event logging on
03 MECR PIC 9 COMP * 1 = last line is <CR>
03 METIME PIC 9 COMP * 1 = refresh time
03 FILLER PIC X(38) * Reserved for expansion
```

```
* Menu Lines Area
03 MELIN1 OCCURS 18 * Line display data
05 MENARR PIC X(30) * Narrative
05 MELTTR * Letter selection code
07 MELTR1 PIC 9(2) COMP * Highlight chr 1..30
07 MELTR2 PIC 9(2) COMP * Highlight chr 1..30
07 FILLER PIC X(2) * space for 4 chars
05 MEAUTH PIC X * Authority level
05 MEPAS8 PIC X(8) * Password
05 MEFUNC PIC X * Line type code
05 FILLER PIC X(9) * For expansion
03 FILLER PIC X(16)
03 MELIN2 OCCURS 18 * Line function data
05 MEORIG PIC S9(4) COMP * 0 = secure line
05 FILLER PIC X * Reserved
05 METLEN PIC 9(2) COMP * Length of type-ahead
05 METYPE PIC X(100) * Type-ahead
05 MEPROG PIC X(8) * Program name or menu
05 MELIB * Library name
07 MEMUN PIC X(3) * Menu file unit
07 FILLER PIC X(5)
05 MENXTM PIC 9(4) COMP * Next menu number
05 MENOTE PIC X(8) * Note pad name
05 MEAP6B * 6 bytes application specific data
07 MECDE1 PIC S9(2) COMP
07 MECDE2 PIC S9(2) COMP
07 MEOPT1 PIC X(2)
07 MEOPT2 PIC X(2)
05 MEHFIL PIC X(8) * Help file
05 MEHUN PIC X(3) * Help file unit
05 MEHBOOK PIC X(2) * Help file book
05 MEASSG PIC X(48) * 8 assignments
05 MEDB PIC X(36) * 4 SpeedBase databases
05 FILLER PIC X(17) * Reserved for expansion
```

The fields in the menu header area of the record will almost certainly not need to be changed, with the exception of MEDEP which must be set to the number of lines in the menu.

Of the line display fields, MELTR1 and MELTR2 should be set to the character number within MENARR which is to be highlit for selection, or zero if no such character exists. MEFUNC is given a value depending on the purpose of the menu line as follows:-

A	application item
B	return to the System Manager
F	load SpeedBase frame
M	change to new menu file
N	move to another menu in current file
R	return to previous menu (exit if main menu)
S	stand-alone program or command

Other fields within the line display area are essentially obvious
in their function.

The usage of some fields in the line function table varies depending on the value of MEFUNC. MEPROG is usually the program name, but serves as the menu file name when MEFUNC is "M". Similarly MELIB is used as the menu file unit (MEMUN). MENXTM is only used when MEFUNC is "N", and indicates the number of the next menu to be displayed.

The MEASSG area is a set of 8 unit assignments in the arrangement:-

```
07 MEAUID PIC X(3) * Symbolic unit
07 MEAUAD PIC X(3) * Unit address
```

Each symbolic unit must have an equivalent unit address to which it will be assigned, or a pre-existing symbolic unit to whose address it will be assigned. Symbolic units of spaces are ignored.

The MEDB area is a set of information for up to 4 SpeedBase databases which are to opened before a frame is loaded (they are only of interest if MEFUNC is "F"). It is organised as:-

```
07 MEDNAM PIC X(5) * Database name
07 MEDUID PIC X(3) * unit
07 MEDFLG PIC 9 COMP * open flag
```

MEDNAM is the five-character name (with the starting "DB" removed) of the SpeedBase database to be opened. MEDUID is the unit on which it resides, and MEDFLG is a flag value, set to zero for shared access to the file and to 1 for exclusive access.

The field MEORIG is used to identify secure lines, which should not be altered, moved or deleted by the program using MHRW$. Such lines may have associated access rights or system access codes, or be vital to the correct functioning of the application. Only lines with MEORIG set non-zero should be altered by the program, and lines it creates should have MEORIG set non-zero (usually set to -1).

9.3.3 Exception Conditions

++++++++++++++++++++++++++

Exception condition 1 is signalled if an irrecoverable I/O error arises when processing the menu file, or if the menu file cannot be found.

Exception condition 2 is signalled if the menu file is locked by another user when MHRW$ is called to read and lock a menu.

9.3.4 Programming Notes

+++++++++++++++++++++++

You cannot create a new menu with MHRW$, only amend an existing one, so the template menu should be created using the menu maintenance system in the usual way. You also cannot create a new menu line in a menu but can only amend existing lines.

No validation on the fields in the ME block is performed by MHRW$ or the menu handler, so you must exercise extreme care when updating or adding values.

If you move a line in the menu, this does not move any access rights or system access controls associated with the line. You should therefore not set up access rights or system access controls on any lines in the menu which might need to be moved, but perform such validation as is required in the menu driver once the line has been selected.

9.4 Checking menu authorization with MHPAS$

+++ ++

The MHPAS$ routine is used to perform authorization and access rights checking equivalent to that performed by the menu handling system for a particular menu entry. You would use it when there were two or more alternative routes into a particular function, one via the menu and others via separate routes, such as options from baseline prompts.

9.4.1 Invocation

++++++++++++++++

To check the authorization of the current operator to perform the function in question you code:-

