OMI v2.3
Oscar's Menu Interpreter for VMS
Reference Guide

Oscar van Eijk
Oveas Functionality Provider
oscar.van.eijk@oveas.com
http://freeware.oveas.com/omi

Belgium, June 25, 2004

Contents

List of Figures

List of Tables

Preface

Oscar's Menu Interface is a DCL program that reads menu files and represents the menu structure on a ChUI based terminal. The layout depends on the user specific configuration.

OMI does not perform any actions on its own. Additional procedures are required to handle upon the user input. OMI is strictly created to create a standard user interface for all kinds of actions, with a clear structure, support for different security levels and password protection.

The additional procedures that are required for any actions are referred to as OMI modules. Several OMI commands have been created for use in OMI modules, and with the expandable Toolbox, it's easy to add your own commands.

Known Bugs

The following bugs have not been solved yet:

- When *OMI$DISPLAY_INFO* needs to display a blank line between two records, the *OMI$RECORD*n needs to contain at least one blank.
  E.g. when the next three lines need to be displayed:
  OMI$RECORD1 = "The first line"
  OMI$RECORD2 = "" ! A blank line to separate
  OMI$RECORD3 = "The last line"
  the command *OMI$DISPLAY* will display only the first line. This can be solved by specifying *OMI$RECORD2* as "␣" (a blank).
- When a *TEXTAREA* input element is defined with the option *LARGE* set to *TRUE*, the command *SHOW TEXTAREA* shows nothing in certain circumstances without warning.
- When a library file is included with *#INCLUDE*, and duplicate values are encountered, a leading value can be specified with the *#LEADING* attribute as described in section 5.7.1. However, when the menu file is validated with the */VALIDATE* qualifier, a warning will be displayed if the first item of two duplicate item names was specified as leading. During run-time this is handled properly.
- Switching menus using the *MENU* command causes `%OMI-W-DUPL` messages. After this, OMI can crash at random moments without warning.

About this document

Conventions

Names All OMI literal names (filenames, logicals etc) are displayed in a slanted font.

Literals This sans-serif font is used when literal text is displayed as it should be entered by the user.

Variables When the user should replace the text as displayed with a environment-specific value, the text is printed italic.

`Examples` All examples are printed in this monotype font.

Copyright

Support and the latest version is available at
http://freeware.oveas.com/omi
For questions, bug reports, feature requests etc., please use the OMI forum located at
http://freeware.oveas.com/omi/forum.html

Acknowledgements

This document was typeset with LaTeX 2$_\varepsilon$ – Lesle Lamports document typesetting system based upon Donald E. Knuth's TeX.
The PDF version was created with `dvipdf`, GNU Ghostscript 6.52.
The Postscript version was created with `dvips` v5.86.
The HTML version was created with LaTeX2HTML v2002 (1.62).

Henry Juengst made me aware of possible crashes and gave me the code that resulted in the `OMI$INPUT_VALIDATE` command (v1.0b4).
Edward Vlak did some debugging and fixed a possible loop in `REQUIRED_ORDER` (v2.2).

Chapter 1

Installation

1.1 System installation

To install OMI, download one of the distributions: either the ZIP file[1] (download in BINary format) or the COM file (download in ASCII format). Also, download *OMI$INSTALL.COM* in ASCII format.

Next, set default to the directory where the installation script is located, and issue the command:

```
$ @OMI$INSTALL
```

Follow the instruction to install OMI. The procedure extracts the distribution and moves all files to the desired location (that you will be prompted for).

The installation procedure also creates the HELP library, and repairs message datafiles, which have been corrupted by the distribution.

1.2 User installation

1.2.1 User specific installation

Let all the users who will be using OMI execute the procedure *OMI$USER_INSTALL* once. They should use the command:

```
$ @<OMI_location>OMI$USER_INSTALL
```

This will create the file *OMI$SETUP.COM* in the *SYS$LOGIN:* directory of the user, and creates a user specific directory to contain OMI menu files (default is *[.OMI]* under the *SYS$LOGIN:*).

Figure 1.2 shows an example session of the *OMI$USER_INSTALL* procedure.

The newly created procedure should be called every time the user logs in, so either the command

```
$ @SYS$LOGIN:OMI$SETUP
```

should be added to the users *LOGIN.COM*, or the following lines should be added to the system-wide login procedure:

[1] This also requires UNZIP for VMS, available from ftp://www.process.com/vms-freeware/

```
$ @omi$install
%OMI-I-INS_SHARE, installing the SHARE distribution
_Install in [$1$DKA200:[APPLIC.OMI]]:
%OMI-I-UNPACK, please wait - unpacking distribution
DELETE $1$DKA200:[TEMP]OMI-V2_1.COM;1 ?  [N]:
%OMI-I-EXISTS, OMI$TOOLBOX.INI exists
-OMI-S-TBPATCHED, OMI$TOOLBOX.INI successfully patched
%OMI-I-EXISTS, OMI$MENU.CFG exists
-OMI-I-NEWNAME, installing new config file as OMI$MENU.CFG_NEW
%OMI-I-CREHELP, creating the help library
%OMI-I-MSGREPAIR, repairing the message datafiles - ignore BADMSGFIL
messages
%OMI-F-BADMSGFIL, message file corrupt - please repair
%OMI-F-BADMSGFIL, message file corrupt - please repair
%OMI-F-BADMSGFIL, message file corrupt - please repair
%OMIMGT-S-MSGREP, message file for OMIMGT successfully repaired
%OMI-S-INSTALLED, OMI succesfully installed
$
```

Figure 1.1: an OMI installation session

```
$ IF F$SEARCH("SYS$LOGIN:OMI$SETUP.COM") .NES. "" THEN -
    $ @SYS$LOGIN:OMI$SETUP
```

Installation is now complete. All users can optionally change the behavior of OMI by modifying the file *OMI$MENU.CFG*, located in their *SYS$LOGIN:*.

1.2.2 Shared installation

To let all users, or groups of users, share OMI installations and configurations, one user should execute the file *OMI$USER_INSTALL*. When the procedure asks for a directory to store the Menu files, enter a shared directory.

After installation is completed, move the files *OMI$SETUP.COM* and *OMI$MENU.CFG* to a location where all users can read them.

Edit the file *OMI$SETUP.COM* and search for the following line

$! DEFINE /NOLOG OMI$CONFIG <*Path and name of your configuration file*>

Uncomment this, and let the logical point to the shared configuration file.

Repeat these steps for all groups.

NOTE: To prevent users to set a key in a shared environment (see section 4.2.1), make sure they don't have WRITE privilege on the configuration file!

1.3 Upgrades

If OMI is already installed on your system, and it just needs to be upgraded, follow the steps in section 1.1. Site specific files will not be overwritten.
User installation (section 1.2) do not need an upgrade.

```
$ @$1$DKA200:[APPLIC.OMI]OMI$USER_INSTALL
Where do you want to store your Menu files (*.MNU) ?
[$1$DKA100:[OSCAR.OMI]]:
Create this directory ?  ([Y]/N)
%COPY-S-COPIED, $1$DKA200:[APPLIC.OMI]OMI$MENU.CFG;4 copied to
$1$DKA100:[OSCAR]OMI$MENU.CFG;1 (2 blocks)
What will be your default printer ?  [SYS$PRINT]:
The file OMI$SETUP.COM has been created in your LOGIN directory.
To be able to use OMI every time you log in, add the command
$ @SYS$LOGIN:OMI$SETUP.COM
to your LOGIN.COM
For modifications to the setup file, refer to the comments.
%OMI-S-SETUP, OMI environment successfully initialized
OMI-I-NEWCMD, the new command OMI has been defined
$
```

Figure 1.2: an OMI user installation session

Figure 1.1 shows an upgrade where the configuration file *OMI$MENU.CFG* is saved, and the toolbox initialisation file *OMI$TOOLBOX.INI* is patched.

1.4 Test the installation

To test OMI, you can use the example menu file *OMI$EXAMPLE.MNU* which comes with the distribution. Start this menu with the command

```
$ OMI OMI$EXAMPLE
```

This menu performs no other action than displaying information. The *Protected menu* is protected with the password test123456789[2].

[2]Passwords are case sensitive.

Chapter 2

Starting OMI

When OMI has been set up properly, as described in chapter 1, OMI can be started from the DCL prompt with the command OMI. If the logical *OMI$STARTMENU* is not defined, this will display a list of all available menus in the directories *OMI$:* and *OMI$MENU_DIRECTORY:*.

Since version 1.4, OMI also reads command line qualifiers[1].

2.1 Parameters

When running interactive, OMI supports three parameters, of which the second and third are obsolete since version 1.4 However, for backwards compatibility, they are still supported. For description of those parameters, refer to section 2.2.7.

The first parameter is the name of the menu file that should be started. This menu file has to be located in the *OMI$:* or *OMI$MENU_DIRECTORY:*. If the parameter is omitted, and the logical *OMI$STARTMENU* (see section 4.1.3), a list with available menus is presented, from which a selection can be made.

When running in the background (see section 2.2.1), OMI supports one parameter, which is the name of the module that should be started in the background.

2.2 Qualifiers

Qualifiers can be specified *after* the first parameter on the DCL command line.

[1]When qualifiers are used, the first parameter, which refers to the menu file, is *always required*, even if the logical *OMI$STARTMENU* is defined and points to the menu file that should be started!

2.2.1 /BACKGROUND[=mode]

/BACKGROUND=BATCH (default)
/BACKGROUND=DETACH

Start an OMI module in the background. The first parameter is now an OMI module name (*filename.OMI*) in stead of a menu name. This module should be created to run unattended (e.g. using Mailboxes for communication, refer to chapter 6 for more information on writing OMI modules).

An OMI module can run as a detached process (requires VMS DETACH privilege), or in a batch queue.

2.2.2 /BATCH

/BATCH
/NOBATCH (default)

Start OMI in Batch mode. In this mode, all output, written to *SYS$OUTPUT*, is suppressed. This also means the menus are not displayed at all. This option is useful when used together with */SUBMENU* and */JUMPS*.

2.2.3 /DEBUG

/DEBUG
/NODEBUG (default[2])

By default, all error messages from DCL are suppressed in OMI. If an error occurs in OMI, it's handled by OMI itself. This should also be the case in all OMI modules, but while these modules are being developed, it is useful to see all messages that are generated by VMS.

With the /DEBUG qualifier, all messages will be displayed. If SET VERIFY is issued before starting OMI, all code is displayed as well.

This qualifier replaces the symbol *OMI$_DEBUG*, that was used until version 1.3 (however this symbol is still supported).

2.2.4 /IGNORE

/NOIGNORE (default)
/IGNORE=*(keyword[,...])*

Ignore specified warnings or errors during the OMI session. If you specify only one keyword, you can omit the parentheses.

DUPLICATES

/IGNORE=DUPLICATES

[2]By default, debug mode is disabled. However, in versions prior to 1.4, the mode was set by the symbol *OMI$_DEBUG*. Setting this symbol to 1 still enables debug mode.

When starting OMI, `%OMI-W-DUPL` warnings are displayed when OMI tries to set a symbol that's already defined. This happens after a crash of the previous OMI session, when the menu file is invalid, or when a new OMI session is started in a subprocess from within OMI (for OTF menus, this is the only way to open a new menu without leaving the OFT context).

The /IGNORE=DUPLICATES qualifier overwrites all existing symbols without a warning.

DCLERRORS

/IGNORE=DCLERRORS

When a DCL command is given in the menu file, or entered at the OMI prompt (commands `DCL` ($) or `SILENT_DCL`), or when a subprocess is spawned from the menu file (as an option) or using the OMI `SPAWN` command, OMI will catch its exit status and act accordingly. It the exit code was an error (severity 2), OMI crashes.

