PART 4
TESTING & DIAGNOSTICS

Telecomputing Limited Oxford

PART 4 - TESTING AND DIAGNOSTIC FACILITIES

1 TESTING APPLICATION ROUTINES

Testing of Application Routines is provided for in two ways:

1. By use of a free-standing Module Tester.

2. By use of the System Tester, which is a special testing Version of TPS itself.

Section 1.1. describes the Module Tester. It is designed to be easy and flexible to use, and is primarily aimed at testing the logic of a single routine although, as described below, it can be used to test more complex groupings of Application Routines. It is, however, strictly a single threading operation; if the system is such that multi-threading aspects require testing (e.g. in contention for work file space, or in the use of common areas) this can only be achieved using the System Tester. The latter is described in Section 1.2.

1.1. THE MODULE TESTER

1.1.1. Description

The Module Tester consists of a Control Segment and one or more Application Routines consolidated together. The Control Segment is constructed using Macros as described in the System Generation Manual, Part 4. The Application Routines may be written in COBOL or PLAN. Once set up, the Tester and routine(s) under test are driven by the contents of an Input File. The Input File contains command and data records and may initially be contained in punched card or paper tape format. However, further records may be contained in simple subfiles of a composite disc file and these are accessed through commands contained in the original card or paper tape file. When all records in a particular subfile have been input then control returns to the next record contained in the original card or paper tape input file. Simple subfiles in a composite disc file may be created and edited using ICL program # XMED. The Module Tester will use either a card or paper tape input file according to the entry point used to start the program.

1.1.2. Sequence of Input

The format of records in the input file is described in Section 1.3 and a complete description of each command is contained in Section 1.4.

For the Module Tester, the sequence of records in the Input File is fairly unconstrained and the run may be constructed in any way appropriate to the test to be carried out. The only exceptions to this are that commands referring to formats held on a disc file and to subfilesof data held on a disc file must each be preceded by appropriate commands to identify the particular files.

The Module Tester does, however, validate parameters and, on detecting an error, outputs a suitable message and halts. The run may be resumed at entry point 22 to resume processing at the next command or at entry point 29 to resume processing at the next *NEW command.

A typical sequence of events might be as follows:-

*FORMAT, with following data records, to set up a simulated screen in store.

*INPUT, with following data records, to insert the variable data in the simulated screen.

*DISPLAY, to print the simulated screen to verify the accuracy of the test data.

*SEND, to transfer the input data to the message area, and activate the routine identified by the first three (or four) characters of the input data.

*DATA, with following data records to be set up in store by the Tester when control is returned from the routine under test. This might for instance simulate the effect of a disc read.

*DATA, with the following data record to set the reply word associated with this simulated disc transfer.

*CONTINUE, to re-enter the routine under test.

*OUTPUT, to print results - e.g. to print the contents of an expected disc output record on exit from the routine under test.

*CONTINUE, to re-enter the routine under test.

*OUTPUT, to print the final state of work areas etc.

*NEW, to reset the Control Block.

*REFORM, to reset the simulated screen.

*INPUT, to start another test sequence.

**** to signify end of run.

1.1.3. Testing Combinations of Application Routines

Although the Module Tester is primarily aimed at testing single routines it may easily be used to test a series of routines by inserting the appropriate *CALL commands into the input sequence. The effect of routines missing from a series can be simulated by appropriate *DATA commands.

1.1.4. Adding extra facilities

Extra facilities may easily be incorporated into the Module Tester in the form of routines which are "under test" as far as the Module Tester itself is concerned, and which are identified in the way described in section 1.5.2. The most likely use for such a routine would be to provide access to data files held on disc. As the Module Tester is a single threading program such routines may be simply written, to normal batch processing standards, and may, for instance, incorporate normal Housekeeping functions. If written in COBOL they are written as subroutines and must observe the rules associated with COBOL subroutines, in particular in connection with accessing files, and must incorporate the same Linkage Section as the standard ARs under test as they will be entered through the standard calling sequence.

1.2. THE SYSTEM TESTER

1.2.1. Description

Unlike the Module Tester, the System Tester incorporates most of the TPS routines which would be present in a "live" version of the program. The routines handling the communications equipment are replaced by routines which direct terminal output to line printers and accept input from card and/or paper tape readers.

The Tester program is generated in the same way as a "live" program, replacing the communications interfacing macros with special "tester interface" macros, as described in the System Generation Manual (Part 5).

The System Tester may be used to test complete Application Routine Trains or part of any train. Any Application Routine within a train may be missing from the program. Entry to any missing routine is "trapped" by the Tester and commands in the input test file may be used either to simulate the action of the routine or to terminate processing of the current transaction. The use of this facility is described more fully in Section 1.2.2.

Input data will be read from one or more card or paper tape test files according to parameters specified in the control routine. (Note that the number of test files handled concurrently will not exceed the number of Control Blocks).

Basic peripherals are defined in "pairs", each "pair" consisting of a printer and a card or paper tape reader. Each "pair" may be used to represent a different terminal; alternatively the terminal identifier may be changed by a command within the input test file.

The George 2 or 3 operating system facilities for using disc files may of course be adopted in order to facilitate the use of several "pairs".

A further printer may be specified to receive all diagnostic printer outprint requested in any input stream. The use of this facility under the System Tester is described more fully in Section 1.2.2.

1.2.2. Sequence of Input

The format of records in the input test files is described in Section 1.3 and a complete description of each command is contained in Section 1.4.

For the System Tester, the sequencing of Application Routines is controlled by TPS in the normal way, missing Routines being "trapped" by the Tester. For this reason commands within an input test file must appear in a logical sequence and commands which directly control the sequencing may not be used (i.e. *CALL, *CONTINUE, *RESUME).

To initiate processing of a transaction, an *INPUT command is used to enter data into a screen image and the command *SEND is used to simulate transmission of the screen. (A simulated screen image is held in core for every input test file).

Alternatively data may be entered directly into the Message Area using the command *DATA and then processing of the transaction is initiated using the command *ENTER. In this latter case the contents of the screen image should be considered to be indeterminate.

When the System Tester detects that control has been transferred to a missing Routine, it expects to read commands in the input test file which simulate the action of the missing Routine. The last of these commands must be either a *NEXT which transfers control to another Application Routine or an *END which terminates processing of the current transaction.

In the event of an error occurring in the logical sequence of an input test file, or in the event of any other type of error being detected, the System Tester will output a suitable message and ignore any further records in that input test file until a *NEW command is encountered.

Apart from the sequence checking involved with *INPUT *SEND *NEXT and *END cards, described above, the sequence of records in the input test file is unconstrained.

Most commands in an input test file take effect when they are encountered. The exceptions to this are the commands that define print requirements which are additional to the normal printing of simulated screens on a "terminal" printer.

A set of up to ten of these commands (*OUTPUT, *DISPLAY), will define the current print requirements for a particular input test file. The printing will be performed on exit from every Application Routine and will be output to the "dump" printer if one was specified to the set-up macro (£TTPA). If none was specified, the "terminal" printer will be used.