```
CALL MHPAS$ USING ma
//
```

where ma is a menu authorization control block in the following
//
format:-

```
01 MA
03 MAMENU PIC 9(4) COMP * Menu number
03 MALINE PIC 9(4) COMP * Line number
03 MAPASS PIC X(8) * Password
03 MAPMST PIC PTR * Pointer to menu stack
```

You must establish the correct menu number and line number in MAMENU and MALINE before MHPAS$ is called. Initially MAPASS should be blank, as MHPAS$ will indicate whether a password is required when you call it. MAPMST is a pointer to the menu stack data. This should either be established from the value placed in MISPTR by the menu handler (see section 9.2.5), or as the location of stack item "MENURETN" as determined by a call to ENTRY$.

9.4.2 Exception conditions

+++++++++++++++++++++++++

Exception condition 1 indicates that the password is not correct (and that a password is required to gain access to the identified function).

Exception condition 2 indicates that access to the indicated function is denied, and that no password can be keyed which will grant access (the line would be 'starred out' of the menu for this operator).

9.4.3 Programming Notes

++++++++++++++++++++++

The menu system remembers if an operator has been granted an access right during this run of the application so that spurious password prompts may be avoided, and MHPAS$ integrates with this handling.

The way we would recommend you to use MHPAS$ is as follows:-

Call MHPAS$ with a blank password, with the appropriate menu
and line number identified. If no exception is returned
then the operator may safely be granted access to the
function;

If exception condition 2 is signalled, the operator can never gain access to the function (it is 'starred out'). Produce an explanatory message, and refuse the operator access to the function. (Sophisticated applications may wish to perform this check before giving a list of options, removing options which the operator cannot gain access to from the list);

If exception condition 1 is signalled, the (blank) password is incorrect. Prompt the operator for a password and call MHPAS$ again. Either access will now be granted, or the password will still be wrong (exception condition 2 cannot arise at this stage). If the password is wrong, then re-prompt the operator, possibly limiting the number of attempts he may make.

Appendix A - An Example Order Entry Application

This appendix explains how to run the example order entry program, ORDENT, which was developed using Screen Formatting. The stages involved in the design of the program are described and then, as an exercise, you use $FORM to modify the principal data entry screen defined by the ORDATA map. You then relink a new version of ORDENT and rerun the enhanced program.

A number of listings are provided following this description:-

Source listing of S.ORCOP, the A-8
copy library containing OR and WK.

Compilation listing L.ORDENT, for the A-8 to A-16
order entry main program.

Compilation listing L.ORVAL, for the A-17 to A-19
validation routine used in conjunction
with the ORCUST map.

Format listing L.ORCUST, for map ORCUST. A-20 to A-22

Format listing L.ORDATA, for map ORDATA. A-23 to A-25

Map listing M.ORDENT, containing the A-26
link map of the ORDENT program

The first page of the data entry log A-27
produced by ORDENT using MAPOUT...PRINT.

The example has been designed to run on terminals with a formatted area (excluding the base line) of at least 23 lines by 79 characters.

Running the ORDENT Program
++++++++++++++++++++++++++
You must first linkage edit C.ORDENT, C.ORCUST, C.ORDATA and C.ORVAL to create the order entry program ORDENT. You can then run the program to create an ORDER file of up to 100 orders for the customers whose account numbers, names and addresses, are supplied on the CUSTOMER file. As each order is completed details are written to the LOGFILE by outputting an image of the ORDATA screen using MAPOUT...PRINT. Before running ORDENT you should assign $PR to the unit on which you wish the LOGFILE to be developed, and $P to the volume containing the other files used by the example application.

Once ORDENT receives control the ORCUST screen is displayed, asking you for a unique 4 digit order number, and an 8 digit account number. The ORVAL validation routine checks that the order number has not been used previously, and that the account number is a multiple of 11. In addition, ORDENT itself tests that the account number is actually on the CUSTOMER file, using a MAPIN...FIELD to obtain corrected input if this is not the case.

Note that, in the interests of simplicity, the customer file only contains records for the 3 accounts 11111111, 22222222, 33333333.

Once you have specified a valid order number and account number
the ORDATA screen appears enabling you to change the despatch
address, or leave it the same by keying <CTRL A> or <CR>. You
------ --
can then enter a product description, quantity and unit price for
up to 9 order lines. Once a line is keyed, its value and the
running total are calculated and displayed and you are asked for
the next line. If there are no more lines for this order, simply
key <CR>. (Note that if you make a serious error during data
--
entry you can hit <ESCAPE> to obtain a fresh ORDATA screen and
------
start again from the beginning of the order.)

Following your keying of <CR> or the 9th order line, a confirm-
--