The /IGNORE=DCLERRORS qualifier ignores errors generated in such circumstances.

NOTE: This does not apply to `(SILENT_)DCL` or `SPAWN` action from within OMI modules !

DCLFATALS

/IGNORE=DCLFATALS

This keyword has the same effect as the DCLERRORS keyword, but for fatal exit codes (severity 4).

Both keywords can be given together as /IGNORE=(DCLERRORS,DCLFATALS).

2.2.5 /JUMPS

/JUMPS=option-list

With the */JUMPS* qualifier, all options that are normally entered one by one at the OMI prompt, can now be specified at the DCL prompt as a list of options separated by comma's. If this list with option contains OMI commands with parameters, blanks are required to separate the parameters if they should be specified. The option has to be enclosed by double quotes.

2.2.6 /PROGRESS

/PROGRESS
/NOPROGRESS

This qualifier can be used to overwrite the *SHOW_PROGRESS* element from the configuration file (see section 4.2.1).

2.2.7 /SUBMENU

/SUBMENU=submenu-name

With the */SUBMENU* qualifier, it is possible to start the OMI session with the submenu that has the specified name. By default, OMI starts a menu file with the main menu (see section 5.1.1), but when one ore more submenus have names (see section 5.2.3), OMI can start in one of those menus.

2.2.8 /VALIDATE

/VALIDATE[**=log-file**]
/NOVALIDATE (default)

This qualifier reads the menu file and validates the syntax line by line. It signals errors and warnings when errors or possible errors are found, like calling non- existing submenus, commands or modules, missing elements (e.g. *ITEM1*, *ITEM2*, *ITEM4*...), duplicate elements etc.

If no errors are warnings are found, this does not guarantee a valid menu, since not everything can be checked (especially when variables are used, or logicals defined by an *ON_INIT* module), but it finds the most common errors.

A filename to write all messages to can optionally be specified. If omitted, all messages are written to *SYS$ERROR*.

This qualifier requires WRITE privilege to the menu.

Chapter 3

Using OMI Menus

Oscar's Menu Interface can be used to walk through pre- defined menu structures. Such structures are created in *.MNU* files, which should be located in the same directory where OMI resides (referred to with the logical OMI$) or a special directory pointed to with the logical *OMI$MENU_DIRECTORY*.

To start OMI, just type the command OMI at the DCL prompt. OMI will start a menu if:

1. the logical *OMI$START_MENU* points to a valid menu file in OMI$: (the directory in which OMI is located) or *OMI$MENU_DIRECTORY*:

2. the file *OMI$MENU.MNU* exists in *OMI$:* or *OMI$MENU_DIRECTORY*:

If none of these are found, a list of all available menus in the directories will be displayed, and OMI prompts to specify a menu file. When OMI is started with a parameter, the menu file pointed to by that parameter will be opened. When that file is not found, the user is prompted to enter a new menu file.

3.1 Navigating through menus

OMI Menus display a list of options and/or values that can be filled in, further called 'options'. To select an option, just type the number on the command line.

OMI commands can also be entered at the command line[1]. All OMI commands are described in chapter 9, and in the online HELP.

When input values need to be entered, the option with which all inputs can be entered at once, will always be visible. The name of the options depends on the setting in the configuration file. By default, this is *All inputs*.

To go back one level in the menu structure, you can use the *BACK* command, the <Ctrl/Z> keystroke, or type a *0* (zero), followed by <Return>. When the current menu is the top level of the menu structure, this will exit OMI.

[1] In "on-the-fly" menus—described in chapter 8, not all OMI commands can be used (see also section 8.1.2). The commands that can not be used are marked in chapter 9.

3.1.1 Fast moving

In stead of entering the options, or using *0* or <Ctrl/Z>, there are several commands that can be used for fast navigation. They should be entered at the OMI prompt.

The *MAIN* command jumps immediately to the top-level menu from anywhere in the structure. The commands *EXIT* or *QUIT* end the OMI session.

The command *JUMP*, followed by a menu name, jumps to the specified submenu. Names can be displayed automatically by changing the configuration file (described in chapter 3), or in a menu using the *SHOW NAME* command.

When in a menu, a different menu file can be selected with the *MENU* command, followed by the name of the new *.MNU* file. This command cleans up the current menu completely before starting the new menu.

3.1.2 Auto mode

When the user knows all options, a menu can also be called in Auto Mode. This means, the menu file is specified as the first parameter for OMI[2], followed by the name of a submenu.

The third parameter can be a list of options that needs to be specified, separated by commas. If this list contains OMI commands with blanks in it (e.g. *RESET PASSWORD*), that option has to be enclosed by double quotes, e.g.

```
$ OMI MY-MENU SUB3 "RESET PASSWORD",EXIT
```

or

```
$ OMI MY-MENU /SUBMENU=SUB3 /JUMPS="RESET PASSWORD",EXIT
```

The example above would remove the password of the menu with the name *SUB3*, and exit the menu. However, OMI will prompt for the current password when entering the menu before executing any command.

If one of the options selects a TAG list, the Ctrl/Z keystroke can be emulated with the string *^Z*[3].

Since version 1.4 qualifiers are also supported. With these qualifiers the menu name can also specified as */SUBMENU=menu-name*, the list of options can be specified with the qualifier */JUMPS=options-list*. For more info on qualifiers refer to chapter 2.

To have a look at the possibilities with this Auto Mode, try starting the example menu OMI$EXAMPLE with the following command:

```
$ OMI OMI$EXAMPLE INPUT 4,5,5,3,10,^Z,3,EXIT
```

or

```
$ OMI OMI$EXAMPLE /SUBMENU=INPUT /JUMPS=4,5,5,3,10,^Z,3,EXIT
```

NOTE: When using qualifiers, the first parameter (the menu file), is always required, and has to be entered *before* all qualifiers!

[2] If the menu file, pointed to by the logical *OMI$START_MENU* or the file *OMI$MENU.MNU* needs to be called, the parameter can be specified empty ("")

[3] = 'shift-6' 'Z' on VT keyboards

3.2 Entering values

When an input option is selected, the user is either prompted for a value, or a list is displayed from which values can be selected.

Some input options need to have special formats. This depends on the definition in the menu file. When a wrong value or format is entered, the error message explains what the format should be.

3.2.1 Tag lists

Some input options display a list of possibilities that can be tagged. If a tag option is selected, a list of possible values will be displayed. Entering the corresponding numbers will cause the values to be selected and added to a list.

The last option from a tag list is always to reverse the current selection, so when a tag list is selected for the first time, and all values need to be selected, the last option will do so. After this, the last option will deselect all values.

When all required values are selected, use the <Ctrl/Z> keystroke to return to the menu.

3.2.2 Select lists

If an input option is selected, that can have one value from a predefined list, this list is displayed in a pop-up window. When the list is too long to fit on one screen, the commands *NEXT* and *PREVIOUS* can be used to scroll through the list.

Typing the corresponding number, followed by <Return>, selects that value and returns control to the menu.
The <Ctrl/Z> keystroke cancels the selection.

It is possible to have an option defined in a select list, with which a value can be entered that is not displayed in the list (a 'free' option). If this option is selected, OMI will prompt to enter a value.

NOTE: If the values need to have a special format (e.g. an uppercase string) it must be entered in the correct format, since unlike the usual input option, OMI does not perform any checks or formatting on the free option in a select list! (See also section 5.6.4.)

3.2.3 Entering all inputs in a menu

When a menu contains input options, the text on the display of the last option is the value of *ALL_INPUTS* in the *[QUESTIONS]* section of the configuration file. This option is generated by OMI, and allows the user to enter all inputs at once.

OMI will ask for all required inputs one by one, without the need to select the options separately. When any if the inputs is omitted (the user enters <Return> without a value), the current value will not be overwritten.

If this is disabled for the current menu (see section 5.2.9), this facility can still be used by entering the *ALL* command at the OMI prompt.

3.3 Protected Menus

When a menu is selected that is protected with a password, the Password prompt will appear. When an invalid password is entered, for three times in a row, access will be denied.

When the same menu is selected for three times in a row, and an invalid password has been entered for nine times in total, the menu will be blocked for the current user. An intrusion record is set, that will not disappear until the user logs out from the system.

Chapter 4

Configuration

4.1 Logical Names

When OMI is installed using the *OMI$USER_INSTALL* procedure, the file *OMI$SETUP.COM* is created in the users *SYS$LOGIN:* directory, or in a shared directory (refer to section 1.2). This file contains a few logicals, which can be changed manually.

4.1.1 OMI$

This logical is not defined in the setup file by default, but can be added there. If the logical does not exist when OMI starts, it will be defined automatically, and deassigned when OMI exits.
This logical points to the location of all OMI files.

4.1.2 OMI$MENU_DIRECTORY

When this logical exists, OMI looks in the directory or directories where it points to for all menu files. If a file is not found here, OMI looks in the default directory, pointed to by *OMI$:*.

Menu files should be located in one of those directories. If they can't be found there, an error message will be displayed. This logical can be defined as a search list.

4.1.3 OMI$STARTMENU

If this logical is set, it points to the menu file with which OMI starts every session.
This logical can be user specific or system wide.

4.1.4 OMI$CONFIG

This logical points to the user specific configuration file. It should represent the full path and filename.
If not set, OMI looks for the default configuration file *OMI$MENU.CFG*, first in the users *SYS$LOGIN:*, then in the OMI directory (*OMI$:*).

4.2 Configuration File

A great deal of the behavior (layout in specific) can be modified using a configuration file. This is a file called *OMI$MENU.CFG*, that should be located in the users *SYS$LOGIN:*, or any other file name in another directory, pointed to by the logical *OMI$CONFIG*
If none of these are found, the default file *OMI$:OMI$MENU.CFG* is used.

This section describes the elements that can be modified in the file. Some values can be empty (only when explicitly said so). All element names are case insensitive.

NOTE: Blank lines and comments (!) are allowed. Double quotes (") are NOT allowed.

4.2.1 [MAIN]

This section defines some generally used values.

EMPTY_VALUE In menus where input values are displayed, *EMPTY_VALUE* represents the way an unspecified value will be displayed. If it shouldn't display anything, just remove any value ('*EMPTY_VALUE* = ').

KEY If a key was set using the *SET KEY* command before version 1.3, it will be stored here in the configuration file. The key has a binary code, and should not be set or changed manually.

Since version 1.3, this key cannot be set anymore. Keys are now named and stored in the user specific keyring (see section 4.2.5).
Decryption of strings that were encrypted with this key before version 1.3 is still supported.

SILENT_OUTPUT If DCL commands are executed in silent mode (using the OMI command *SILENT_DCL*), the output, if any, including errors, will be written to the NULL device (*NLA0:*), or any file name specified by this element. The final return status of the command will always be displayed.

TIME_FORMAT This element can have the values 12 or 24. It specifies the default time format for the TIME format type (see section 5.6.2). The default value is 12.

EDITOR When you need to use an editor somewhere in an OMI procedure, this symbol will be used. Specify any (foreign) command.
This setting can be used in OMI modules as:

```
$ 'MAIN'$EDITOR filename
```

FLOAT_POINT Specifies the default character ("." or ",") that will be used for the floating point INTEGER format type (see section 5.6.2).

VERSION_ID This element is obsolete since v1.2.

PROTECT_PROMPT When an own prompt is defined (see section 4.2.3), this can be overwritten by a prompt in a chosen menu. Setting *PROTECT_PROMPT* to *1* or *TRUE*, will tell OMI to use the prompt from the configuration file in all menus.