Each set of "print" commands will override any previous set and will be effective from the time that the next *SEND, *NEXT or *END command is processed.

The *MOP command will apply to all sets of "print" commands subsequently encountered in the same input test file.

Although print commands are available with the System Tester it is envisaged that their use will be the exception rather than the rule, since the object of the System Tester is to simulate a "live" situation as far as is possible. During the printing of this type of output, multi-threading will not take place.

A typical sequence of events might be as follows:

*INPUT with following data record to enter a message code requesting a screen format.

*SEND to pass the message to TPS. TPS will retrieve the required screen format, which the Tester routine will store in the simulated screen image and print.

*INPUT with following data records to insert variable data in the simulated screen image.

*SEND to pass the input data to TPS. TPS will locate the Application Routine Train in the normal way and pass control to the first Routine in the Train.

*DATA with following data records. When the System Tester detects that control has been transferred to a missing Application Routine it reads the next records from the input test file which simulate the action of the missing routine. These may for example simulate the effect of a disc read. They may also set parameters in the Control Block to represent a TPS request.

*NEXT to transfer control to another Application Routine in the Train. Any terminal output generated by an Application Routine will be directed by the System Tester to the "terminal" printer.

*TERMINAL to change the terminal identifier.

*FORMAT with following data records, to set up a new simulated screen. (For example, a screen format which would normally have been output at the end of a previous Application Routine Train, and which cannot be retrieved in response to inputting a message code).

```
*OUTPUT   )  To define areas which are to be printed
*OUTPUT   )  on exit from each Application
*DISPLAY  )  Routine.
```

*INPUT with following data records, to insert data in the simulated screen.

*SEND to pass the screen data to TPS. TPS will locate the Application Routine Train in the normal way and pass control to the first Routine in the Train. On exit from every Application Routine the areas specified in the *OUTPUT and *DISPLAY commands are printed.

*END to terminate the processing of the transaction when the Tester detects a missing Application Routine.

*REFORM to reset the simulated screen image as defined by the previous *FORMAT command.

etc.

**** indicates end of input test file.

The System is closed automatically when all input test files have been exhausted.

1.3. FORMAT OF INPUT RECORDS

The Input File which controls the running of a test program contains records which fall into two categories

Command Records: These direct the action of the master segment (i.e. the tester itself).

Data Records: These contain data to be set up in store for action by the routine(s) under test. They occur after the following command records, as described below.

*DATA *FORMAT *INPUT *PATCH

1.3.1. Format of Command Records

The format of each Command Record is described separately below. In general they are structured as follows:

*MNEMONIC▾Param, Param

* must always occur in Col. 1

MNEMONIC is one of the Command words described below, and follows directly after the * without a space. At least three letters of each Mnemonic are required; the whole word need not be used.

▾ A single space must follow the Command word

Param, Param. One or more parameters may follow the command, in which case they are separated by commas. A comma is not required after the last parameter.

1.3.2. Format of Data Records

Data records which follow the commands *DATA, *FORMAT and *INPUT are set up as character strings in the form in which they are to appear in store. There are no abbreviations nor conversion into binary. Binary values must be calculated and expressed in character form, unless they are set up by *ALTER command. Trailing spaces will be set into store from each card image unless prevented by a count parameter in a *DATA, or by use of the terminator in any card following *INPUT. The normal terminator is the character ← but this may be changed by the *VDU command.

Data records which follow the command *PATCH are in the format of PLAN instructions where the operation is represented by the normal PLAN mnemonic, the accumulator is a decimal integer and the operand takes any of the following forms:

A standard identifier (see 1.5.1.)

Any identifier generated by the user (see 1.5.1. and 1.5.2. and the command *DEFINE)

Any absolute program location, in decimal or octal form.

Any symbolic operand may be qualified by a displacement (+n).

A modifier field (M) may also be added in normal instruction format.

1.3.3. Command Functions

The functions of the Command records fall into six groups:

1. SET UP DATA IN STORE for use in the test:

*FORMAT	Create a screen image.
*INPUT	Insert data into "unprotected" fields.
*DATA	Move following data to a specified store area.
*REFORM	Reset screen image and cursor position.
*ALTER	Set a value in a single word.
*PATCH	Program patch.
*DEFINE	Rename a store area.
*VDU	Define terminator character for INPUT.
*ZEROISE	Zeroise a named store area.
*MOVE	Copy store area to store area.
*CURSOR	Set cursor position in simulated screen.
*RECALL	Get screen image from disc file.

Data contained in the records which follow these commands is set up in areas indicated by Identifiers quoted in the commands. The range of identifiers is described in section 1.5 below:

2. ACTIVATE THE ROUTINE under test:

*SEND	Copy 'unprotected' fields to message area; activate A.R.
*ENTER	Activate the routine identified by the screen identifier.
*CALL	Activate a nominated routine.
*CONTINUE	Return to the same routine
*RESUME	Branch to named program instruction.
*DAS	"DISPLAY" and "SEND"

The routine to be entered is indicated either by Message Type of the data to be processed (*SEND, *ENTER) or by an Identifier quoted in the command. The range of identifiers is described in section 1.5. below.

3. Define the PRINTING REQUIREMENTS:

*DISPLAY	Print the contents of the screen image.
*SIZE	Specify size of the screen image.
*OUTPUT	Print contents of named store area (compact).
*PRINT	Print contents of named store area (full).
*MOP	Defines that output is to a MOP terminal.
*LIST	List all input data.
*NOLIST	Suppress listing all input data.
*COMMENT	Comment line.

4. CONTROL THE RUNNING of the program:

*CWARN	Change the warning character for commands.
*HALT	Stop.
*NEW	Display a progress identifier on console log.
*NEXT	(In place of an absent A.R.) Go to next A.R.
*END	(In place of an absent A.R.) Terminate processing this message.
*TERMINAL	Change terminal identifier for test data.

5. Enable SCREEN FORMATS to be stored on a disc file:

*FILE	Name the file to hold screen images
*OPEN	Open the file
*INITILIASE	Open and clear the file.
*STORE	Copy screen image to the file.

6. Enable the INPUT FILE TO BE HELD ON DISC:

*IDF	Name a file to hold test data.
*ISF	Accept commands from named subfile.