ation prompt appears on the base line. If it is incorrect, key N
-
or <CR> to cause the last field to be reinput. You can then use
--
BACKTAB (<LINE-FEED>) to go back to the incorrect field, and
----
change it. When you have made all the changes necessary, key
<CTRL A> to return to the confirm prompt.

When all is well, key Y. This will cause an image of the ORDATA
-
screen to be written to the LOGFILE and a new ORCUST screen to
appear, requesting details for the next order. If there are no
more orders, hitting <ESCAPE> will terminate the program.

Design Notes
++++++++++++
The ORDENT program was designed and coded by the author in a
little over 3 hours, although some minor modifications were made
once the program was operational.

The first stage was to consider the data requirements of the
program, and define the record format of the ORDER and CUSTOMER
files. In the interests of simplicity a common format, described
by the OR copy book, was adopted: although the records on both
files are the same, all order fields on the CUSTOMER file are set
to binary zeros. All the other fields required on the screens,
which were not to be part of the order record itself, were
collected into a copy book named WK. The printout of the copy
library S.ORCOP shows how these books are defined.

To help finalise the data requirements, and eventually code the
program, a sketch of the two maps involved was made on a large

piece of squared paper. A photo-reduced, slightly tidier, version of the original appears in Figure A1. It is not necessary to space out the screen area very precisely, since this can easily be adjusted at the terminal using $FORM. What is important is to identify the fields involved, and their attributes. In the figure each field is described by a shorthand of the form:-

name[(occurrence)] - attribute [,attribute...]
//

e.g:-

ORDATA(2) - U

or:-

ORVALUE - O, F2, B

or:-

WKNUM - I, N, V2

The (occurrence) is only used in defining multi-valued fields
which are not part of a record. The attribute list is
unnecessary for the inbuilt $$DATE and $$RECNO, but is present
for all the other fields. It starts with the field type
(O=output, I=input, U=update) and then uses the attribute letters
you will need to key in response to $FORM's ATTRIBUTES: prompt
(Table 3.4.2A). In addition, Fnn is used to represent field
//
number nn, and Vnn, validation code nn. To summarise:-
// // //

B = Blank when zero.

D = Date.

Fnn = Field number nn.
// //

I = Input field, at start of list, or invisible field,
elsewhere.

N = Number filled.

O = Output field.

U = Update field.

Vnn = Variant number nn.
// //

VCnn = Validation code nn.
// //

? = Defaulted.

The sketch of the maps was sufficient to allow the ORDENT program
to be written and during the coding the sketch was updated in
minor ways. For example, some fields that had not previously
been numbered were numbered, so that FIELD operations which were
not previously anticipated could be used, and an extra validation
code was added. When the program was complete $FORM was run to
create precise hard copy documentation: format listings L.ORCUST
and L.ORDATA, included below.

Initially ORDENT was tested without the validation routine ORVALR
by linking C.ORDENT, C.ORCUST and C.ORDATA together with the
system library and ignoring the warning message which identified
ORVALR as a missing global symbol. Once the program was
functioning correctly on clean data the validation routine was
coded and linked in. We were then able to check that the ORCUST
screen would not accept an order number that had been used

previously, or an account number which was not a multiple of 11. At this stage the program seemed to be functionally correct. The last page of this appendix shows the beginning of the LOGFILE produced during a typical run.

Figure A1 - Sketch of the ORCUST and ORDATA maps

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Amending the ORDATA Screen
++++++++++++++++++++++++++

Unfortunately, there is a fairly obvious failing in the program as far as the user is concerned, inasmuch as the ORDATA screen, and in consequence the LOGFILE, does not record the order number - certain to be a requirement in any practical application. You can correct this error by using $FORM to amend C.ORDATA. You should extend the box containing the account number and add the words ORDER NO. inside the box. (The field name of the PIC 9(4) COMP order number is ORNUM.)

Once you have produced a new C.ORDATA, you should relink C.ORDENT, C.ORCUST, C.ORDATA and C.ORVAL to create an enhanced order entry program named NEWENT. When you run this program the order number should be correctly displayed on each ORDATA screen, and on the log file.

There are numerous other ways in which you can change the maps without having to alter C.ORDENT in any way - we deliberately do not provide you with its source file. For example, the program will support up to 15 order lines so you can rearrange the text and redefine the fields to cram in more records if you have room. You could decide to make the customer name an update field, so that it can be changed if necessary. It is currently an output field which cannot be modified. You could add today's date to the ORDATA screen and more explanatory information to the ORCUST screen. Make a number of such changes until you are thoroughly familiar with using $FORM and screen formatting.

Source listing, S.ORCOP - Page 1