PRINTER This element can be used to define a user specific printer queue. By default, this will be *SYS$PRINT*. The installation procedure will prompt for a default printer (see section 1.2).

SHOW_PROGRESS By default, OMI display the percentage of the menu file that was read during initialization. This can be switched off by setting this element to *FALSE* or *0*.

Display the percentage read is useful if large menu files are used, since initialization can take pretty long, depending on the system on which OMI runs.

The setting can always be overwritten by specifying the */[NO]PROGRESS* qualifier when starting OMI from the DCL command line (see section 2.2.6).

4.2.2 [SCREEN]

This section contains all elements that define the layout of the menu screen.

WIDTH_MARGIN You can use the full width of the screen (without borderlines at both sides), by setting this margin to *0*. If you do want borders, any value (n) will cause the leftmost and rightmost n columns not to be used.

HEIGHT_MARGIN You can use the full height of the screen by setting this margin to *0*. If you do want smaller windows, any value (n) will cause the uppermost and lowermost n lines not to be used.

WIDTH The symbol *WIDTH* allows two values: 80 and 132. Values up to 80 will be changed to 80. All values greater than 80 will be changed to 132. It defines the screen width of the menu window.

HEIGHT Define the number of lines the menu should use. It is not recommended to change the default value of 24.

EXIT_WIDTH When leaving the menu, *EXIT_WIDTH* is the screenwidth that is set on exit.

WINDOW_TOPMARGIN Specify the number of lines that should be left blank at the top of the window. This is inside the menu window, so is specifies the number of blank lines between the menu header and the first line being used.

SCROLL_REGION You can enable or disable a *SCROLL_REGION*. When enabled, a scroll region is created, leaving the menu screen intact, for the output of DCL commands.

SCROLLREGION_AUTODISABLE Using a scroll region might screw up the menu window when a width margin is used, since it does overwrite the left and right borderlines. By setting *SCROLLREGION_AUTODISABLE* to *1* or *YES*, the use of a scroll region is automatically disabled in this case.

SEPARATE_INPUTS This is a boolean, which can be set to *TRUE* or *FALSE* (or *1* or *0*), that specifies whether or not the menu screen should be split in two separate parts when input options are specified in the current menu, using the lower part of the screen for the input values.

If set to *FALSE*, all items and inputs will be displayed as one long list, which spares the use of two extra lines for separating items and inputs.

DISPLAY_NAMES This element, which can be *TRUE* or *FALSE*, specifies if menu names should be displayed. If set to *TRUE*, the name of the submenus, if they are set, will be displayed in the window, enclosed by brackets.

When set to *FALSE*, the only way to find the names of menus, is using the *SHOW NAME* command when inside the menu.

TAB The *TAB* setting is used when more columns are displayed on the menu screen. It specifies the number of blanks between the two columns.

A second column is used to display the values of input options on the menu screen, and for tag and select lists if they don't fit in one column.

4.2.3 [QUESTIONS]

This section defines the several strings, as they will be displayed whenever OMI or the menu needs input.
These strings can be used to translate all questions in your native language.

ALL_INPUTS When a menu contains input options, the last option will be to enter all values without the need to make the choice for all options separately. This element specifies how this choice is represented on the screen.

REVERSE_TAGS In tag screens, the last option will reverse the current selection. This element specifies how this choice is represented on the screen.

OPTION This element represents the OMI prompt. This can be overwritten by a *PROMPT* element in a menu file, unless the *PROTECT_PROMPT* (see section 4.2.1) is set to *TRUE*.

INPUT Represents how the user will be prompted to enter input when an input option is selected.

DCL_COMMAND Represents how the user will be prompted to enter a DCL command.

DEFAULT_INPUT When input is required for dynamic input (specified by the string ~*?* in the item element—see also section 5.5.1 on page 33), this option specifies the default prompt if none is specified in the item element (this can by done by adding {*prompt*} to the input string).

CONFIRM Several commands ask for a confirmation before executing by default. Setting this element to *FALSE* can change this.

WAIT_PROMPT When OMI waits until the user hits return (or any called procedure using the *OMI$WAIT* command), this represents the prompt.

ANSWER_YES
ANSWER_NO By default, the OMI$CONFIRM command asks the user to enter 'Y(es)' or 'N(o)' by default (see section 6.2.7. If you want your users to be able tp answer in their own language on confirmations, these values can be modified with the values of your choice.
E.g., in Dutch, this would:

```
answer_yes = J
answer_no = N
```

for 'Ja' (Yes) and 'Nee' (no).

4.2.4 [BGRPROCESS]

This section contains all elements that define how background processes should be started.

BATCH_QUEUE Define which queue should be used when OMI modules are started in a batch. Default is *SYS$BATCH*.

DETACHED_LGICMD When starting processes in DETACHED mode, the user's *LOGIN.COM* is not executed. Therefore, several logicals, symbols etc. might not be available.

This element specifies a command procedure which sets up the proper user environment. Default is *SYS$LOGIN:LOGIN.COM*.
The file *OMI$SETUP.COM* will be executed automatically.

LOGFILE Specifies the name of the logfile that will be written by the background process. Default is *SYS$LOGIN:OMI$BACKGROUND_<identifier>.LOG*.

OPTIONS_BAT Used to add extra qualifiers to the *SUBMIT* command, eg:
OPTIONS_BAT = /NOTIFY
For more information, refer to the OpenVMS documentation.

OPTIONS_DET Used to add extra qualifiers to the *RUN /DETACH* command, eg:
OPTIONS_DET = /JOB_TABLE_QUOTA=8192
For more information, refer to the OpenVMS documentation.

4.2.5 [KEYRING]

All encryption keys that are created by the user with the *SET KEY* command, or imported with the *IMPORT KEY* command, are stored in this section. They have a binary value, and cannot be added or modified manually.

4.2.6 Adding sections

When you want to create default settings that can be used in your own menus and modules, you can add your own sections to the configuration file.
E.g., if you want to add a printer queue that can be changed per user for use in OMI, the following section and element can be added:

```
[MYDEFS]
    PRINT_QUEUE = MY_PRINTER
```

If a file needs to be printed on the user specific printer queue, use the command:

```
$ PRINT /QUEUE='MYDEFS$PRINTER_QUEUE' filename
```

NOTE: Several section names are reserved! They are listed in table 4.1.

[MAIN]
[SCREEN]
[QUESTIONS]
[BGRPROCESS]
[KEYRING]
[COUNTER]
[INTERACTIVE_AUTH]
[MENU_*] (all sections starting with *MENU_*)

Table 4.1: Reserved section names

It is possible to add elements to existing sections (e.g. adding the MY_PRINTER element to section *[MAIN]* would result in the symbol MAIN$MY_PRINTER), but this is not recommended, since this might conflict in future releases.

4.2.7 Example Configuration file

Figure 4.1 shows how a configuration file can look like.

```
[main]
   empty_value = .....
   silent_output = NLA0:
   editor = edit/tpu

[screen]
   width_margin = 4
   height_margin = 1
   width = 80
   height = 24
   exit_width = 80
   window_topmargin = 1
   scroll_region = enabled
   scrollregion_autodisable = y
   tab = 8

[questions]
   all_inputs = All Inputs
   reverse_tags = Reverse selection
   option = OMI>
   input = Enter Value
   dcl_command = DCL Command
   default_input = Input
   wait_prompt = Press <Return>to continue
   answer_yes = Y
   answer_no = N

[bgrprocess]
   batch_queue = omi$batch
   logfile = Omi$:Omi$Background_Process.Log
   options_bat = /notify /retain=error

[keyring] !  This section is created by OMI
   my_key = ÒÒÒÒ[1]
   group_key = ÒçÒÒÒ[1]
   system_key = ÒíÒÒÒ[1]

!
!  Below are my personal additions
!
[mydefs]
   print_queue = my_printer
   data_location = device:[data_dir]
```

Figure 4.1: the OMI configuration file

Chapter 5

Creating Menus

The Menu File contains all specifications and actions of the menu structures. The files are built with sections, specified as a name between square brackets (*[section_name]*).

Any section name can be chosen. The elements that are specified in the section, are defined as symbols, and are available during the OMI session.
E.g. if the following section is defined:

```
[MY_SECTION]
    A_STRING = My String
    AN_INTEGER = 2
```

the symbols MY_SECTION$A_STRING (with value "My String") and MY_SECTION$AN_INTEGER (with value 2) are available in OMI.

These sections are used for defining and configuring the menu structure. For special purposes, the section names *COUNTER*, *MAIN*, *SCREEN* and *QUESTIONS*, and all section names starting with *MENU_* are reserved (see also table 4.1 on page 22).

5.1 Defining Menus

The name of a menu section always starts with *MENU_*, and the main menu has to be named *MENU*, so a section *[MENU_MENU]* is required.
All information, and all elements, of the menu are specified inside a menu section.

5.1.1 The Main Menu

As stated above, every menu file has to contain a section called *[MENU_MENU]*. This is the main menu of the menu file.
All elements that will be described below, can be specified in the main menu.

5.2 Menu Characteristics

Each menu can have unique characteristics, which can make the navigation easier. These elements are all optional. Every menu can have it's own prompt,

title, name and comment line. Title and comment lines will be displayed by default.

5.2.1 The ON_INIT and ON_EXIT elements

When the user starts enters a menu (by starting OMI or by entering a sublevel), OMI will execute an OMI module if the menu contains the *ON_INIT* element. This will only be done if the user comes from a higher level, *not* when a coming up from a sublevel.
If the module returns status *OMI$_WARNING* (described in chapter 6), control will not be passed to the selected menu level.

If a menu contains the *ON_EXIT* element, the specified exit module will be executed when the user leaves that menu. This will only be done if the user leaves a menu to go to a higher level, *not* when a submenu is chosen!
If the exit module returns status *OMI$_WARNING* (described in chapter 6), control will be returned to the current menu.

If a submenu is left with any of the commands *MENU*, *EXIT* or *QUIT*, OMI will check both the current submenu and the main menu for the *ON_EXIT* element.
The default type of init- and exit modules is *.OMI*. Figure 5.1 shows the use of init- and exit modules.

Example of the Menu file

```
[MENU_MENU]
   ITEM1 = Exit the menu#COMMAND#EXIT
   ITEM2 = Start another menu#MENU#OTHER_MENU
   ON_INIT = START_WITH ! This module will be executed when OMI starts
   ON_EXIT = EXIT_WITH ! This module will be executed when OMI exits
```

Example of EXIT_WITH.OMI:

```
$ OMI$CONFIRM "Are you sure you want to exit ?  " -
   'QUESTIONS$ANSWER_NO' $ IF .NOT. OMI$CONFIRMED THEN $ EXIT
OMI$_WARNING
$ EXIT OMI$_OK
```

This example shows how you can ask the user for a confirmation before actually leaving OMI.

Figure 5.1: using initialising- and exit modules in a menu

Execute OMI commands with ON_INIT and ON_EXIT

Since version 2.2 it is also possible to execute OMI commands in the *ON_INIT* and *ON_EXIT* elements.

To do so, the element value should start a colon (":"), immediatly followed my the OMI command. The command will be executed, then control is passed to the selected menu. Figure 5.2 shows an example where all variables in the menu are reset to their default values (using the command *RESET VARIABLES*, see section 9.20.6) when the user leaves the menu.

The return value of the OMI command is not evaluated.

```
[MENU_ASKINPUT]
   ON_EXIT = :RESET VARIABLES BACKGROUND
   ITEM1 = Back to the main menu#COMMAND#MAIN
   ITEM2 = Start another menu#MENU#OTHER_MENU
   INPUT1 = Select a day#{SEL|WEEKDAYS}wk_day#VALUE1
   INPUT2 = Enter a time#tme##FRM_TIME
```

the variable name *wk_day* will be reset to it's default value listed as *VALUE1* in the *WEEKDAYS* select list. The variable *tme* is removed from memory (if it was set).