1.4 COMMAND RECORD FORMATS AND FUNCTIONS

In this section, the individual Command records are described in alphabetical order. The complete list is:

Command.	System Tester	Module Tester	Command.	System Tester	Module Tester
*ALTER	S	M	*LIST		M
*CALL		M	*MOP	S	M
*COMMENT		M	*MOVE	S	M
*CONTINUE		M	*NEW	S	M
*CURSOR	S	M	*NEXT	S	
*CWARN	~~S~~	M	*NOLIST		M
*DAS		M	*OPEN		M
*DATA	S	M	*OUTPUT	S	M
*DEFINE	S	M	*PATCH	S	M
*DISPLAY	S	M	*PRINT		M
*END	S		*RECALL		M
*ENTER	S	M	*REFORM	S	M
*FILE		M	*RESUME		M
*FORMAT	S	M	*SEND	S	M
*HALT	S	M	*SIZE		M
*IDF		M	*STORE		M
*INITIALISE		M	*TERMINAL	S	
*INPUT	S	M	*VDU	S	M
*ISF		M	*ZEROISE	S	M

*ALTER

SYSTEM TESTER
MODULE TESTER

FORMAT *ALTER   A , B

PARAMETERS

A	a single word location identified by any of the methods described in section 1.5.
B	an integer in decimal or octal form.

FUNCTION

Set a given value into a single word location. It has the simple advantage of requiring only a single card, and is useful for setting such parameters as the Reply word or Re-entry point.

*CALL MODULE TESTER ONLY

FORMAT *CALL A

PARAMETERS

A a Mnemonic, set up as described in section 1.5., identifying the routine to be activated.

FUNCTION

Activate the nominated routine at either its first instruction or at a subsequent instruction if the Mnemonic name is defined in the format SEGNAME+n. This latter option must be exercised with caution. In a routine written in COBOL it will bypass the linkage address generation with unpredictable results.
In an A.R. written in PLAN it will bypass the restoration of accumulators. It should not be considered as a normal procedure.

*COMMENT MODULE TESTER ONLY

FORMAT *COMMENT A

PARAMETER

A the comment.

FUNCTION

This is used for providing comment lines within the data. Each line of comment must start with the *COM command.

*CONTINUE MODULE TESTER ONLY

FORMAT *CONTINUE

PARAMETERS

None

FUNCTION

Cause control to be returned to a routine under test. This routine must already have been activated and must have returned control to the tester. If any other routine has been called by the tester in the interim *CONTINUE may not be used. The routine to be re-activated is called at its first instruction in standard TPS form. (i.e. the re-entry parameter is significant).

*CURSOR SYSTEM TESTER MODULE TESTER

FORMAT *CURSOR A, B

PARAMETERS

A, B line and column co-ordinates in simple numeric form.

FUNCTION

Set the simulated cursor position for the simulated screen. This would normally be used preceding a *SEND command to control the amount of data passed from the simulated screen into the message area.

*CWARN MODULE TESTER ONLY

FORMAT *CWARN A

PARAMETER

A	any character other than a space

FUNCTION

Cause the character 'A' to be used as the warning character preceding a command instead of the standard '*'.

```
e.g.    *CWA #
        will cause '#' to be needed for following commands
        such as #DATA
                #OUT etc.
```

*DAS MODULE TESTER ONLY

FORMAT *DAS

PARAMETERS

None

FUNCTION

Combines the effect of *DISPLAY followed by *SEND (Display And Send). It is provided because these commands often occur as a pair after a simulated screen has been set up.

*DATA SYSTEM TESTER MODULE TESTER

FORMAT *DATA A, B

PARAMETERS

A	the identifier of a store area in any of the forms described in section 1.5.
B	the size, in words, of the area to be accessed.

FUNCTION

*DATA is followed by one or more data records which will be transferred into the specified location. The number of words specified by B will be transferred. If B is omitted the whole of the following records will be transferred until a command record is encountered, a command record being one which contains an asterisk (or alternative warning character if *CWARN has been used) in the first character position. However, the number of words transferred will not exceed the default size associated with the identifier 'A'.

*DEFINE SYSTEM TESTER MODULE TESTER

FORMAT *DEFINE A,B,C

PARAMETERS

A	a new store area identifier.
B	any store area identifier as described in 1.5.
C	a number of words.

FUNCTION

Set up a new mnemonic identifier for a store area. The address associated with that identifier is given by parameter B in any of the forms described in section 1.5. The parameter C gives the size to be associated with the new identifier A.

Parameter C may be omitted in which case the length L associated with identifier B is used. See 1.5.3.

*DISPLAY SYSTEM TESTER MODULE TESTER

FORMAT *DISPLAY

PARAMETERS

None

FUNCTION

Print, on the Line Printer, the contents of the screen image held in store, including the current contents of the unprotected areas. It is used to verify that the test data is set correctly in the form in which it will appear from an actual terminal.

For the Module Tester a limited facility is currently provided to set up output, generated by a routine under test, within the screen image for output via this command. Data supplied as TEXT (£TTXT, TPSCTXT) will be set correctly in the screen image, but FORMATS or STANDARD MESSAGES are not available to the tester. If these cannot be supplied at a suitable intermediate point by *FORMAT command the image will be incomplete.

*END SYSTEM TESTER ONLY

FORMAT *END

PARAMETERS

None

FUNCTION

Terminate processing of the current message. It will be effective after the Tester detects that an Application Routine to which control is to be transferred is missing.

(*END is equivalent to £TEND in a PLAN routine or TPSCEND or the 'X' sequencing parameter in a COBOL routine).

*ENTER SYSTEM TESTER MODULE TESTER

FORMAT *ENTER

PARAMETERS

None

FUNCTION

Cause the screen identifier in the Message Area to be used as an identifier indicating the routine to be activated. The routine is called at its first instruction.

*FILE MODULE TESTER ONLY

FORMAT *FILE A

PARAMETER

A in the format F(g) where F is a filename and g is a generation number.

FUNCTION

Defines the name of the file that is to be used for holding the simulated screen images and should be used before either the *OPEN or *INITIALIZE command is used to open the file. The generation number, together with enclosing parentheses, may be omitted if the file of given name but highest generation is required.

*FORMAT SYSTEM TESTER MODULE TESTER

FORMAT *FORMAT A,B

PARAMETERS

A,B line and column co-ordinates, in simple numeric form, at which the simulated cursor is to be left after constructing the screen image.

FUNCTION

*FORMAT is followed by data records which contain data to create the screen image in its "blank" form (i.e., without data in the unprotected areas). Each position on the screen must be represented by a character (i.e., trailing spaces on each line must be punched) except that trailing whole lines of spaces may be omitted.
The screen area is cleared to spaces so that residual data from previous formats will not appear in trailing blanklines. Start of unprotected and start of protected data areas are represented by [and] respectively.

The cursor position defined by A and B is stored to be used for input of variable data by *INPUT command.

*HALT SYSTEM TESTER MODULE TESTER

FORMAT *HALT A

PARAMETERS

A a two character mnemonic to be displayed on the console.

FUNCTION

Stop the program with the message HALTED A.

If A is not specified in the command "TP" will appear in the message.

*IDF MODULE TESTER ONLY

FORMAT *IDF A

PARAMETER

A in the format
F (g)
where F is a file name
and g is a generation number.

FUNCTION

*IDF defines a file that holds data, for test purposes, held in simple subfiles of that file. Such data subfiles may be created and edited with the standard program #XMED.
The generation number together with enclosing parentheses may be omitted and then the file of given name and highest generation number is used.