```
+++++++++++++++++++++++++++++++++
```

Compilation listing, L.ORDENT - Page 1

++++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORDENT - Page 2

++++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORDENT - Page 3
++++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORDENT - Page 4

++

Compilation listing, L.ORDENT - Page 5
++++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORDENT - Page 6

++++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORDENT - Page 7
++++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORDENT - Page 8
++

Compilation listing, L.ORVAL - Page 1

++++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORVAL - Page 2
++++++++++++++++++++++++++++++++++++

Compilation listing, L.ORVAL - Page 3

+++++++++++++++++++++++++++++++++++++

Format listing, L.ORCUST - Page 1

```
++++++++++++++++++++++++++++++++
```

Format listing, L.ORCUST - Page 2

```
++++++++++++++++++++++++++++++++++
```

Format listing, L.ORCUST - Page 3

```
++++++++++++++++++++++++++++++++
```

Format listing, L.ORDATA - Page 1
++++++++++++++++++++++++++++++++

Format listing, L.ORDATA - Page 2

```
++++++++++++++++++++++++++++++++
```

Format listing, L.ORDATA - Page 3

```
++++++++++++++++++++++++++++++++++
```

Map listing, M.ORDENT - Page 1

Data Entry Logfile from ORDENT - Page 1

Appendix B - Storage Estimates and Included Routines

This appendix contains information to help you make sizing calculations when planning an application. Table B shows the size of the mapping routine, certain system routines it references and the other system subroutines described in this manual. If your application is coded as an overlay structure using dependent programs, as described in section 6.2 of the Global Cobol User Manual, then rather than including all the mapping software in each overlay, you will probably want to make it part of the root program. This will reduce the size of the overlays, decreasing the amount of file space they occupy and hence the time taken to load them. If the root does not contain any MD statements, you can force the inclusion of the mapping routine by specifying its entry point as a global in the root, using the statement:-

GLOBAL SM$100

Size of Each Map Definition
++++++++++++++++++++++++++
Each map definition supplied in working storage occupies 24 bytes. Each MD coded in the linkage section requires only 2 bytes of the data division, but a MAPOUT or MAPIN statement referring to a linkage section MD requires an extra 4 bytes compared with the same statement using a working storage map definition.

Size of Each Map
++++++++++++++++
The number of bytes of memory occupied by each map is given by the formula:-

SIZE = 20(I + 2) + T
/ /

where I is the number of items the map contains, and T is the
/ /
number of text characters. (The size is printed on the format listing when the map is created.)

To calculate the number of items, I, consider each field and text
/
string the map contains to be an item, but only count items belonging to records once. Each text string occupies as many characters as you typed when defining it.

The quantity T in the formula is the number of characters
/
contained in text items, counting items within records only once.

The ORDATA map used as an example throughout this document, which might be considered to be a typical average, occupies 512 bytes.

Size of Each Map Processing Statement
++++++++++++++++++++++++++++++++++++++
The map processing statements only establish a linkage to the mapping routine, and generate between 12 and 22 bytes of code:-

* Count 12 bytes for each MAPOUT, MAPCLEAR or MAPIN;

* Add 4 bytes if the MD involved is coded in the linkage section;

* Add 4 bytes if the statement uses a field number or print file FD operand, and another 2 bytes if this operand is defined in the linkage section.

ROUTINE	PROGRAM NAME(S)	SIZE (K)
Mapping Routine	SM$A, GB$A, GE$A & GC$A	6.7
ACCE$	QF$A, QP$A, QS$A, GE$A	2.0
	& OS$A	
BASEL$ or KCR$	EE$A & BZ$A	0.9
BASEX$	EQ$A & BZ$A	1.4
BOX$	CB$A & EC$A	1.3
CEOL$ or CEOS$	GA$A	0.4
paged		
CHAR$, CHARX$, EOFCH$	GF$A	0.6
or ECHO$		
paged		
CHECK$ or CHKPR$	GE$A	0.3
paged		
CLEAR$ or CLR-N$	GA$A	0.4
paged		
COLOR$ or SCOLR$	GG$A	0.2
paged	GG$Z	
DBUF$	CH$A	0.1
DISP$	QG$A	0.1
DSYSR$ or ESYSR$	CF$A	0.1
FMODE#		
KEYS$	CJ$A	0.4
LINE$	IW$A & OS$A	0.4
MENU$	IP$A, GG$A, IW$A, OS$A,	2.6
	GA$A, GC$A, QL$A & EA$A	

```
| | | |
---------------------------------------------------------------
```

Table B - Included Routines