Figure 5.2: using initialising- and exit commands in a menu

5.2.2 The PROMPT element

The prompt element defines the prompt that will show up in the menu. Every submenu can have its own prompt. If none is specified, the prompt from the top-level menu is used. If no prompt element is found at all, the prompt from the configuration file (element *OPTION* from section *[QUESTIONS]*) is used. This last prompt will always be used if it is protected (see section 4.2.1).

5.2.3 The NAME element

The name will only be displayed when the element *DISPLAY_NAMES* in the configuration file is set to *TRUE*, and with the *SHOW NAME* command on the OMI prompt.

The name is used for the *JUMP* command (see section 9.15), and in auto mode (see section 3.1.2).

5.2.4 The COMMENT element

Comment lines can contain variables. They should be enclosed by accolades. The comment line is displayed on the first line of the menu screen. If the comment element is omitted, the menu screen will only display the specified elements.

5.2.5 The TITLE element

The title will be displayed in the title bar of the menu. If omitted, a default title will be displayed.

5.2.6 The AUTO_REFRESH element

When this element is specified, it gives the number of seconds that OMI waits for input on the OMI prompt before the menu screen is automatically refreshed. The value should be between *0* and *255*. When this element is omitted, or has value *0*, the automatic screen refresh is disabled.

5.2.7 The AUTO_INCREASE element

If a submenu uses a counter from another (higher level) menu, and default values are used in array- variables, the counter has to be specified in the menu. To prevent the counter from being increased automatically, set the *AUTO_INCREASE* value in the submenu to *FALSE*. If this element is set to *TRUE* (the default value), the counter value will be increased each time the menu is chosen.

5.2.8 The REQUIRED_ORDER element

The user can be forced to enter the input values in a predefined order. This can be done with the *REQUIRED_ORDER* element. Specify the list of values in the proper order, separated by comma's, as the value for this element.

When the user selects an input element that is found in the *REQUIRED_ORDER* element, but not as the first one, a warning message is displayed, telling the user to select another input element first.

If the selected element is the first one required, it is removed from the list and the user can enter the input. When the list is completely empty (all required inputs have been entered in the correct order), no checks will be performed until the *RESET ORDER* (see section 9.20.4) is issued from the menu.

Using the *REQUIRED_ORDER* element will set the *ALL_INPUTS* element (see section 5.2.9) to *FALSE*.

```
[MENU_INPUTS]
   REQUIRED_ORDER = 1,2,3,5
   INPUT1 = Directory#{SEL|DIRLIST}DIRECT
   INPUT2 = Filename#{SEL|FN_'DIRECT'LIST}FNAME
   INPUT3 = Filetype#{SEL|FTYPELIST}FTYPE
   INPUT4 = Comment#COMMENTLINE
   INPUT5 = Action#{SEL|ACTIONLIST}ACTION
```

This example forces the user to enter input element 1 as the first one. With input 1, a value is selected from a select list (select lists are described in section 5.6.4), which is required to determine from which select list input 2 has to be selected.

Figure 5.3: Force the order in which input is entered

In figure 5.3, input 5 cannot be selected when the inputs 1, 2 and 3 have not been selected yet. Input 4 is optional.

5.2.9 The ALL_INPUTS element

By default, OMI will display an option that can be selected to enter all input elements without selecting them separately (see also section 3.2.3). Adding the *ALL_INPUTS* element to a menu section with value *0* or *FALSE* can disable this.

This element will be disabled by default when the *REQUIRED_ORDER* element (see section 5.2.8) is specified for this menu.

Example of MYMENU.MNU:

```
[MENU_MYMENU]
   TITLE = This is the title of My Menu
   NAME = my_menu !  This element can not contain blanks!
   COMMENT = This is an example, username is {OMI$CURRENT_USER}
```

This example shows the use of menu characteristics. The symbol *OMI$CURRENT_USER* will be translated to the current username, and displayed on the comment line. This symbol is described in section 6.3.1.
The name can be used to let OMI start with MYMENU (specified by the section name [MENU_MYMENU], in stead of the main menu *[MENU_MENU]*, by entering

```
$ OMI MYMENU MY_MENU
```

on the DCL prompt.
Note that the first parameter ("MYMENU", which points to the file- name) is required, even if the logical *OMI$START_MENU* points to that file, when the menu- name is used as a parameter! (See also section 3.1.2)

Figure 5.4: Setting menu characteristics

5.3 Counters

A special menu element is the counter. This defines a variable that will be increased every time the menu is called (unless the *AUTO_INCREASE* item is set to *FALSE*, as described in section 5.2.7). The use of counters requires a section *[COUNTER]*, which contains all counters.

Using this structure, it is possible to define arrays, as in figure 5.5.

5.3.1 Manipulating counter values

Counter values can also be manipulated using commands on the OMI prompt. If any of these commands are used, the comment line in figure 5.5 will not be updated until the screen is refreshed with the *REFRESH* command, except for *INCREASE* command, with which the *REFRESH* parameter can optionally be specified.
The *INCREASE* command increases the current counter value with one.

If the value needs to be reset to 0[1], use the command
RESET COUNTER.

The command *SET COUNTER*, followed by an integer value, sets the counter to the specified value.

For more information about this and other OMI commands, refer to the chapter 9.

[1]NOT the initial value as specified in the *[COUNTER]* section!

```
[MENU_TEST]
   COMMENT = The counter is now {COUNTER$TEST_COUNTER}
   ITEM1 = Display the array#CALL#DISPLAY_ARRAY
   INPUT1 = Enter any value#MY_VAR'COUNTER'$TEST_COUNTER‘
   COUNTER = TEST_COUNTER

[MENU_TEST2]
   .
   .
   .
   COUNTER = TEST_COUNTER

[COUNTER]
   TEST_COUNTER = 0
```

The TEST menu above has a counter that will be increased every time the menu is called. After every call, the user can enter a value by selecting the option 'Enter any value' on the menu screen, which will be stored in the variable with the name MY_VARn, where 'n' is the current value of the counter.
The *[COUNTER]* section is required to initialize all counters that are used in the menu with any value. Most likely, the initial value will be 0, but there's no default, so a value has to be specified!
In the example, a second menu, TEST2, uses the same counter. This means, the counter is increased every time *any* of the two menus is called! Note that one (sub)menu can have only one counter. The total number of counters in the menu file is unlimited.
The current value of the counter can be displayed in the comment line, as in the example, or with the *SHOW COUNTER* command.
The option "Display the array" calls an example OMI module, DISPLAY_ARRAY.OMI, which could be created to display all entered values.

Figure 5.5: Using counters

5.3.2 Using default values in arrays

If a submenu uses a counter from another (higher level) menu, and default values are used in array- variables, the counter has to be specified in the menu.

To prevent the counter from being increased automatically, set the *AUTO_INCREASE*, value in the submenu to *FALSE*. If this element is set to *TRUE* or omitted, the counter value will be increased each time the menu is chosen.

5.4 Security

Every menu can be protected using security elements and passwords. Besides this, every submenu can also have an owner.

None of these elements are required, and there's no default.

5.4.1 The OWNER element

The owner can be either a single username, or a list of usernames separated by commas.

If an owner is specified for the main menu, this user will be considered the owner for the complete menu file (all submenus), except for those submenus that have a different owner element.

If no *SECURITY* element is specified, all users have *READ* and *EXEC* privilege on all menus, except the owner, who has *READ*, *EXEC* and *WRITE*.

5.4.2 The SECURITY element

The *SECURITY* element should point to a section in the menu file with the security settings for the menu.

The security section contains a list of usernames with their privileges on the selected menu. The use of this section overwrites the default highest level for the menu owner.

By default, users that are not listed, have no access to the menu. This can be overwritten with the *ALL_USERS* element.
Valid security levels are listed in table 5.1.

NONE	The specified user has no access to the menu
READ	The user can enter the menu and see the choices, but cannot execute any of the options, or any command at the command line. The only way to leave the menu is by using <Ctrl/Z>.
EXEC	The user has full access to all menu elements for execution
WRITE	The user can make modifications to the menu, using the *SET* or *RESET* command

Table 5.1: Security levels in Security sections

When no security section is used, *READ, EXEC* is the default for all users except for the owner, who'll also have *WRITE* access.

5.4.3 The PASSWORD element

The *PASSWORD* element holds the password in encrypted format. It cannot be modified using an editor; a password can only be set or changed with the *SET PASSWORD* command from the OMI prompt.

If it is required to add a password using the editor anyway, e.g. while creating a menu file, this can be done by specifying the password element with an equals-sign, but without a value:

```
PASSWORD =
```

The password will default to *Omi$System* (case sensitive!). This can be used as a reminder to change the password interactive.

The optional *PASSWORD_LEVEL* element defines the security level for all users when a password is required. This is valid for all users, including the menu-owner.

If this element is not specified, the default security handling is used.

The value should be specified as an integer. Table 5.2 provides an overview with the valid values and their corresponding security level.

0	NONE
1	READ
2	EXEC
3	WRITE

Table 5.2: Security levels in the PASSWORD_LEVEL element

5.4.4 Interactive authorization

Users can be blocked from using interactive DCL with the *SPAWN*, *DCL* or *SILENT_DCL* command by including the section *[INTERACTIVE_AUTH]* in the menu file. This section can contain user names that are, or are not authorized to use interactive DCL.

By default, all users can use DCL. Users that should be blocked can be included in this section, with value *FALSE*.

The default can by overwritten, by including the *ALL_USERS* with value *FALSE*. All users are now disabled for interactive DCL, except those that are explicitly included with value *TRUE*.

Note that this does not affect the *(SILENT_)DCL* and *SPAWN* commands that are called by menu items; they can always be executed by all users that have *EXEC* privilege to the submenu that contains the items.

5.5 The ITEM Element

Items are used to define the menu structure, and the actual actions being taken. The basic layout of an item element has the keyword ITEM, followed by an integer, and a value consisting of three arguments, separated by a hash (#):

ITEMn = Text on display#item type#value for type

The keyword *ITEM* must be followed by an integer, starting with '1', and without gaps in the counting, so if a menu has three item options, the elements have to be specified as *ITEM1*, *ITEM2* and *ITEM3*.

The value of items consists of three arguments, separated by a hash (#). The first argument represents how the option will be displayed on the window. The second argument specifies the item type, which can be *CALL*, *COMMAND* or *SUBMENU*. The meaning of the third argument depends on the item type.

5.5.1 The CALL item type

The *CALL* item type specifies that an OMI module should be executed. The third argument of the item value should be the full path and name of a DCL procedure, without the at-sign (@). The default file type for OMI modules is OMI.

NOTE: When an OMI module is executed, all error, warning and informational messages generated by VMS are suppressed. If you don't want this (e.g. while creating and testing menus or modules), use the command

```
$ SET MESSAGE 'OMI$_MESSAGE'
```

in the top of the procedure, or start OMI with the qualifier */DEBUG*.

For a further description of OMI modules, refer to chapter 6.

Dynamic input If the procedure should be called with parameters, they can be specified in the third argument. These parameters, or a part of it, can be dynamic. This means, the user is prompted to enter a value after selecting the menu option, but before the module is actually executed, as in figure 5.8.

Dynamic input can also be validated and formatted using a format section. This requires a prompt enclosed by accolades. After the prompt (still between the accolades), the name of the format section to use can be specified. The prompt and section name are separated by a pipe-sign (|). This is shown in figure 5.9. For more information about formatting sections, refer to section 5.6.2.