*INITIALISE MODULE TESTER ONLY

FORMAT *INITIALIZE

PARAMETERS

None

FUNCTION

Performs two functions; firstly the file that is used to hold the simulated screen images will be opened (unless this has already been opened by a previous command) and secondly the file will be cleared of stored screen images.

*INPUT SYSTEM TESTER MODULE TESTER

FORMAT *INPUT A, B

PARAMETERS

A,B line and column co-ordinates in simple numeric form.

FUNCTION

*INPUT is followed by data cards the contents of which are inserted into the "unprotected" areas of the screen image, starting from the position defined by the co-ordinates stated in the parameters A,B (line, column).

If the parameters A,B are omitted the data is inserted from the"current cursor position" which will have been set either by the last previous *FORMAT command, the last previous *CURSOR command, the last previous *INPUT command or, in the case of the System Tester, the current cursor position may also be changed when a message is output to a terminal. If the nominated position is within a "protected" area the run will produce an error message. The "current cursor position" will be updated to point to the next unused unprotected character when all the data cards have been inserted. Note that unless a terminator is used the whole of the trailing spaces in the last card will be input. The terminator does not itself increment the cursor position.

*ISF MODULE TESTER ONLY

FORMAT *ISF A

PARAMETER

A the name of a data subfile on the file specified by a previous *IDF command.

FUNCTION

This gives the name of a subfile on the file identified on a preceding *IDF command. When this command is encountered in the data (tape or card reader data) then a search for the given subfile is made. Subsequent commands to the tester are then taken from the subfile but when all data from the subfile is used then further data is expected from the tape or card reader - i.e., from records immediately following the *ISF command. Further *ISF commands may then be given for other subfiles on the same file or another file may be given by *IDF command followed by one or more *ISF commands.

*LIST MODULE TESTER ONLY

FORMAT *LIST

PARAMETERS

None

FUNCTION

Causes the tester to list all subsequent input data records.

*MOP SYSTEM TESTER MODULE TESTER

FORMAT *MOP

PARAMETERS

None

FUNCTION

Indicates that output is to be to a MOP type terminal and therefore the line length used is restricted to 80 characters instead of the 120 characters used for a line printer. With this the lines following an *OUTPUT command will consist of just the character and octal representation of words being given with the decimal representation being suppressed. Also the display of simulated screen (*DISPLAY) will start at column 1 rather than column 21 of the line.

* MOVE SYSTEM TESTER MODULE TESTER

FORMAT *MOVE A, B, C

PARAMETERS

A	store area identifier (see Section 1.5) of 'destination' area.
B	store area identifier (see Section 1.5) of 'source' area.
C	the number of words to be moved.

FUNCTION

Move (copy) a number of words of data from the store area identified by B to the store area identified by A. The number of words moved will be given by C.
If C is omitted, the size defined for area B at generation time will be used.

*NEW SYSTEM TESTER MODULE TESTER

FORMAT *NEW A

PARAMETER

A any four character identifier; not necessarily one set up at generation time.

FUNCTION

Display the identifier named on the console log (or the Monitor File under G3) to show how far the test has successfully reached.

Under the Module Tester the Control Block will be reset to its initial state. The *NEW command is sometimes used to define a restart point in the input test file in the event of an error (See Section 1.1.2 and 1.2.2.)

*NEXT SYSTEM TESTER ONLY

FORMAT *NEXT A

PARAMETER

A a sequencing parameter (1 - 4094), as used on exit from an Application Routine.

FUNCTION

Cause the System Tester to transfer control to a new routine in an Application Routine Train. It will be effective after the Tester detects that an Application Routine, to which control is to be transferred, is missing. (*NEXT is equivalent to £TNXT in a PLAN routine, or, in a COBOL routine, to TPSCNXT or a numeric sequencing parameter supplied to any other TPS COBOL interface routine).

*NOLIST MODULE TESTER ONLY

FORMAT *NOLIST

PARAMETERS

None

FUNCTION

Cause the tester to suppress the listing of input data records. This is only needed when *LIST has previously been used, as the default action of the tester is to suppress the listing.

*OPEN MODULE TESTER ONLY

FORMAT *OPEN

PARAMETERS

None

FUNCTION

Open the file that is used to hold the simulated screen images.

*OUTPUT SYSTEM TESTER MODULE TESTER

FORMAT *OUTPUT A,B

PARAMETERS

A	the identifier of a store area in any of the forms described in section 1.5.
B	the size in words of the area to be output.

FUNCTION

Causes the contents of store to be printed starting at the location specified by A, and continuing for B words. If B is omitted and A is one of the identifiers described in section 1.5.1. the whole of that area will be printed. Otherwise, for any other identifier, a single word will be printed.

Words are printed four per line in a character, octal and decimal representation.

*PATCH SYSTEM TESTER MODULE TESTER

FORMAT *PATCH A

PARAMETER

A	an Identifier (set up as described in Section 1.5) of an area to receive the contents of the following data cards; may either be PA (the standard Mnemonic for a patch area set up at generation time) or in the form SEGNAME+n.

FUNCTION

The contents of the following data cards (which will be program instructions) are set up in the nominated area.

This facility is unlikely to be of practical use to COBOL programmers.

*PRINT MODULE TESTER ONLY

FORMAT *PRINT A, B

PARAMETERS

A	the identifier of a store area in any of the forms described in section 1.5.
B	the size in words of the area to be printed.

FUNCTION

Cause the contents of store to be printed starting at the location specified by A, and continuing for B words.

If B is omitted the size associated with the identifier A is used.

One word is printed per line in character, octal, decimal and instruction represention.

*RECALL MODULE TESTER ONLY

FORMAT *RECALL A

PARAMETER

A	a screen identifier

FUNCTION

Recall the named simulated screen image from the disc file provided for this purpose. The file must previously have been opened with either an *OPEN or *INITIALIZE command.

*REFORM SYSTEM TESTER MODULE TESTER

FORMAT *REFORM

PARAMETERS

None

FUNCTION

Reset the screen image to its state as set by the last preceding *FORMAT command. Also reset the stored "current cursor position" to that left by the last preceding *FORMAT.

Under the System Tester *REFORM may not be used in any input test file for which "SUP" was used as a parameter to the £TTPT macro. (see System Generation Manual Part 5, Section 2.2.1.)

*RESUME MODULE TESTER ONLY

FORMAT *RESUME A

PARAMETER

A	the identifier of a program instruction.

FUNCTION

Cause the program control to branch to the instruction identified by A. A may be the mnemonic identifier associated with an Application Routine or an absolute program location or the identifier of a work area if for example program instructions had been input there by the *PATCH command.

*SEND SYSTEM TESTER MODULE TESTER

FORMAT *SEND

PARAMETERS

None

FUNCTION

Take the contents of the unprotected area of the screen image and move them into the message area linked to the Control Block. This is done exactly as if the message had been transmitted from a terminal. Under the Module Tester the first three or four characters of the message are then used as an identifier associated with which is an Application Routine (this association is set up at generation time). The associated Application Routine is entered as a result of the *SEND command.