```
++++++++++++++++++++++++++++
```

ROUTINE	PROGRAM NAME(S)	SIZE (K)
MHPAS$	OO$A & OZ$A	4.3
MHRW$	ON$A & ER$A	0.9
MSG$	BO$A	0.1
NAR$	BW$A	0.4
PASS$	CD$A	0.2
POSI$	QD$A	0.2
RDSCR$, GTSCR$ & PTSCR$	BR$A	0.3
SCRN$	IH$A, EC$A & BJ$A	0.8
TEST$	II$A	0.1
TXEDT$	BX$B, QF$A, QD$A, GF$A, QP$A, QS$A, OS$A & GE$A	11.2
TXSML$	BX$C, QD$A & GA$A	5.8
TXVAL$	BY$A	0.2
TYP$ or TYPF$	BQ$A	0.3
VIDEO$	CC$A	0.6
VIEW$	IY$A, BC$A, EC$A & BZ$A	1.8
WIDE$	BW$A	0.4
WINDO$/WRES$	SN$A, EC$A, BR$A & GA$A	

Table B - Included Routines (cont.)

Appendix C - The Structure of the Screen Formatting System

You may find Figure C and the explanation below helpful in understanding how the various components of Screen Formatting interrelate and how the map processing statements, MAPOUT, MAPCLEAR and MAPIN, operate.

The components invoked in any application using Screen Formatting are: a main program containing the map definition, the mapping
+++++++++++++ +++++++
routines responsible for the run-time handling of MAPOUT and
++++++++
MAPIN requests; the map itself, which is simply a collection of
+++
tables; up to five data areas containing the input, output and
+++++
update fields; and the validation routine which is optional.
++++++++++++++++++
Pointers to the map and data areas must be established in the map definition prior to any map processing statement being executed. The validation routine, if present, is addressed by a pointer located within the map itself. The MD also contains a pointer to the mapping routine.

Any mapping statement transfers control to the mapping routine using a CALL on the pointer located in the MD. Two or three parameters are passed:-

* the MD itself;

* an operation code indicating which type of MAPOUT, MAPCLEAR or MAPIN is required;

* The optional, the field number or printer FD.

All the data required for the mapping operation is thus readily available, since the MD contains the record number, variant number, and pointers to other necessary information.

For example, consider a MAPIN...ALL operation. By scanning the field attribute tables within the map (accessed via an MD pointer) and taking into account the record number and variant number, the mapping routine determines the screen location and other attributes of each field to be accepted. Suppose the current field was assigned a validation code by $FORM, and the internal pointer within the map shows that a validation routine has been supplied. Then the mapping routine will first of all establish the tentative new value of the field and create an FV block to define it in working storage. Control will then pass to

the validation routine which will perform its own checking using
the new value and the FV. If all is well this routine exits,
signalling normal completion, and mapping regains control. The
validated new value is displayed, and then moved to one of the
five data areas addressed by the MD. The number of the area to
use, and the correct displacement within it, are, of course,
field attributes established in the map.

Once the mapping routine has processed every input and update
field affected by the MAPIN...ALL request it will return control
to the statement following the CALL expanded by the compiler.
However, in the event that $$ESC had been set to 2 and the
operator keyed <ESCAPE>, the routine would terminate the MAPOUT
------
prematurely and indicate that this had happened by signalling
exception condition 2.

```
| |
--------------------------------
| |
| PROGRAM ORDENT * MAIN PROGRAM |
 ++++++++++++
| DATA DIVISION |
| | | |
 -----------------------
| | | | |
 ---
| | |MD M-CUST MAP "ORCUST"| |
| | |RECORD R-CUST | | |
 -------------------------------------
| | |AREAS "A1"..."A5" | | | | |
 -----------------------------------
| | | | | | | | |
 ----
| | | | || | | PROGRAM ORCUST * MAP | |
 ------------------------ +++
| | || | | DATA DIVISION | |
| | || | | | |
| | | | || | | Contains the validation | | |
 ------------------------ -----
| | | | | | | routine pointer and the | | |
| | | Data area A1 | | | | text of the map together | | |
| | | | | | | with tables of field | | |
 -----------------------
| | | | | attributes and run-time | | |
| | . | | | options. There is no | | |
| | . | | | procedure division. | | |
| | . | | | | | |
| | . | | | ENDPROG | | |
| | | | | | | | | | |
 ------------------------- ----------------------------
| | | | | | |
| | | Data area A5 | | | |
| | | | | | | | |
 ----------------------- ----------------------------
| | | | | | |
| | | | PROGRAM SM$A | | |
| | PROCEDURE DIVISION | | * MAPPING ROUTINE | | |
 +++++++++++++++
| | SECTION MAIN | | DATA DIVISION | | |
| | | | | | 01 FV etc. | | |
 ------------------------
| | | | | . | | |
| | MAPIN M-CUST etc. | | | PROCEDURE DIVISION | | |
| | | | | ENTRY SM$100 USING MD OP NO| | |
 ----------------------- ---
| | | | |
| | | The mapping routine uses | |
| | | the MD to obtain access to | |
| | | all parameters and calls | |
| | | the validation routine as | |
| | | appropriate | |
| | | | |
| ENDPROG | | ENDPROG | |
| | | | |
------------------------------- -----------------------------
 |
 | | |
 -----------------------------
 | | |
 | PROGRAM ORVAL * VALIDATION | |
 ++++++++++
 | DATA DIVISION * ROUTINE | |
 +++++++
 | . | |
------> Data references | LINKAGE SECTION | |
 | 01 FV etc | |
------ Flow of control | . | |
 | PROCEDURE DIVISION | |
 | ENTRY ORVALR USING FV | |
 -----
 | |
 | User supplied tests |
 | |
 | ENDPROG |
 | |
 -----------------------------
```

Figure C - Structure of an Application using Screen Formatting

Appendix D - Global Screen Display Standards

This appendix summarises the general guidelines for handling the layout of screen displays and accepts for use by standard Global applications software. Note that these are the standards as of July 1988 and are subject to future revisions.

Standard use of colours

All standard software will only make use of colour by way of the eight standard colour combinations defined within the Operating system customization. Special purpose software (such as Global Graphics) may make particular and specific use of colour in appropriate circumstances (such as chart display).

The use of the standard colour combinations (SCC) is shown in table D.1.

SCC	Standard Use
1	Scrolled working (e.g. operating system command), baseline prompts, audit areas
2	Lines and Boxes
3	Text (i.e. non-variable information, all headings and titles)
4	Variable information
5	Not used
6	Help information
7	Highlit fields (i.e. fields to which it is desired to draw attention)
8	Input field (i.e. the field currently being

