# MH$ - Call Menu Handler

The MH$ routine ...

## 1. Invocation

????

## 2. STOP Codes and Exception Conditions

????

## 3. Programming Notes

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.

### 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(4)
*
 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(4)
77 MXCLOS REDEFINES MICLOS PIC X(4)
77 MXSECR REDEFINES MISECR PIC X(4)
77 MXSECW REDEFINES MISECW PIC X(4)
*
PROCEDURE DIVISION
*
MOVE #00000000 TO MIPTRX            * Set special value
CALL MH$ USING MIPTR
MOVE #00000000 TO MXSACR MXCLOS MXSECR MXSECW
 . . .
 * code to establish fields in MI
 . . .
CALL MH$ USING MIPTR                * Invoke handler to display
                                    * and get response from menu
ON EXCEPTION
*     Exception handling
END
 . . .
 * code to process results from handler
 . . .
EXEC MIPROG                   * Execute selected program
*
ENDPROG
ENDSOURCE
```

### 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 ofdata 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 hem.

### 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).

# 4. Examples

# 5. Copy-Books

See copy-book "M$" in copy-library S.SYS32 for a definition of the ....

# 6. See Also