Every item can have an unlimited number of dynamic inputs.

5.5.2 The COMMAND item type

The *COMMAND* item type is used to execute an OMI command. The command that should be executed is specified in the third argument of the item value, including all keywords and/or parameters. As with the *CALL* item type, a command can also have dynamic input, as can be seen in figure 5.9.

For a further description of OMI modules, refer to chapter 6.

5.5.3 The SUBMENU item type

This item type points to a submenu. The third argument has to contain the name of the menu section (without the *MENU_* string!).

You can also specify dynamic menus. This is done in the third argument. In stead of *my_menu_name*, it should contain the value
{*Text*|*submenu*}{*Text*|*submenu*}*prompt*, as in

```
ITEM1 = Choose a submenu#SUBMENU#{First submenu|MYMENU1} -
    {Second submenu|MYNAME2}Select a menu
```

You can have as many menus in dynamic menu lists as you like, but take care the total length of the record in the MNU file, including leading blanks and the *ITEMn* = , is less than 256 characters (instead of the VMS default of 512)!

Also, if the list is too long, displaying can screw up the window screen. This has no effect on the functionality, but the user will have to enter the REFRESH command after selecting a menu.

Figure 5.10 illustrates the use of the SUBMENU element.

5.6 The INPUT element

Inputs can be used to collect information from the user, which can be passed to OMI modules. The keyword INPUT has to be followed by an integer, starting with '1', and without gaps in the counting, so if a menu has three input options, the elements have to be specified as INPUT1, INPUT2 and INPUT3.

The value of inputs consists of two to four arguments, separated by a hash (#).

The first argument represents how the option will be displayed on the window.

The second argument contains the variable name (the symbol that is defined after the user selected the corresponding option). This also indicates the input type, which can be 'Free input', 'Select list' or 'Tag list'.

The third argument can optionally contain a default value. The way this is handled depends on the input type. This argument is not valid in Tag lists.

The fourth argument is only valid for the Free input. It will be described below, and more specific in section 5.6.2.

5.6.1 The Free Input element

The Free input type is an element that prompts the user for input (using the prompt represented by INPUT from the [QUESTIONS] section in the configuration file) when selected. The basic layout is as follows:

INPUTn = Text on display#VARIABLE_NAME#default value#format

in which the third argument, 'default value', and the fourth argument, 'format' (described below), are optional.

The default value will be initialized when OMI starts, so if the value is changed, and another submenu is chosen, the new value will not be overwritten when the same menu is called again.

If a FORMAT argument is used, without a default value, the default value has to be specified empty, as in

INPUTn = Text on display#variable name##format

5.6.2 Formatting the input

If the Free input element contains a fourth argument, this should be the name of a format section. This section contains a description of what kind of input is expected, and/or how the input should be formatted.

The keyword TYPE is required in the format section. This can be STRING, FILESPEC, INTEGER, DATE or TIME.

All booleans that can be used in the format sections described below, will be FALSE by default if omitted.

The keywords will either be used for validating the user response, which will return a warning message when validation fails (e.g. the WILDCARDS keyword

in the FILESPEC type), or for converting the user input (e.g. the UPCASE keyword in the STRING type). This is described per keyword.

Section 5.6.3 shows an example of the use format sections.

The FILESPEC type This indicates the input should be a filename, with or without a device and directory specification.

If the input contains blanks, they will be removed, and the file name will automatically be converted to uppercase.
Valid keywords are listed in table 5.3.

REQUIRED	[TRUE or FALSE]	If set to *TRUE*, OMI checks to see if the file exists. If not, an error is displayed.
WILDCARDS	[TRUE or FALSE]	If set to *TRUE*, OMI will allow wildcards ('*' and '%') in the file name.
FDEVICE	[*device name*]	This can specify a default device name.
FDIRECORY	[*directory name*]	This can specify a default directory name.
FTYPE	[*file type*]	This can specify a default file type.

Table 5.3: Keywords for the FILESPEC format type

The STRING type This indicates the input should be of the STRING type. An invalid response will result in a warning message.
Valid keywords are listed in table 5.4.

The INTEGER type This indicates the input should be of the *INTEGER* type. An invalid response will result in a warning message.
Valid keywords are listed in table 5.5.

The DATE type This indicates the input should be of the *DATE* type. An invalid response will result in a warning message. The user response has to be in a valid *ABSOLUTE* time format (DD-MMM-YYYY).
Valid keywords are listed in table 5.6.

The TEXTAREA type Enter free input, using a text editor. The value is both stored in a file, and in the specified variable name. The size of the value in the variable name is limited by VMS. This does not affect the value stored in the file.

When a valid filename is entered, the editor, specified in the configuration file (see section 4.2.1), will be started. When the no filename is entered, a default value will be used. This is the default value from the menu file or a newly generated value if omitted.
The default location is *OMI$MENU_DIRECTORY:*[2].

[2]Note that the *OMI$MENU_DIRECTORY:* logical can be a search path; the first physical directory in the seach list is used by default.

UPCASE	[TRUE or FALSE]	If set to *TRUE*, the input string will be converted to uppercase.
LOWERCASE	[TRUE or FALSE]	If set to *TRUE*, the input string will be converted to lowercase.
COLLAPSE	[TRUE or FALSE]	If set to *TRUE*, all blanks and tabs will be removed from the input string[a]
MINLENGTH	[*integer value*]	Minimum number of characters.
MAXLENGTH	[*integer value*]	Maximum number of characters.
IVCHARS	[*list of invalid characters*]	Specify a list of characters that are not allowed in the input string. If any if the listed characters are encountered, a warning message will be displayed[b].
ALPHANUM	[TRUE or FALSE]	If set to *TRUE*, the keyword *IVCHARS* will be filled automatically with the character list: `‘’~^!?@#$%&* -+=(){}[]<>:;,.|/`[c]. This overwrites any other value of *IVCHARS*!

Table 5.4: Keywords for the STRING format type

[a] In versions prior to 2.3, the keyword *BLANKS* was used for this format, but this was very confusing, since *BLANKS = TRUE* would remove the blanks—all defaults are FALSE with the formatting keywords. Therefore, *BLANKS* has been replaced by *COLLAPSE*. The *BLANKS* keyword is still supported, but will be overwritten by *COLLAPSE* if both are used.

[b] This list can NOT contain an exclamation mark (!)or a double quote (”). Also, if the list should contain a blank, it cannot be the first or the last character of the list, blanks are allowed, but have to be enclosed by other characters

[c] Unlike the use of *IVCHARS* as a keyword directly, the exclamation mark (!) will be added to the list of invalid characters this way, but the double quote (”) still not. The *ALFANUM* keyword makes sure the formatted string only accepts all characters (‘A’-‘Z’), numbers (0-9) and the underline (_) as valid input.

The file will only be available during the current OMI session, and the file will be deleted when OMI exits, unless *KEEP* is set to *TRUE*. This and other valid keywords are listed in tabel 5.7.

The TIME element This indicates the input should be of the *TIME* type. An invalid response will result in a warning message. The user response has to be a valid time format (*HH:MM [AM|PM]*or *HH.MM [AM|PM]*).
Valid keywords are listed in table 5.8.

5.6.3 Example of the Free input type with Formatting.

In the INPUTS menu in figure 5.11, the user can enter two values; a filename, which has to exist (a warning will be displayed if the file is not found), and a date, which will be the current date by default.

The filename will be converted to uppercase and all blanks will be removed before other validation is performed.

MIN	[*integer value*]	The lowest value allowed.
MAX	[*integer value*]	The highest value allowed.
FLOAT	[TRUE or FALSE]	Specifies the input value can be a floating point[a].
FLOAT_POINT	[. or ,]	Specifies which character will be used as the floating-point character. It overwrites the default value from the configuration file (see section 4.2.1)[b].

Table 5.5: Keywords for the INTEGER format type

[a]When an integer can have a floating point, the *MIN* and *MAX* values will still be regular integers; they can never have floating point values.

[b]Input entered by the user, must contain the same floating-point character!

FORMAT	[ABSOLUTE, COMPARISON or DELTA]	Specify the format to which the input will be converted.

Table 5.6: Keywords for the DATE format type

The date will be converted to the VMS COMPARISON type, so if the date entered is "25-JUN-1997", this will be changed to "1997-06-25".

5.6.4 The Select list input element

If the user cannot freely enter any input, a list with choices can be presented, from which the user can make a selection.

To do so, the variable name in the second argument is preceded with the *SEL* keyword, followed by the pipe sign (|) and the name of a section with choices, enclosed by accolades:

INPUTn = Text on display#{SEL|section_name}variable_name#VALUEn

The optional third argument points to the default value. This should be the locator of the value (*VALUEn*, see below), *not* the value itself. This input type accepts no fourth argument.

The select section contains a list of variables, called *VALUE1...VALUEn*, that will be displayed in a pop-up window.

The name of a selectlist section can be freely chosen, but it must be the same as in the input element where it's being called from (the name after the pipe sign between accolades).

The selectlist section only contains numbered VALUE elements, like

VALUE1 = any value
VALUE2 = any value
VALUE3 = any value

One of the options can be enclosed by accolades. This means, the user does not have to select from the list, but can also enter free input. It is recommended to use the last option for this purpose[3].

[3]If this possibility is used, it won't display very nicely if the same list is used as a taglist

FILENAME	[textitfile name]	The name of the temporary file that will be used to create the text area. The default location is *OMI$MENU_DIRECTORY:* (see footnote 2 on page 35), and the default type is *.TXT*. If this field is not specified, the name *TA_menu$INPUTn* will be used, where *menu* is the current menu name, and *INPUTn* the OMI internal element name.
KEEP	[TRUE or FALSE]	If set to *TRUE*, the temporary file will not be removed when OMI exists, and so will be available for a future session.
HISTORY	[TRUE or FALSE]	If set to *TRUE*, older version of the temporary file will also be saved, limited by the VMS VERSION_LIMIT. When *KEEP* is set to *FALSE* or not specified, *HISTORY* is automatically set to *FALSE*. By default, only the last version will be saved (if *KEEP* is *TRUE*)
LARGE	[TRUE or FALSE]	The size of a textarea is limited to 255 characters by default. With this option it is possible to store values up to 1024 characters in the variable[a]. The maximum size does not affect the value that is stored in the temporary file.

Table 5.7: Keywords for the TEXTAREA format type

[a]Setting this option to *TRUE*, can cause the command *SHOW TEXTAREA* not to show anything without a warning!

NOTE: Free input in a select list, like option 14 in figure 5.12, can NOT be formatted or validated! So, if an OMI module is called with the variable USRNAME from the USER menu in the example, which requires the username to be in uppercase, lowercase user input will cause failures!

When this input is asked using the *ALL* command, or the last input option from the menu (see section 3.2.3, the current value is highlighted in the pop-up window, and won't be overwritten if no new value is selected.

5.6.5 The Tag list input element

If a variable needs to contain a list of values, that can be searched in an OMI module using the F$ELEMENT lexical, tag lists can be used as an input element.

somewhere else (refer to section 5.6.5)

HOURS	[12 or 24]	Specifies if the time should be in 12 or 24 hours format. In 12 hours format, the string "am" or "pm" will be added, and the time value will, if necessary, be converted (e.g. "21:45" will be converted to "9:45pm", "10:15" will be converted to '10:15am"). When omitted, the default from the configuration file will be used (see section 4.2.1).
SEPARATOR	[*separator string*]	By default, a colon (:) will be used to separate hours and minutes (HH:MM), but by using this element, any character string can be diaplayed in the output. For user input, only a colon (:) or a dot (.) are allowed.
TRZERO	[TRUE or FALSE]	If set to *TRUE*, trailing zeros will be included for the hours if necessary, e.g. "9:45pm" will be displayed as "09:45pm".
UPCASE	[TRUE or FALSE]	If set to *TRUE*, the string am or pm will be converted to uppercase. This element will be ignored when *HOURS* is specified as *24*.