Under the System Tester, an Application Routine Train is located in the normal way and control transferred to the first Routine in the Train.

*SIZE MODULE TESTER ONLY

FORMAT *SIZE A,B

PARAMETERS

A	the number of lines on the screen image held in the tester.
B	the number of columns on the screen image.

FUNCTION

*SIZE specifies the size of screen image to be represented in the Tester. Normally the tester uses an image 25 x 80 characters in size. This command is needed only for any other size.

*STORE MODULE TESTER ONLY

FORMAT *STORE

PARAMETERS

None

FUNCTION

Store the image of the current simulated screen, together with the current simulated cursor position, in the disc file provided for this purpose. The file must have been previously opened with either an *OPEN or *INITIALIZE command. This particular screen will be identified by the screen identifier. If a screen image with the same identifier has already been stored on the file then that screen image will be overwritten by the current image being stored with the *STORE command.

* TERMINAL SYSTEM TESTER ONLY

FORMAT *TERMINAL A

PARAMETER

A	a terminal identifier (1 - 4095)

FUNCTION

Change the terminal identifier associated with the input test file to the specified identifier.

*VDU SYSTEM TESTER MODULE TESTER

FORMAT *VDU A

PARAMETER

A any character other than a space.

FUNCTION

Cause the character 'A' to be used as the terminator character for the string of characters being input to the simulated screen following an *INPUT command. In the absence of this command the terminator character is ← .

*ZEROISE SYSTEM TESTER MODULE TESTER

FORMAT *ZEROISE A,B

PARAMETERS

A the identifier of a store area in any of the forms described in section 1.5.

B the size of the area.

FUNCTION

Zeroise B words of the area of store identified by the parameter A. If B is omitted then the size associated with the identifier A is used instead of parameter B.

1.5 ESTABLISHING IDENTIFIERS

Mnemonic identifiers are used within the Command Records to define areas of store or program routines. Some standard identifiers are used in the Tester programs but others may be generated when the Control Segment is constructed (see System Generation Manual Parts 4 and 5). The following conventions are used in establishing identifiers:

1.5.1. Store Area Identifiers.

1. A set of standard identifiers is recognised by the program. These identifiers are as follows:

CB	The Control Block
MS	The Message Area
C1	Terminal Control Record - System
C2	Terminal Control Record - User
Wn	$1 \leq n \leq 9$. User Work Area n ≡ TPSLINKn
Wnn	$10 \leq nn \leq 20$. User Work Area nn ≡ TPSLINKnn
SA	The Scratch Area.
PA	The Patch Area - for patching the program.

These identifiers do not need to be set up by the user. Each of these identifiers has associated with it a size for the area which it defines - see section 1.5.3.

2. Additional identifiers may be defined by the user. These are created in the Control Segment by the macro £TTAI in the Module Tester or the macro £TSAI in the System Tester, or they may be created at run time by the command *DEFINE. The area to which an identifier refers may be defined in one of three ways:

(i) by reference to one of the standard mnemonic identifiers in (1) above or to a user identifier previously defined.

e.g. A new identifier 'WORK' refers to the area starting at word 50 of the additional store area number 1, and 'SPARE' starts at word 15 of the 'WORK' area.

```
∴ WORK = W1 + 50
  SPARE = WORK + 15          are valid.
```

(ii) by reference to the name of an area that is defined in the same Control Segment

e.g. NEWID = AREA

(Common Areas may be set up by use of the macros £TTDA or £TSDA in the Module or System Testers respectively)

(iii) by an absolute program location, i.e. an integer number expressed in octal or decimal.

Each identifier created as in (i), (ii) or (iii) above must have associated with it the size of the area which is thereby defined - see section 1.5.3.

3. Usually when any identifier as defined above is used in a command in a tester, it is also valid to use instead an absolute program location, expressed in octal or decimal form. When this is used, the size of the area associated with that address is taken as 1 word.

1.5.2. Program Routine Identifiers

Identifiers for routines to be tested may be set up by using the macro £TTAR in the Module Tester or £TSAR in the System Tester. An identifier 'SEG' may be defined as

SEG = SEGNAME (or SEGNAME +n, see below)

These identifiers may be used in conjunction with commands in the same way as store area identifiers. In addition, in the Module Tester, they may be used to enter Application Routines. This is achieved in either of the following ways:

1. If the mnemonic identifier is the same as the Message Type identifier set up either in the simulated screen image or directly in the message area, then the command *SEND or *ENTER respectively may be used to enter the routine associated with that identifier. This use is appropriate to testing an A.R. which expects to act on a message in the message area.

2. Otherwise the mnemonic identifier may be used with a *CALL command to enter any routine, e.g. a free standing subroutine or an A.R. from later parts of an A.R. train.

If the expression 'SEGNAME +n' is used in defining an identifier, then the identifier may be used to enter a routine at an intermediate point. This might be used to test parts of a routine written in PLAN, but should not be used for a COBOL routine as such routines must be entered at the start to pick up the linkage addresses.

A program routine identifier has associated with it a size of 1 word, see 1.5.3.

1.5.3. Size Associated with Identifier

Some of the commands available in the tester such as *DATA, *OUTPUT or *PRINT have as their first parameter an identifier which defines an address of store within the program. A second parameter associates that identifier with a size in words of the area for that particular command.

```
e.g.  *OUPUT W1, 25
```

would result in the first 25 words of the 1st additional store area being output.

However in such cases it is possible to omit the second parameter

e.g. *OUTPUT W1

and then the whole of the 1st additional store area would be output.

To facilitate this, each identifier has associated with it a size which is used if the size parameter is omitted from any command.

The standard identifiers (see 1.5.1.) have the following standard sizes in words

CB	60
MS	512
C1	512
C2	~~512~~
W1, W2, W3	512
W4 - W20	1 in Module Tester, 512 in System Tester.
SA	50
PA	100

The user identifiers have a size as defined by the userin the Control segment. Also the standard identifiers can have their size altered by redefining the identifier with a new size.

A command such as *OUT SA

will then result in 50 words of the scratch area being output.

However a command *OUT SA + 20

would result in 30 words (i.e. size 50 - 20 words) being output from word 20 of the scratch area.

Similarly the commands *DATA SA
or *DATA SA + 20

would prepare the testers to accept data from following records of either 50 words or 30 words in length. If whilst trying to find that number of words of data another command record was found then the previous *DATA command would be terminated.

The same type of consideration applies with the *DEFINE command.

e.g. *DEFINE FRED, SA+25, 10

would introduce a new identifier 'FRED' with associated length of 10 words.

*DEFINE BERT, SA+25

would introduce an identifier 'BERT' with associated length of 25 words (SA size 50 - 25).