```
+
| | input) |
| | |
-----------------------------------------------------------------
```

Table D.1 - Uses of standard colour combinations
++

The standard colour combinations should be set up so that combinations 1 to 4 have the same paper colour, with the contrast between ink and paper increasing from 2 to 4, and the combination 1 contrast similar to that of colour 3. Combinations 6, 7 and 8 should all have different paper colours (and also differ from that of combinations 1 to 4), and are intended to stand out from the rest of the screen without being garish or unreadable.

To this end the default colour combinations (as held within System Manager) are as shown in table D.2.

SCC	Default colours
1	Green on Black
2	Red on Black
3	Cyan on Black
4	White on Black
5	White on Green
6	Blue on Cyan
7	White on Blue
8	White on Red

Table D.2 - Default colour combinations
++++++++++++++++++++++++++++++++++++++

It should be noted that the standard colour definitions are essentially for use with screens which use (or have the general look of screens which use) Screen Formatting for their creation. In other circumstances the standard colour usages may be inappropriate.

When using the standard colour combinations the following points should be noted:-

* All title fields (even variables) and all menu lines should be in colour combination 4;

* When a number is to be displayed in a particular colour combination, any 'text' field strongly associated with the

number (such as a quantity type or qualifying symbol such as
% or $) should be displayed in the same colour combination;

* When information is displayed as a sentence, or series of
sentences, the whole sentence should be in the same colour
combination.

Standard menus
++++++++++++++
In general software looks more integrated if the various main
menus displayed during its use all share a common form. To this
end the MENU$ subroutine was enhanced as part of V6.0 to produce
a menu in the recommended format, as used by Business Software.
Programs displaying menus should either use MENU$, set up the
menu using screen formatting, or use an application menu driver
with $MH.

A standard menu is laid out as follows:-

* The top line of the screen has a suitable overall title, which can be up to 50 characters. This will be the system name ($$SNAM) if no more sensible option is available. In the Business software the company name from the data files being used is often a more sensible choice;

* The rightmost part of the line consists of the day name (e.g. Tuesday) followed by the short form of the system date (e.g. 12/07/88) and the system time in hours and minutes only (e.g. 11.17). The last character of the time should be at the extreme right-hand edge of the screen display, regardless of the current screen width;

* The second line is a full underline (using graphics characters if possible) of the screen;

* The third line is blank;

* The fourth line is centred, and is the package title in capitals (e.g. GLOBAL 2000 - SALES LEDGER);

* The fifth line is an underline (using graphics characters) the same width and position as the previous title;

* The sixth line is the menu title, centred and in capitals (e.g. MAIN MENU). A single extra space may be inserted into this line to preserve a balanced look if required;

* The seventh line is an underline of 33 characters (using graphics characters) centred on the screen, acting as a top for the following menu lines;

* The following consecutive lines are menu entries. Each consists of a function description of up to 30 characters, padded to the right with alternate periods and spaces (with the first period being notionally in the first character position in the line). The 31st character will always be a period, and the 32nd and 33rd characters will be the function number, with a leading space if necessary. The final line of the menu body will have the function description 'Exit', and will have the characters '<CR>' as the last 4 characters on the line (positions 30 to 33). All the lines are centred, and the only capital letters are the first word on every line, and the first character of any proper names;

* The line following the menu body is blank:

* The next line is the function selection prompt, consisting

of the description 'Please select a function' under the
first characters of each function line, and a ':' character
(for the prompt) at the 31st character position.

An example of a standard menu is shown in figure D.3.

A menu must not have more than 13 function lines (due to screen
size limitations), and should ideally not have more than 9.

Figure D.3 - Example standard menu

The prompt will be an ordinary colon prompt if using MENU$, or a
prompt in the input colour if using screen formatting.

Data entry screens
++++++++++++++++++
A data entry screen will have the top line in the same format as
a menu, which is to say a system title with date and time, and
the second line will be either a solid underline, or the top of a
screen surrounding box.

The body of the screen may be surrounded by a box and/or it may
be split into areas by use of boxes and lines. Care must be
taken not to obscure information by overusing boxes, but used
sparingly they help to show sensible separations of the data on
the screen. Where boxes or vertical lines are used a single
space gap should be allowed between the vertical line and any
adjacent text or data.

In the body of the screen descriptions of fields (or groups of
fields) should be in lower case with an upper case first letter.
Information provided by the program suite should be in upper case
where practical, but information derived from the operator (such
as name and address fields) will be in whatever case it was
originally typed.

If screen formatting has been used to lay out the screen (the
recommended practice) then the following features will be
available automatically - if screen formatting has not been used
a subset of these features should be provided as far as is
practical:-

* Keying <CR> after some characters causes the characters so
far typed to be the new value of the field;

* Keying <CR> without first keying any other characters causes
the currently displayed default to be accepted;
///////

* In both cases the input will pass to the next field;

* Keying <CTRL A> to terminate a field input will cause all subsequent fields to take their default values (if possible);

* Keying cursor up (or LINEFEED if cursor up is unsuitable) will cause the current input to be erased, and the original default to be restored. If keyed at the start of a field it will instead cause input to pass back to the preceding input field (if one is available).

Note that most of these facilities are provided at the most fundamental level in the operating system accept handling. Field editing should also be available on any prompt, but this is generally simply a case of linking the software with the most up-to-date system routines, and taking care with the default management.

Baseline prompts
++++++++++++++++
It is standard within Global Software to use baseline prompts to select processing options from a data screen. For example if a record is shown, and it is possible to amend the whole record, or certain selected groups of fields, to delete it, or to confirm that processing is complete, then these options would be selected by means of a baseline prompt.

Standard software will call the BASEL$ subroutine, which is documented in section 7.21 of this manual to perform its baseline prompts. If for some reason this is not possible then the general form of handling provided by BASEL$ should be imitated as closely as practical.

There are some guidelines for laying out a baseline prompt which are summarised below:-

* If the prompt includes an information request then this should be the first element of the prompt;

* Further options should appear in alphabetical order;

* Responses of <CR> and <ESC> should be at the end of the list (ESC to precede CR if both are present).

The following option letters are given standard meanings - if a baseline prompt needs to be able to select one of these options it should use the standard option name given. Wherever possible other options (with different meanings) which use the same letters should be avoided:-

| | |