Table 5.8: Keywords for the TIME format type

This is done by preceding the variable name by keyword *TAG*, followed by a pipe sign (|) and the name of a tag list, enclosed by accolades:

INPUTn = Text on display#{TAG|taglist_name}variable_name

When selected, all possible values will be displayed on the menu screen, and the user can select all desired values, which will be highlighted. The last option in the tag list will always be to reverse the current selection.

Pressing <Ctrl/Z> will return to the menu screen. By then, all values have been stored in the named variable, separated by a specified delimiter.

The tag list input element cannot have a default value (a third and fourth argument will be ignored).

The name of a taglist section can be freely chosen, but it must be the same as in the input element(s) from where it's being called.

The layout of the tag list section is the same as the select list section, with the following differences:

- The *DELIMITER* keyword is required. This must me a single character, which will be used to separate the selected items.
- Using the optional keyword *MESSAGE*, a message can be displayed on the top of the screen to give the user a small explanation of what's expected.
- A 'Free input', as option 14 in figure 5.12, cannot be used.

- Due to the same layout, Tag lists can be used as Select lists and vice versa (as long as no 'Free input' is required for the Select list), since the *DELIMITER* and *MESSAGE* keywords are ignored by the Select input.

Figure 5.13 shows how a tag list element can be created, with which the user can tag one ore more weekdays.

If this option is selected, the list of weekdays will be presented in the menu window with all days preceded by option numbers 1-7, and option 8 to reverse the current selection.

If the user selects the numbers 1, 3 and 5 (one at a time), the value of the variable DAYLIST will be "Monday/Wednesday/Friday".

5.6.6 Reading Tag and Select values from files

It is possible to let OMI read the values for a tag list or a select list from a file. To do so, an input element and a named section should be created as described in the previous sections but in stead of the *VALUEn* elements, the section should contain the *FILENAME* element, which points to the filename that contains the values. All other elements can still be used in the same way.

The default location of the file is *OMI$MENU_DIRECTORY:*, or *OMI$:* if not found there.

Figure 5.14 shows the use for input files in tag- or select lists. The *VALUEn* fields are filled dynamically with values from the specified file. The file in this example changes every month, and contains all dates for the weekends in the selected month.

In stead of reading the variable from F$TIME(), it's also possible to use another input element to get the variable pointing to the requested value file.

If you have write privilege to the current menu, the value file can be modified with the *EDIT VALUE_FILE* command.

5.7 Including Standard Libraries

An *#INCLUDE* directive can be used to load one or more library menus into the menu file. Library menus are normal OMI menus, but they should not have a main menu (section *[MENU_MENU]*). All sections from the library menu are available in the menu that includes the library. The default file type for libraries is *.OML*.

The *#INCLUDE* directive can be used anywhere in the menu file. The number of includes is unlimited, but nested includes are not allowed (the *#INCLUDE* directive cannot be used from within libraries).

NOTE: OMI comes with one standard library, *OMI$LIBRARY.OML*. It is advised not to edit this library, but to create your own ones, since the library might be overwritten with a new distribution.

5.7.1 Handling duplicate items

When including one or more standard libraries, it can happen that one of the libraries contains a section with values that is used in the calling menu file as well.

To prevent warning messages, the attribute *#LEADING* can be added to the item that should be used.

The order in which the files are read is not relevant[4]; the item that has been specified with *#LEADING*, will be the value that is used. If all values are specified with *#LEADING*, the last one will be used.

Figure 5.15 shows how libraries can be used and warning messages for duplicate values can be prevented.

5.8 Adding descriptions to the Menu file

Menu files can contain descriptions, if they are outcommented using the exclamation mark (!). However, if a line starts with an exclamation mark, it's still read by OMI. To improve performance with long descriptions, they can be placed at the bottom of the file, after the string *<EOF>*.

Whenever OMI reads a line with (only!) this identifier string, it will consider it as the end-of-file, and will stop reading from the file. Figure 5.16. shows an example of this. The same identifier can also be used in configuration files.

5.9 Special Characters

This section briefly lists characters that are reserved by OMI in menu files.

The exclamation mark (*!*), double quote (") and accolades ({}) cannot be used for elements or their values.
An exclamation mark should be used for comments.

The hash (#), the pipe-sign (|) and the combination of tilde and question mark (*~?*) cannot be used in input- or item- elements.

5.10 Creating menu- specific help files

Each menu file can have a help file. This file can be used for brief usage information or other descriptions.

Help files have to be located in the same directory where the menu is found, and should have the same name but with file type `.OMH`, e.g. if your menu file is called `MY-MENU.MNU`, the help file should be called `MY-MENU.OMH`.

The layout of the menu file is similar to the menu file. There is a section name between square brackets, which should be equal the menu name without the `MENU_` prefix. For the main menu—who's section in the menu file always has to be `[MENU_MENU]`—this means the help section is called `[MENU]`, and if a menu is created in the menu file with section name `[MENU_MY-FUNNY-SUBMENU]`, its helpsection is named `[MY-FUNNY-SUBMENU]`.

[4]When a menu file is validated with the */VALIDATE* qualifier (see section 2.2.8), a warning will be displayed if the first item of two duplicate item names was specified as leading. During run-time this is handled properly.

The text in the section is displayed on the screen as entered in the file. As with long file descriptions, the *<EOF>* identifier can also be used to force the end of file (see also section 5.8). Additionally, the textstring *<FF>* can be used to force a newpage. By default, the user has to press `Return` when the last line in the OMI screen is written, but *<FF>* forces an `OMI$WAIT`[5] at the line where it occured.

Helpfiles are read with the `INFO` command. For more information on this command, refer to section 9.14.

The menu `OMI$MANAGE`, which comes with the distribution, has a help file that can be used as an example.

[5]see section 6.2.25

```
[MENU_MENU]
   OWNER = SYSTEM
   SECURITY = TOPLEVEL_SECURITY
   ITEM1 = First submenu#SUBMENU#FIRST
   ITEM2 = Second submenu#SUBMENU#SECOND

[MENU_FIRST]
   OWNER = FIELD
   PASSWORD =

[MENU_SECOND]
   SECURITY = SECONDLEVEL_SECURITY

[TOPLEVEL_SECURITY]
   SYSTEM = READ,WRITE,EXEC
   FIELD = READ,EXEC
   ALL_USERS = NONE ! Default when the SECURITY element is used

[SECONDLEVEL_SECURITY]
   FIELD = READ
   ALL_USERS = READ,EXEC
```

The owner of the main menu, and of the menu SECOND, is user SYSTEM. In the menu FIRST, the owner is FIELD. This menu is also protected with a password[a].
The *SECURITY* element defines the privileges for all listed users. If a user is not listed, the privileges from the *ALL_USERS* will be used, or the value *NONE* if *ALL_USERS* is not specified.
Note that user SYSTEM is granted *WRITE* on the main menu, even though it's the owner. This is necessary since the *SECURITY* element is used.
In the menu FIRST, all users have *READ,EXEC* privilege, except user FIELD (the owner for that menu), who has *READ,WRITE,EXEC*, since that menu uses no *SECURITY* element.

Figure 5.6: Defining menu security

[a] which is empty in this example, so it defaults to *Omi$System*, but usually this will contain an encrypted value

```
[INTERACTIVE_AUTH]
   SYSTEM = TRUE
   SYSTEST = TRUE
   DEVELOPER = TRUE
   ALL_USERS = FALSE ! Default is TRUE

[MENU_MENU]
   ITEM1 = Exit#COMMAND#DCL LOGOUT
   ITEM2 = Mail#COMMAND#SPAWN MAIL
```

When the menu file contains this section, the users SYSTEM, SYSTEST and DEVELOPER can use the commands *(SILENT_)DCL* and *SPAWN* at the OMI command line, all other users can't.
All users however can execute the items that are given in the main menu in the example.

Figure 5.7: Disallow users to use interactive DCL

Extract from a menu file:
ITEM1 = Display the weekday#CALL#DISPLAY WEEKDAY ~?{Enter a date}
Example of DISPLAY.OMI:

```
$ IF P1 .EQS. "WEEKDAY" THEN $ GOTO WEEKDAY$
.
.
.
$WEEKDAY$:
$ OMI$DISPLAY_MESSAGE "Weekday for ''p2':  ",
F$CVTIME(P2,,"WEEKDAY")
$ EXIT
```

In this example, the user gets the prompt "Enter a date" when the CALL item type is selected. After entering a date, it will be passed to the module as the third parameter.
The prompt is enclosed by accolades in the item element. If omitted, the prompt specified as *DEFAULT_INPUT* in the *[QUESTIONS]* section of the configuration file is used.

Figure 5.8: Using dynamic input in the CALL item type

```
   ITEM1 = Type a text file#COMMAND# DCL TYPE /PAGE -
      ~?{Filename:|FILE_FORMAT}

[FILE_FORMAT]
   TYPE = FILESPEC
   REQUIRED = TRUE
   FTYPE = .TXT
```

Figure 5.9: Using dynamic input with a format section

```
[MENU_MENU]
   ITEM1 = Go to the second Menu#SUBMENU#SECOND_MENU

[MENU_SECOND_MENU]
   ITEM1 = Dynamic Menu#SUBMENU#{Menu 1|SUB1} -
      {Menu 2|SUB2}Which Menu:

[MENU_SUB1]
   !  Menu definitions...

[MENU_SUB2]
   !  Menu definitions...
```

OMI starts with a menu in which an option is displayed that is represented as "Go to the second Menu". When the user chooses this option, the next menu is displayed, containing the option "Dynamic Menu"'. Choosing this option, results in the following 'pop-up' window:

1> Menu 1
2> Menu 2

and on the command line, the prompt "Which Menu:". The menu that will be called depends on the user input.

Figure 5.10: Using the SUBMENU element

```
[MENU_INPUTS]
   INPUT1 = Enter a file name#FNAME##FILNAM_FRM ! No default value
   INPUT2 = Enter a date#DATE#TODAY#DATE_FRM ! Default is Today

[FILNAM_FRM]
   TYPE = FILESPEC
   REQUIRED = TRUE ! File has to exist

[DATE_FRM]
   TYPE = DATE
   FORMAT = COMPARISON
```

Figure 5.11: Formatting Free input

```
[MENU_USERS]
   INPUT1 = Select a user#{SEL|SYSTEM_USERS}USRNAME#VALUE3
[SYSTEM_USERS]
   VALUE1 = RDB$REMOTE
   VALUE2 = RDM_MONITOR
   VALUE3 = SYSTEM
   VALUE4 = SYSTEST
   VALUE5 = SYSTEST_CLIG
   VALUE6 = UCX$FTP
   VALUE7 = UCX$NTP
   VALUE8 = UCX$REXEC
   VALUE9 = UCX$RSH
   VALUE10 = UCX$SNMP
   VALUE11 = UCX_LPD
   VALUE12 = UCX_SMTP
   VALUE13 = DEFAULT
   VALUE14 = {Other user}
```

This list lets the user select a VMS System username. The default username is SYSTEM (VALUE3). Selecting the option in the USERS menu, will display a pop-window like this:

1> RDB$REMOTE
2> RDM_MONITOR
3> SYSTEM
4> SYSTEST
5> SYSTEST_CLIG
6> UCX$FTP
7> UCX$NTP
8> UCX$REXEC
9> UCX$RSH
10> UCX$SNMP
11> UCX_LPD
12> UCX_SMTP
13> DEFAULT
14> Other user

However, if the user needs to select a username that's not in the list, selecting 14 will cause OMI to prompt for "Other user". The value between accolades is the value as displayed in the list, and will also be used as prompt when this option is selected.