1.6. SIMULATED SCREEN IMAGE

Both the Module Tester and the System Tester may be used with simple commands which enable data to be put in various areas, routines to be entered, and, on return from routines, for contents of areas to be output. However in order to simplify checking of data associated with both input and output messages, then the 'Simulated Screen Image' feature of the testers may be employed.

For this the testers have an area of store reserved which at any time will hold a representation of the appearance of a VDU screen appropriate to the particular test. The contents of this area will consist of characters in the standard ICL 64 character code and therefore certain character conversions are needed.

e.g. Lower case letters will be held as equivalent upper case.

In particular:
Start of unprotected (steady or flashing) is shown by [.

Start of protected (steady or flashing) is shown by] .

1.6.1. Use in Module Tester

In the Module Tester the screen image may be examined at any time by input of the command *DISPLAY.

Commands to directly alter the screen image are:

*FORMAT	followed by data records to represent the new contents of the screen.
*RECALL	to overwrite the screen image with one that has been stored on a Format file with a previous *STORE command.
*REFORMAT	to reset the screen image to the state it was immediately after the last *FORMAT or *RECALL command.
*INPUT	to add data to the unprotected areas of an existing screen image.

The screen image is also altered as a result of messages output from A.R.'s under test. For the Module Tester the normal TPS formats and standard messages are not available and the following events therefore occur.

The tester distinguishes between screen images which have been set up and sent (*SEND) to an A.R., and those set up but not sent. The latter are assumed to have been set up just to receive an output message and therefore requests for clear screen and/or a format are ignored. In the former case however, the screen image set up was an input screen and therefore requests for clear screen and/or a format are intercepted - a clear screen request would be obeyed and a request for a format would have one of two effects depending on whether or not a Module Tester format file was in use -

-if no file, then a format request would result in the screen image being space filled.

-if file available, then an attempt is made to find a Module Tester format of the required name - if found this will become the new screen image; if not the screen is space filled.

Irrespective of which of the above actions occurs then texts set up by the use A.R. are added to the screen image but all standard message requests are ignored.

1.6.2. Use in System Tester

In the System Tester the screen image is output whenever an A.R. 'Edits and Sends' a message to a terminal. In addition, if a *DISPLAY command is input then the screen image is output everytime an exit from an A.R. to TPS is made.

The screen image may be directly set up using the commands

```
    *FORMAT
and *INPUT
             as described in 1.6.1. above.
```

The screen image is also set up from output messages - when using the System Tester the normal TPS formats and standard messages should be available and therefore the simulated screen image is constructed completely in accordance with the output parameters.

1.7. ERROR MESSAGES

The following error messages may be output on the line printer

COMMAND IS UNRECOGNISED	An asterisk or other warning character has been detected in column 1 of an input record but the next 3 characters are not recognised as a command.
AREA IS UNDEFINED	The identifier following a command is not a known mnemonic identifier.
AREA FORMAT INCORRECT	The identifier following a command contains unacceptable characters.
COMMAND WAS EXPECTED	The current input record should have been a command record but is not.
*INP - PROTECTED AREA	The co-ordinates at which data is to be input on the screen image lie within a protected area of the image.
*PAT - PLAN MNEMONIC UNKNOWN	Following a * PATCH command the contents of the instruction field of a PLAN statement are not recognised by the tester.
*PAT - ACC FIELD INVALID	The accumulator field of a PLAN statement following *PATCH command does not contain 0-7 or spaces.
DEFINITION SPACE FULL	In any run of the tester only 20 *DEFINE commands may be used - this has now been exceeded.
Module Tester Only:	
FILE NOT OPEN	A command referring to a format on a Module Tester format file has been used but the file has not been opened.
FORMAT NOT ON FILE	The format being 'recalled' from a format file could not be found.

2. Diagnostic Facilities

2.1. THE DIAGNOSTIC BUFFER

The Diagnostic Buffer is an area in store into which TPS places records that define events which have taken place within the system. (User routines may also enter similar information into the buffer.) When the buffer becomes full, entries are made from its start, progressively overwriting the previous contents. Thus a record of the most recent events is always available.

2.1.1. Format of the Buffer

The Diagnostic Buffer is of user specified length and is made up of an undefined number of variable length Entries. Each Entry, which may be supplied by TPS or by user Application Routines, is identified and defined by an Identifier/Length Field which occupies the last word in the entry. When the Buffer is first being filled the contents of the unused portion are undefined, except for the last word which is set to - 1. Once the Buffer has been filled a residual area too small to hold an entry may remain at the end. This will be set as a dummy entry with an Identifier of zero and a Length of the appropriate number of words. The contents of this dummy remain undefined. Note that in all circumstances the length in the Identifier/Length word does not include that word itself.

2.1.2. Defining the Buffer

The buffer is established by a single parameter entered in Form 3 of the System Generation Forms (see Part 1 of the Systems Generation Manual). Its contents are influenced by two other parameters set at generation time. If "FULLDB" is specified on form 12.3 (see section 2.4.12.3 of Part 1 of the System Generation Manual) the "full" entries specified below (section 2.1.7) will be found in the buffer. If this parameter is not specified, the entries for entry to or return from an AR, and for the File Manager block level transfer request, will be handled in a special way. Each entry will be inserted as "full" form initially, but will be compressed into its "short" form when a subsequent entry is made into the buffer. Thus the last entry always appears in full. Even if "FULLDB" is specified the items for entry to and return from a TPS "Application Routine" will be compressed, unless the parameter DIAG1 is also specified on form 12.3.

2.1.3. Use of the Buffer by TPS

TPS makes an entry in the Buffer on each of the following occasions:

1. Whenever a transaction is accepted by Input Handler and passed for processing by an AR.

2. Whenever a unit of overlay is transferred into store.
3. Whenever control is passed to an AR.
4. Whenever control returns from an AR.
5. Whenever an overlay unit is released on terminating an AR.
6. Whenever a request is accepted into File Manager.
7. Whenever a block level transfer is requested within File Manager.
8. Whenever a File Manager request is completed.
9. Whenever a transaction is aborted by the File Manager.
10. Whenever an AR requests an Output to a terminal.
11. Whenever a Minor Logic Error occurs. + others

The Formats of these entries are found in section 2.1.7.

2.1.4. Use of the Buffer by a COBOL Application Routine

The writer of a COBOL A.R. may insert entries into the Diagnostic Buffer using the subroutine TPSCDGN. This subroutine has three parameters, the first of which specifies a two character identifier. This identifier, which must be alphabetic to distinguish it from the TPS entries, which have numeric identifiers, is inserted by TPS into the identifier field of the entry. The other two parameters specify the length and location of the data to be stored. The user must set up the required data in a contiguous area of store before making the call to TPSCDGN (note that this is in contrast to the procedure in a PLAN A.R.). The area is then moved into the buffer, and the identifier field is added as a further word after the data itself.