```
| | |
| Option | Meaning |
++++++ +++++++
| | |
--------------------------------------------------------------
| | |
| ? - help | provide help information |
+
| | |
| A - Amend | permit change to all fields of the record |
+
| | |
| B - Back | go back one page (or one record) |
+
| | |
| C - Cancel | abandon changes (similar to <ESC> on input) |
+
| | |
| D - Delete | delete the record |
+
| | |
| F - First | go to first page (or record) |
+
| | |
| I - Insert | insert a record |
+
| | |
| L - Last | go to last page (or record) |
+
| | |
| N - Next | go forward to next page (or record) |
+
| | |
| P - Print | print the record/set of records/whole list |
+
| | |
| S - Search | enter search (e.g. alpha enquiry, account name)|
+
| | |
| U - Update | change one field or group of fields |
+
| | |
| <CR> | confirm, accept the updated record |
++++
| | |
| <ESC> | exit |
+++++
| | |
--------------------------------------------------------------
```

Figure D.4 - Standard baseline options

Note that in many situations <CR> to confirm and <ESC> to exit will have the same effect.

Lengthy processes
++++++++++++++++
During a lengthy process it is important to give the operator some idea that the program is still functioning. This may not always be possible, but where it is the following points are of interest:-

* Information to be displayed should be sufficiently frequent that it is obvious that something is going on, but not so frequent that the whole process is unnecessarily slowed by the displaying.

* Messages should be displayed no more often than twice a second. It is often convenient to display some information from records being processed, and in this case it may be appropriate to display only from some of the records (every fifth, for instance) to avoid too many displays.

* Where the messages will be relatively infrequent, describing individual processes which make up a very lengthy process, they should be displayed in the format:-

Infrequent message . . .

* With the trailing dots serving to suggest that it may be a little while before a further message is displayed. This kind of message is particularly appropriate during file reorganise and restore operations.

* The messages displayed should either be displayed in scroll mode, so that a record exists on the screen, or they should be successively displayed on some line in the lower half of the screen, with each message overwriting the preceding one. It is dangerous to use the baseline for this as successive displays will cause a comma prompt to appear.

* At the end of a long process a simple baseline message should appear of the form:-

process complete - Key <CR> to continue:
///////////////

Error reporting
+++++++++++++++
When an error arises from validation of an input data field the error message should appear on the baseline, and the operator should then be reprompted in the body of the screen.

If the error is obvious (e.g. the operator has failed to key one of a small number of legal replies) a simple BELL verb will suffice as the error message. In cases where more complex validation has revealed an error (e.g. the operator has keyed an account number which is not on file) then a message explaining the error is required (e.g. in the previous case 'ACCOUNT xxxxxxx NOT ON FILE').

There will also be cases where an error arises which is not directly associated with a prompt. In such a case a message MUST be displayed on the baseline describing the nature of the error, and providing the operator with an opportunity to select a recovery action (if possible). Even when no recovery is possible a 'Key <CR>:' prompt must appear to give the operator the opportunity to read the message. Such a message is produce by calling the KCR$ routine, which is included within BASEL$.

Typically with operational errors the only sensible options will be to retry the operation (key <CR>) if this is possible, or to abandon processing (key <ESC>).

It is important that all error prompts which appear from data input validation execute a BELL verb to cancel type-ahead and terminate job management. Error prompts arising from operational errors should ideally bypass job management and cancel type-ahead if they may be retried. If they cannot be retried, and in any case if the operator elects to abandon the operation, then a BELL verb should be executed to cancel job management.

PG D-4