Figure 5.12: Using the Select list

```
[MENU_MODIFY-JOB]
   INPUT1 = Select days#{TAG|WEEKDAYS}daylist
[WEEKDAYS]
   MESSAGE = Select the day(s) on which the job should run
   DELIMITER = /
   VALUE1 = Monday
   VALUE2 = Tuesday
   VALUE3 = Wednesday
   VALUE4 = Thursday
   VALUE5 = Friday
   VALUE6 = Saturday
   VALUE7 = Sunday
```

Figure 5.13: Using the Tag list

Menu file:

```
[MENU_PLANNING]
   INPUT1 = Weekends to work#{TAG|WEEKENDS}work_weekends
[WEEKENDS]
   message = Select the weekend day(s) you want to work this month
   delimiter = /
   filename = ''F$EXTRACT(3,3,F$TIME())'.DAT
```

File NOV.DAT:

```
Saturday 1st
Sunday 2nd
Saturday 8th
Sunday 9th
Saturday 15th
Sunday 16th
Saturday 22nd
Sunday 23rd
Saturday 29th
Sunday 30th
```

Figure 5.14: File input in Tag and Select lists

The menu file contains:

```
[MENU_MENU]
   INPUT1 = Select a TCP product#{SEL|TCPPRODS}TCP_PROD#VALUE1
[ANOTHER_SELECT]
   VALUE1#LEADING = Set in the menu file
   VALUE2 = Also set in the menu file
#INCLUDE MY_LIB
```

The file MY_LIB.OML contains:

```
[TCPPRODS]
   VALUE1 = UCX
   VALUE2 = WOLLONGONG
   VALUE3 = TCPWARE
   VALUE4 = MULTINET
   VALUE5 = CMU/TEK [ANOTHER_SELECT]
   VALUE1 = Set in the library
   VALUE2#LEADING = Also set in the library file
```

The menu uses a select list that's not available in the menu file, but it is in the library that is included.
When OMI starts, the select list is initialized from the library, and UCX will be the default value stored in the variable TCP_PROD.
The *#LEADING* settings cause the duplicate values to be set as follows:
ANOTHER_SELECT$VALUE1 = "Set in the menu file"
ANOTHER_SELECT$VALUE2 = "Also set in the library"

Figure 5.15: Including standard libraries

```
!  Comments are at the bottom of this file
[MENU_MENU]
   ITEM1 = Leave this menu#command#exit
!  Pretty short menu, isn't it?!
<EOF>
The example you are reading now is not described in this
document, since this text is still a part of the example.
OMI won't read this from the menu file, since the end-of-file
was already reached.  Therefore, these lines don't need to be
outcommented.
```

Figure 5.16: Comments in Menu files

Chapter 6

Writing Modules

Menu items can be used to start OMI modules, using the *CALL* item-type, as in:
ITEM1 = Text on Display#CALL#FILENAME[.OMI]
This item executes the module FILENAME.OMI when selected. Parameters can optionally be specified in the menu item, eventually using the dynamic input identifier (~?), as described in section 5.5.1.

This chapter describes a set of commands and symbols from OMI and the standard toolbox that are available for use in OMI modules.

6.1 Return values of OMI commands

The commands that are available in the standard OMI toolbox will be described in section 6.2. They all return their values as global symbols, if any value is returned. When OMI exits, all symbols will be removed.

NOTE: Make sure you don't use a local symbol with a name that's used for any of the global symbols in OMI commands, since that will always overwrite the returned value of this command!

The status return values of all functions can be: *OMI$_OK* (normal successful completion) *OMI$_WARNING* or *OMI$_ERROR*, where *OMI$_ERROR* has an integer value bigger than *OMI$_WARNING*. This can be used to see if there was any failure by checking the status as below:

$ IF $STATUS .GE. OMI$_WARNING THEN $ <*error-handling*>

If the return value of any command is equal to or greater than *OMI$_WARNING*, the global symbol which should be defined by the command, might not have been set, resulting in more errors or unpredictable behavior due to the use of the value of a former call if the return status is not checked.

6.2 OMI Commands in modules

All commands that are available for use in OMI modules start with the string OMI$. The way they are defined, and can be extended, is described in chapter 7.

6.2.1 OMI$ASK

Format: OMI$ASK *question*

Prompt the user for input. The prompt is specified as a parameter. If no parameter is specified, the value *DEFAULT_INPUT* from the section *[QUESTIONS]* from the configuration file is used.

Return value

The user response is returned in the global symbol *OMI$RESPONSE*. If the user enters <Ctrl/Z>, *OMI$RESPONSE* will be empty, and the status code will be *OMI$_CANCELLED*.

6.2.2 OMI$CALC

Format: OMI$CALC *calculation*

Invoke the OMI calculator. For a description of the calculator, refer to section 9.3. The result is returned in the global symbol *OMI$CALCULATED*.

6.2.3 OMI$CALL

Format: OMI$CALL *omi-module [parameters]*

This command executes an OMI module. With this, it is possible to create several standard OMI modules that will perform generic actions, which will be executed from the action-specific modules.

Parameters

The first parameter is required. It is the name of the OMI module that will be executed.

By default, the file has type *.OMI* and is located in *OMI$:*. If not found there, OMI will look in *OMI$MENU_DIRECTORY:*, unless a full path was specified.
All other parameters will be passed to the module.

Return value

The status code returned by this command is the final exit status of the called module.

6.2.4 OMI$CHECK

Format: OMI$CHECK *variable [message]* [[NO]EMPTY_ALLOWED]

This command can be used to perform checks on the existence of required variables. This is useful if an OMI module needs input, which has to be specified in a menu where no default values are used.

Parameters

The first parameter is required. It specifies the variable name that needs to be

checked.

If a message needs to be displayed when the required variable has not been specified, this can be passed in the second parameter. This is optional.

The third parameter indicates whether or not the variable is allowed to be empty (the variable does exist but has no value). By default, empty values are not allowed. You can override this by specifying *EMPTY_ALLOWED* as the third parameter. This only influences the the message being displayed or not, the return value will not change.

Return values

When the variable is valid, the value *OMI$_OK* is returned.
When the variable exists but is empty, the value *OMI$_WARNING* is returned.
When the variable doesn't exist at all, the value *OMI$_ERROR* is returned.

Examples

```
$ OMI$CHECK USERNM "* ERROR * You didn't specify a username"
$ IF $STATUS .GE. OMI$_WARNING THEN $ EXIT
```

In this example, the OMI module requires a username as input. When this variable has not been filled with a value or has not been specified at all, the error message is displayed. The return value can be *OMI$_WARNING* or greater, indicating not all required information was specified, due to which control is returned to the menu.

```
$ OMI$CHECK FNAME "* ERROR * No file name specified" EMPTY_ALLOWED
$ STATUS = $STATUS
$ IF STATUS .EQ. OMI$_ERROR THEN $ EXIT
$ IF STATUS .EQ. OMI$_WARNING
$ THEN
$ OMI$DISPLAY_MESSAGE "* INFO * Using all files"
$ FNAME = "*.*"
$ ENDIF
```

This example shows how *OMI$CHECK* can be used to check the value of the variable FNAME, allowing an empty value.
The parameter *EMPTY_ALLOWED* causes OMI to display the error message only when the variable doesn't exist at all. The return value does indicate an empty value, but the message won't be displayed, and the OMI module takes action upon that.

6.2.5 OMI$CLEAR_SCREEN

Format: OMI$CLEAR_SCREEN

This command removes all text from the window, leaving the layout intact.

6.2.6 OMI$CMDLINE_CLEAR

Format: OMI$CMDLINE_CLEAR

This command erases the contents of the command line of the OMI menu window.

6.2.7 OMI$CONFIRM

Format: OMI$CONFIRM *question [default]*

This command asks a question that can be answered with Y(es) or N(o)[1], and handles all input and output, returning the value back to the calling procedure.

Parameters

The first parameter is a string, which will be displayed exactly on the input line of the menu string, followed by the choices than can be made "(Y/N)", where 'Y' and 'N' are values that can be modified in the [QUESTIONS] section of the configuration file (see section 4.2.3).

The second parameter represents the default value. This parameter can be *'QUESTIONS$ANSWER_YES'* or *'QUESTIONS$ANSWER_NO'* (***note*** *the use of the single quotes (')!*). If omitted, no default value will be available, and the user is required to enter valid input.

Return value

This command defines a global symbol *OMI$CONFIRMED*. This can be *TRUE* or *FALSE*. The symbol will be cleaned up at the end of the procedure, but also every time the *OMI$CONFIRM* command is called, to prevent conflicts.

Example

```
$ OMI$CONFIRM "Are you sure ?" 'QUESTIONS$ANSWER_NO
$ IF OMI$CONFIRMED THEN $ GOTO USER_IS_SURE
```

This command will display the following question on the input line:
Are you sure ? (Y/[N])
The square brackets indicate the default answer.

6.2.8 OMI$CREATE_MBX

Format: OMI$CREATE_MBX *[logical-name]*

Open a temporary mailbox. This command creates a mailbox for read and write, and a logical which points to the I/O channel.

The mailbox will automatically be removed when the command
$ CLOSE *logical-name*
is issued and no other processes have an open connection with the mailbox. This can be done by the OMI module, but when OMI exits, the mailboxes that are still opened, will be closed automatically.

[1] refer to section 4.2.3 to translate these possible answers

Parameter

The optional parameter specifies the name of the logical that points to the I/O channel, which is the same name that's used for the logical. An error occurs if the parameter specifies a name that's already in use for an existing logical.

If the parameter is omitted, the name will default to *OMI$MAILBOX*.

Return values

OMI$CREATE_MBX returns a logical name which points to the I/O channel. This can be used to write text to the mailbox using the command

```
$ WRITE logical-name "Line of text"
```

or to read using the command

```
$ READ logical-name my-symbol
```

A global symbol with the same name is also defined, containing the device name of the mailbox (MBAxxx:). This value can be transferred to other processes, with which they can open the same mailbox for communication:

```
$ OPEN /READ /WRITE my-logical 'symbol-name'
```

6.2.9 OMI$CREATE_OTF

Format: OMI$CREATE_OTF

This command is used to invoke *on-the-fly* menus. These menus are described in chapter 8.

6.2.10 OMI$DATE_INFO

Format: OMI$DATE_INFO *date*

Calculates the current day number (day-of-year), week number and month number. The values are returned in the following global symbols:

- *OMI$DAYNUMBER*
- *OMI$WEEKNUMBER*
- *OMI$MONTHNUMBER*

Parameter

A date can optionally be specified as a parameter. If omitted, it defaults to the current date. The date information is returned for the current date. The parameter has to be a date in valid ABSOLUTE time format.

6.2.11 OMI$DECRYPT

Format: **OMI$DECRYPT *encrypted-string key-name***

This command can be used to decrypt a string, that was encrypted with *OMI$ENCRYPT*.

Parameters

The encrypted string has to be specified as a parameter. If the encrypted string is encrypted with OMI v1.1 or higher, an encryption key is required. That has to be the same key that was used for encrypting. This key is generated with the *SET KEY* command and stored in the user specific configuration file.

The second parameter should be the name of the key with which the string was encrypted. It has to exist at the keyring of the users configuration file (see section 4.2.5). If it's not there, it can be imported with the OMI command *IMPORT KEY*.