2.1.5. Use of the Buffer by a Plan Application Routine

A PLAN routine can acquire an entry in the Diagnostic Buffer by use of the Macro £TDGN. This Macro has three parameters. The first gives an Identifier to be set in the buffer to enable the entry to be recognised. The second gives the length of the entry. On the basis of these two parameters the software allocates the required entry and passes its address to the AR in the accumulator nominated as the third parameter. The user AR may then move data into the given area without the need to first bring the data together in a single area. Although there is no requirement to complete the entry immediately it should be borne in mind that, if this is not done, the entry could still be incomplete at the time of a system failure.

(Contd.)

If the setting of data into the buffer is fragmented, and a request to TPS is made before the entry is complete, it is possible that the buffer may have cycled completely before return is made to the AR in question. In this case any later data will corrupt a subsequent entry. It should therefore be ensured that all the data to be placed into a single entry in the buffer has been completed before any entry to TPS.

2.1.6. Accessing the data in the Diagnostic Buffer

The contents of the buffer can be accessed in three ways:

A The buffer can be printed in a fully self explanatory format in the event of a program halt by restarting with "GO 25". A line printer must be available at this time. The same output can be achieved by keying in the four character message *DBP from any terminal. This has the advantage of printing the buffer before any further entry can be made in it.

B The buffer can be dumped to magnetic tape or disc. This is fully described in Part 4 of the Operating Manual.

The tape or disc buffer may be printed in the same self explanatory format by running the program TPSD.

Full operating instructions for the program are found in Part 5 of the Operating Manual.

C The contents of the buffer may be interpreted from a core dump. The buffer itself and the pointer fields which give access to it are Common Areas, and therefore appear in the Consolidation listing.

The names to look for are:

TPSDBUFF	:	the start of the buffer itself.
TPSDBCURR	:	a current pointer to the buffer.
TPSDBCURRL	:	the length and Identifier of the last entry allocated in the buffer.
TPSDBEND	:	the address of the first word after the end of the buffer.

When requested to allocate an entry in the buffer, the diagnostic routine stores the Identifier and length, given as parameters with the request, in TPSDBCURRL and returns to the requesting routine the pointer stored in TPSDBCURR. When the next request is received the previously stored identifier/length will be removed from TPSDBCURRL; the length field will be added into TPSDBCURR thus moving the pointer, and the Identifier/length word will at that time be stored in the buffer itself at the end of the previous entry (i.e. the one to which it related). The newly given identifier/length is then stored in TPSDBCURRL and the newly evaluated pointer in TPSDBCURR is passed to the latest requesting routine.

To interpret the contents of the buffer, the following steps should be taken:

1. The contents of the location TPSDBCURR are the address of the start of the last entry in the buffer. The composition of this entry can be established from the Identifier and length held in TPSDBCURRL, remembering that it is possible that this entry, although allocated, has not actually been inserted.

2. The contents of the preceding entry can be evaluated by taking the word preceding the word addressed by TPSDBCURR, which will again be an Identifier/length word, and by counting backwards the given number of words.

3. This process may be repeated until the start of the buffer is reached. At this point the contents of TPSDBEND may be used to locate the end of the buffer (i.e. contents of TPSDBEND -1).

4. The last word in the buffer is then considered. This may be a normal Identifier/Length word, or a word with an identifier of 00 (zero), indicating that an unusable number of words remained at the end of the buffer when "wrap around" occurred. This dummy entry should be "stepped over" using the length field in the identifier/length word, and interpretation of the true entries may continue.

If the last word in the buffer is found set to -1 this implies that the buffer has not yet been wholely filled and that all available entries have been considered.

5. Once the jump forward to the end of the buffer has occurred the process of stepping back down the entries may continue. Eventually the length of a particular entry will point to a start point beyond the original entry indicated by TPSDBCURR, at which point no further entries are "visible" - i.e. at this point the advance of new data overwriting old has been encountered. If short entries are being used there may be a number of non-significant words after the end of the entry pointed to by TPSDBCURR. In this case, the last word of significant data can be found by adding to the address in TPSDBCURR the length from bits 12-23 of word 0 of TPSDBCURRL, and the value contained in word 1 of TPSDBCURRL.

2.1.7. Format of TPS Daignostic Buffer Entries

The full list of TPS entries, with their identifiers, is as follows:

Record Identifier	Routine generating entry
01	A.R. Return
02	A.R. Entry
03	Overlay Control 1
04	Overlay Control 2
05	Input Handler
06	Output Handler
07	Logic error 2
08	Deadly Embrace Manager.
09	Deadly Embrace Manager.
10	File Manager (Entry)
11	File Manager (Exit)
12	File Manager (Releasing locks)
13	File Manager (Pericon Call)
14	File Manager (aborting trans.)
15	File Manager (Pericon Exit)
20	Total Facility A.R. (Entry)
21	IDMS Facility AR (Entry)
22	Prior to EDITOR outputting a window.
23	IDLC Input Transmission

Records type 20 and 21 are only written if switch 10 is on.
12, 15

The contents of each entry are in accordance with the following reference sheets:

IDENTIFIER 01

EVENT On Exit from an AR to TPS

FULL ENTRY

WORD	CONTENT	WORD	CONTENT
0	Control Block Address	19	)
1	Message Type	20	)
2	Queue Link	21	) Request Parameters
3	Home Queue Link	22	)
4	Weighting	23	)
5	User Accumulator 0	24	)
6	" " 1	25	TPS
7	" " 3	26	TPS
8	" " 4	27	FM Reply Word
9	" " 5	28	TPS
10	" " 6	29	Current Displacement from Start of AR Train
11	" " 7		
12	System Terminal Number	30	Address of Current AR Train
13	)	31	TPS
	Message Serial Number		
14	)	32	TPS
15	Current AR Number	33	TPS
16	AR Exit Parameter	34	TPS
17	Re-entry Address or Re-entry number	35	TPS
		36	TPS
18	Request Word	37	Address of Message Area
		38	Address of System TCR
		39	Address of User TCR (0 if no user TCR)
		40+	User Links
		Last User Link + 1	01nn (ID/length) where "nn" is length of Control Block + 2

Note: Words 3-40+ represent words 0-End of the Control Block

SHORT ENTRY

0	Control Block address	3	Re-entry address or re-entry number
1	Current AR Number		
2	AR Exit Parameter	4	Request Word
		5	0105 (ID/length)

IDENTIFIER 02

EVENT Prior to entry into an AR from TPS

FULL ENTRY

WORD	CONTENT	WORD	CONTENT
0	Control Block Address	19	)
1	Message Type	20	)
2	Queue Link	21	)
3	Home Queue Link	22	) Request Parameters
4	Weighting	23	)
5	User Accumulator 0	24	)
6	" " 1	25	TPS
7	" " 3	26	TPS
8	" " 4	27	FM Reply Word
9	" " 5	28	TPS
10	" " 6	29	Current Displacement from Start of AR T
11	" " 7		
12	System Terminal Number	30	Address of Current AR Train
13	) Message Serial Number	31	TPS
14	)	32	TPS
15	Current AR Number	33	TPS
16	AR Exit Parameter	34	TPS
17	Re-entry Address or Re-entry number	35	TPS
		36	TPS
18	Request Word	37	Address of Message Area
		38	Address of System TCR
		39	Address of User TCR (0 if no user TCR
		40 +	User Links
		Last User Link + 1	02nn (ID/length) where "n/n" is length of Control Block + 2