```
 SOFTWARE V6.1 Monday 18/11/88 16.33
*****************************************************************************
 GLOBAL - SALES LEDGER
 **********************
 MAIN MENU
 ********************************
 Transaction processing. . . . . 1
 Allocation. . . . . . . . . . . 2
 End of day. . . . . . . . . . . 3
 End of period . . . . . . . . . 4
 Management reporting. . . . . . 5
 System maintenance. . . . . . . 6
 Customer maintenance. . . . . . 7
 Enquiries . . . . . . . . . . . 8
 Interface to Writer . . . . . . 9
 Restore data. . . . . . . . . .10
 Exit. . . . . . . . . . . . .<CR>

 Please select a function _
```

PAGE 2-13

```
 ACCOUNT POSTING TRANSACTION

ACCOUNT NUMBER :_
AMOUNT
TRANSACTION TYPE

DATE

TAX AMOUNT
TAX CODE
```

TOTAL

CONFIRM CORRECT PRIOR TO POSTING

PAGE 2-15

ACCOUNT POSTING TRANSACTION

ACCOUNT NUMBER :_
AMOUNT :1413.17
TRANSACTION TYPE :PL

DATE :22/08/88

TAX AMOUNT :211.97
TAX CODE :A

TOTAL :1625.04

CONFIRM CORRECT PRIOR TO POSTING :Y

ACCOUNT NUMBER 12345678 POSTED

```
-----------------------------------------------------------------------------------------------
| | | |
| $$COND | CONTROL FUNCTION | TYPICAL IMPLEMENTATION |
| | | |
-----------------------------------------------------------------------------------------------
| | | |
| 1 | PROGRAM FUNCTION 1 | |
| | | |
| 2 | PROGRAM FUNCTION 2 | The implementation of the five program functions is entirely |
| | | application-dependent and there is no attempt to define a |
| 3 | PROGRAM FUNCTION 3 | standard usage. |
| | | |
| 4 | PROGRAM FUNCTION 4 | |
| | | |
| 5 | PROGRAM FUNCTION 5 | |
| | | |
| | | |
```

6	RETURN	This is the normal field terminating character. In word
		processing applications it may be used to indicate that a
		forced end of line is required.
7	ESCAPE	When CHAR$ is used for console input normal ESCAPE key handling
		does not take place. The ESCAPE function should either use a
		STOP RUN to return to the monitor in the normal way or perform
		some application-dependent recovery process.
8	CLEAR	This is normally interpreted as a request to delete the current
		field, line or screen, depending on the current status of the
		the application.
9	CURSOR HOME	This usually causes the cursor to move to the start of the
		first input field, normally at or near the top left hand corner
		of the display. The contents of the screen remain unchanged.
10	PAGE	This is normally used during editing operations to indicate
		that a new display page is required from file.
11	CURSOR LEFT	These functions provide non-destructive cursor control. Left
		and right operations are normally confined to a single line
12	CURSOR RIGHT	and upward and downward movement is limited to the currently
		displayed page, avoiding the base line so that scrolling is not
13	CURSOR UP	inadvertently caused by cursor movement.
14	CURSOR DOWN	
15	TAB RIGHT	Tab operations are normally confined to a single line, the tab
		settings being application-dependent. The operations are
16	TAB LEFT	usually non destructive, with the possible exception of tab
		right in data entry applications.
17	ERASE	This function is normally used to delete the characters
		previously keyed, by echoing BS SP BS, for example. Normally
		only characters from the current field should be erased.
18	UNDEFINED	You should usually ignore undefined control functions by simply
		invoking CHAR$ immediately to obtain the next input.

(PG 7-18) TABLE 7.8.4 CONTROL FUNCTIONS AND THEIR TYPICAL IMPLEMENTATION