Note: Words ~~0~~ 2-40+ represent word 0-End of the Control Block

SHORT ENTRY

0	Control Block address	3	Re-entry address or re-entry number
1	Current AR Number	4	Request Word
2	AR Exit Parameter	5	0205 (ID/length)

IDENTIFIER 03

EVENT Release of an overlay

WORD	CONTENT
0	Control Block Address
1	AR Number
2	Area/Unit to be freed
3	Free/not freed FREE/0
4	0304 (ID/length)

IDENTIFIER 04

EVENT Request for an overlay

WORD	CONTENT
0	Control Block
1	AR Number required
2	Area/Unit required
3	Demand raised or not DEMA/~~0~~REQU
4	0404 (ID/length)

IDENTIFIER 05

EVENT Passage of an Input Message to its first AR

WORD	CONTENT
0	Control Block Address
1	AR number to be entered
2	Terminal Number
3	) Message Serial Number
4	)
5	Message Type
6	0506 (ID/length)

IDENTIFIER 06

EVENT Passage of an Output Message for transmission

WORD	CONTENT
0	Control Block Address
1	Number of the current AR
2	Terminal Number originating request
3	Terminal Number for output
4	) Message Serial Number
5	)
6	Expected Screen
7	Previous expected
8	0608 (ID/length)

DB 07

IDENTIFIER 07

EVENT Occurrence of a Minor Logic Error

WORD	CONTENT
0	Control Block Address
1	Message Type (4 characters)
2	Terminal Number
3	) Message Serial Number
4	)
5	Request Code
6	Accumulator 0
7	" 1
8	" 3
9	" 4
10	" 5
11	" 6
12	" 7
13	Application routine number
14	0714 (ID/length)

IDENTIFIER 08 or 09

EVENT On action by Deadly Embrace Manager.

WORD	CONTENT	WORD	CONTENT
0	Control Block Address	20	Request Parameters
1	Message Type	21	
2	Action	22	
3	Queue Link	23	
4	Home Queue Link	24	
5	Weighting	25	
6	User Accumulator 0	26	TPS
7	,, ,, 1	27	TPS
8	,, ,, 3	28	FM Reply Word
9	,, ,, 4	29	TPS
10	,, ,, 5	30	Current Displacement from Start of A.R. Train
11	,, ,, 6	31	Address of Current AR Train
12	,, ,, 7	32	TPS
13	System Terminal Number	33	TPS
14, 15	Message Serial Number	34	TPS
16	Current A.R. Number	34	TPS
17	A.R. Exit Parameter	36	TPS
18	Re-entry Address or Re-entry number	37	TPS
19	Request Word	38	Address of Message Area
		39	Address of System TCR
		40	Address of User TCR (0 if no user TCR)
		41+	User Links
		Last User Link + 1	08nn or 09nn (ID/length) Where "nn" is the length of Control block + 3

IDENTIFIER 10

EVENT Entry with a new request into File Manager

WORD	CONTENT
0	Control Block Address
1	IN
2	Message Serial Number
3	Request code
4	1004 (ID/length)

IDENTIFIER 11

EVENT Exit from File Manager on completing a request

WORD	CONTENT
0	Control Block Address
1	OUT
2	File Manager reply
3	Request code
4	1104 (ID/length)

IDENTIFIER 13

EVENT File Manager request for bucket transfer

FULL ENTRY

WORD	CONTENT
0	Control Block Address
1	PCB address
2	Logical File number
3	Read/Write/Locks*
4	LBN
5	Request Code
6	Calling link (X7)
7	1307 (ID/length)

SHORT ENTRY

0	CB address
~~0~~ 1	1300 (ID/length)

* The contents of Word 3 have the following significance:

Bit	Value	Meaning
0	0	Dynamic transfer - internal only.
	1	Transfer to/from a nominated area.
16-17	0	Read
	1	Write
	2	Read to end of cylinder
	3	Write to end of cylinder
18-19	-	(Reserved)
20-23	1	Busy lock request
	2	LBN " "
	4	FILE " "
	8	URN " "

IDENTIFIER 14

EVENT Aborting a transaction within File Manager

WORD	CONTENT
0	Control Block address
1	FMAB
2	Request Code
3	Reply word
4	1404 (ID/length)

IDENTIFIER 20

EVENT Entry to TOTAL Facility AR (Switch 10 being ON)

WORD	CONTENT
0	Control Block Address
1	Parameter C Address
2	Parameter B Address
3	End Word
4 - 5	Function (5 characters)
6	Reply word
7	File
8	Reference
9	Expected Reply
10 - 11	Link Path
12 - 14	Key
15	200? (ID/length)

IDENTIFIER 21

EVENT Entry to IDMS facility A.R. (switch 10 being ON)

WORD	CONTENT
0	Control Block Address
1	Function
2	Reply
3	2103 (ID/length)

IDENTIFIER 22

EVENT Prior to EDITOR outputting a window.

WORD	CONTENT
0	Control Block Address
1	Reply
2	Last edit instruction
3	Current record number
4	Flag
5	2205

2.2. PROGRAM DUMPING FACILITIES

This section describes the ways in which the program image may be dumped from store for diagnostic purposes, and the methods by which access may be made to the dumped data.

2.2.1. The Dump File

To use these facilitites, the user must make available a Dump File, as described in the Operating Manual, page 117.

The file must be defined when the System File is generated, as a Main File in Section 02 of the input to TPSM and on forms 10, 11 and 12 of the Control Routine.

When the Dump File is made available in this way it will be used by the system to output the current store image in the following circumstances:

1. on discovering a Major Logic Error
2. on starting the program at entry point 26 (e.g. in the case of an ILLEGAL message at the central console)

The program may be dumped several times to the file, the number being governed by the parameter set in Form 12.2 which in turn is constrained only by the size of the file allocated. When the system is started normally (a "Cold" start) the file is reset and used from the beginning. When the system is restarted in a "warm" start previous dumps are retained; any further failure causes a subsequent dump to be made unless the file becomes full, in which case the program merely halts.

2.2.2 Analysing the Dump File

The contents of the Dump File may be analysed by running the program TPSP which will print all or selected parts of a nominated dump within the file. Full instructions for this program are contained within Part 5 of the Operating Manual.

2.2.3 On line access to a dump, or to the contents of store

The contents of a dump in the dump file can be displayed at a video terminal in the same format as that of the TPSP line printer output (see Operating Manual, Part 5). This is achieved by the use of the standard function YCOR as defined in the System Designers Manual, Part 1, Section 8.7B This function may also be used to display the current contents of store in the same format.