Strings that where encrypted with OMI v1.0b3 - v1.0 can still be decrypted without the need for a key.

Return value

The decrypted result will be returned in the global symbol *OMI$DECRYPTED*.

6.2.12 OMI$DISPLAY_INFO

Format: **OMI$DISPLAY_INFO**

Writes specified information to the menu screen. The information has to be specified as separate records, named *OMI$RECORD1*, *OMI$RECORD2*, ... *OMI$RECORDn*.

The *OMI$RECORDn* symbols should be local.

NOTE: If this command is called more than once during the same run (without any *exit* commands that clean up the local symbols OMI$RECORDn), make sure the symbols *OMI$RECORDn* of the previous calls are erased or overwritten.
E.g. If the first call writes 5 records to the screen, and the second call should write 4 records, it'll also write the last record of the first call. The older fifth record should be removed using:

$ DELETE /SYMBOL /LOCAL OMI$RECORD5

or by specifying it as empty:

$ OMI$RECORD5 = ""

Example

```
$ OMI$RECORD1 = "Username : " + USR$NAME
$ OMI$RECORD2 = "Login dir : " + USR$HOME
$ OMI$RECORD3 = "Owner : " + USR$OWNER
$ OMI$RECORD4 = "Used quota : " + USR$QUOTA
$ OMI$DISPLAY_INFO
$ OMI$WAIT
$ OMI$REFRESH
$ EXIT
```

In this example, information, gathered by the procedure, is stored in the OMI$RECORDn symbols, and presented on the menu window. After being displayed, *OMI$WAIT* is called, to give the user the change to read the infor-

mation before the menu window is set up again.

6.2.13 OMI$DISPLAY_MESSAGE

Format: **OMI$DISPLAY_MESSAGE *message***

This command writes a message to SYS$ERROR, on the message line of the OMI menu window.

Parameter

You can enter any string as a parameter, enclosed by double quotes ("), using double single quotes (") for variable substitution, in exactly the same way as the DCL command WRITE.

6.2.14 OMI$DUMP_INFO

Format: **OMI$DUMP_INFO *filename***

This command can be used to dump the information, that can be displayed by *OMI$DISPLAY_INFO* (see section 6.2.12) to a specified file. The information has to be specified as separate records, named *OMI$RECORD1*, *OMI$RECORD2*, ... *OMI$RECORDn*.

The *OMI$RECORDn* symbols should be local. (See also the note on page 54.)

Parameter
A parameter is required, and should contain the name of the file to write the information to. If the file already exists, the new information will be added at the end of the file. Otherwise, the file will be created.

6.2.15 OMI$ENCRYPT

Format: **OMI$ENCRYPT *string***

This command can be used to perform some rude encryption on the input string, which is specified as a parameter. The OMI encryption method requires an encryption key, which is stored in the user specific configuration file. This key is generated with the *SET KEY* command.

Return value
The result will be returned in the global symbol *OMI$ENCRYPTED*

6.2.16 OMI$GET_VMSMESSAGE

Format: **OMI$GET_VMSMESSAGE *status-code***

This command translates a VMS status code and returns it in the global symbol *OMI$VMS_MESSAGE*.

Parameter

A valid status code is required. This must be an integer value, which can be

specified as decimal or hexadecimal.

Return value

If a message was found, this function returns *OMI$_OK* and stores the found message in the global symbol *OMI$VMS_MESSAGE*.

If no message was found, this function returns *OMI$_WARNING*. On other warnings and errors, *OMI$_ERROR* is returned.

6.2.17 OMI$INPUT_VALIDATE

Format: **OMI$INPUT_VALIDATE**

On any input prompt, variables can be entered using single quotes. This command can be used to validate input entered by the user, to see if the string is fixed, or should be interpreted as a variable or a function.

Before the function is called, the variable *OMI$VARIABLE* has to be defined, containing the name of the variable that should be checked.

Return value

OMI$INPUT_VALIDATE returns *OMI$_OK* when the input string or variable has the correct syntax.

When a variable name was entered between two single quotes (e.g. 'USER-INPUT') and the variable (USER-INPUT in this example) is not defined, the function returns *OMI$_WARNING*.

When a variable name was entered between three single quotes (e.g. "USER-INPUT') and the variable (USER-INPUT in this example) is not defined, the function returns *OMI$_ERROR*.

OMI$_ERROR is also the return value when *OMI$VARIABLE* is not set.

Examples

```
$ OMI$ASK "Password: "
$ OMI$CMDLINE_CLEAR
$ OMI$VARIABLE = "OMI$RESPONSE" ! No single quotes!
$ OMI$INPUT_VALIDATE
$ IF $STATUS .EQ. OMI$_ERROR
$ THEN
$ OMI$DISPLAY_MESSAGE "Invalid password"
$ RETURN
$ ENDIF
$ OMI$ENCRYPT ""OMI$RESPONSE"'
```

In this example, *OMI$INPUT_VALIDATE* is called to check the syntax of the password that was entered.

If the input was enclosed by single quotes ('USER-INPUT'), these are compared literally, but if three single quotes are used ("USER-INPUT'), USER-INPUT will be translated by the call of *OMI$ENCRYPT*.

If USER-INPUT is a function like F$PID(GOTO), the function is executed

causing GOTO to be defined as a new symbol!

```
$ OMI$ASK "Password: "
$ OMI$CMDLINE_CLEAR
$ OMI$VARIABLE = "OMI$RESPONSE"
$ OMI$INPUT_VALIDATE
$ IF $STATUS .EQ. OMI$_WARNING
$ THEN
$ OMI$DISPLAY_MESSAGE "Invalid password"
$ RETURN
$ ENDIF
$ OMI$ENCRYPT OMI$RESPONSE
```

The main difference in this example is the call of *OMI$ENCRYPT*, where the password is not passed to the function between double quotes (causing the input to be translated to uppercase).

Due to this, the return value of *OMI$INPUT_VALIDATE* is checked on warning status; any input will be evaluated when enclosed by single quotes ('USER-INPUT').

If you're not sure if you should check on *OMI$_WARNING* or *OMI$_ERROR*, it's always safe to use the following check:

```
$ IF $STATUS .GE. OMI$_WARNING
```

This might cause valid input to be signaled as invalid, but it will never accept invalid input.

6.2.18 OMI$MSGLINE_CLEAR

Format: **OMI$MSGLINE_CLEAR**

This command erases the contents of the message line of the OMI menu window.

6.2.19 OMI$REFRESH

Format: **OMI$REFRESH**

Rebuilds the menu window with the current menu options. This option is especially useful after the *OMI$DISPLAY_INFO* command.

6.2.20 OMI$POPUP

Format: **OMI$POPUP** ***message*** [***OPTION***[,...]]

This command displays a popup- message box on top of the menu screen. The user then has to press return to acknowledge the message.

Parameter

The first parameter should be enclosed by double quotes. It contains the message that will be displayed in the popup box.

Options

By default, the screen will be refreshed after the call of this command, but when the command is called by an *ON_INIT* element (see section 5.2.1), the menu screen is yet to be drawn. To prevent the screen to be drawn twice, specify a second parameter as *NOREFRESH* in this situation.

Also default behaviour is a call of *OMI$WAIT* after displaying the popup. This can be suppressed by the option *NOWAIT*. This is useful when the popup has a *please wait...* function.

Both options can be concatenated with a comma (,), but make sure there are *no* blanks between the options!

6.2.21 OMI$REVERSE

Format: **OMI$REVERSE *string***

This command does what you would expect the DCL lexical *F$EDIT(string,"REVERSE")* to do, if it would exist; it reverses the string that is given as an argument.

Return value

The reversed value of the input string will be returned in the global symbol *OMI$REVERSED*.

Example

```
$ DIRECTORY = "DEVICE:[TOPDIR.SUBDIR]"
$ OMI$REVERSE 'DIRECTORY'
$!                          result: ]RIDBUS.RIDPOT[:ECIVED
$ REVERSED = OMI$REVERSED - "]"
$!                          result: RIDBUS.RIDPOT[:ECIVED
$ FILENAME = F$ELEMENT(0, ".", REVERSED)
$ REVERSED = REVERSED - ""FILENAME'."
$!                          result: RIDPOT[:ECIVED
$ FILENAME = "RID." + FILENAME + "]" + REVERSED
$!                          result: RID.RIDBUS]RIDPOT[:ECIVED
$ OMI$REVERSE 'FILENAME'
$ FILENAME = OMI$REVERSED
```

This example reads a directory specification as input, reverses it and filters the last directory as a filename. The new filename is composed completely reversed including the file type (reversed of ".DIR").

After this, the complete name is reversed again, resulting in the value DEVICE:[TOPDIR]SUBDIR.DIR.

6.2.22 OMI$SIGNAL

Format: **OMI$SIGNAL facility identifier[,variable,variable...]**

This command signals a message that will be read from the message data file.

Message data files can be created and filled with the *OMI$MANAGE* menu[2].

Parameters

Facility The first parameter points to the facility of which the message should be signaled. This requires the message file *OMI$:facility$MESSAGES.DAT*

Identifier The second parameter identifies the message. The message file has an indexed format, and the identifier is the key that will be used to read the message.

variable(s) If the message contains variables, the string in the data file has to contain the substring *˜S*, which will be replaced by the arguments that were given in the second argument, separated by commas.
NOTE: If arguments are passed to *OMI$SIGNAL* using lexical functions they have to be preceded by a single quote, e.g.:
$ OMI$SIGNAL MYFACIL MYIDENT,VAR1,'F$EDIT(STR2,"UPCASE")

6.2.23 OMI$SUBSTITUTE

Format: **OMI$SUBSTITUTE** ***search-string replace-string input-string***

This command can be used to replace a substring by another string in the input string.

Parameters

The first parameter specifies the search string. This string will be replaced by the substring specified in the second parameter

The third parameter is the input string in which the substring has to be replaced.

Return value

The result of this function is returned in the global symbol *OMI$SUBSTITUTED*.

If the search string of the first argument was not found in the input string, this command returns *OMI$_WARNING*.

6.2.24 OMI$SUBMIT

Format: **OMI$SUBMIT** ***omi-module***

With this command, an OMI module can be started in the background. It works the same as using the qualifier */BACKGROUND=BATCH* from the DCL prompt.

When starting a background process from the OMI prompt, a module can only be started as a batch job; not as a detached process.

All defaults for the Submit command (batch queue, log file) are taken from the *[BGRPROCESS]* section in the user's configuration file.

[2]This menu is not documented, but comes with a basic OMI help file. Type `INFO` in the menu for more information.

Parameter

The parameter is the name of the OMI module to start (*name.OMI*). It has to be located in the *OMI$MENU_DIRECTORY:* search path. This module should be created to run unattended (e.g. using Mailboxes for communication).

6.2.25 OMI$WAIT

Format: **OMI$WAIT**

Causes the procedure to wait until the user hits the Return key. By default the user will be prompted with the line *Press <Return> to continue*. This can be overwritten by modifying the value of *WAIT_PROMPT* in the section *[QUESTIONS]* of the configuration file.

This command is very useful when output is displayed, e.g. using the *OMI$DISPLAY_INFO* command and the screen has to be refreshed before returning to the menu.

When running in batch mode (see section 2.2.2, this command is ignored.

Return value

By default, this command returns status code *OMI$_OK* (normal successful completion). When the user presses Ctrl/Z in stead of <Return>, this function returns *OMI$_CANCELLED*.

Example

```
$LOOP:
$ READ /END_OF_FILE=END INPUT RECORD
$ OMI$RECORD1 = "Date : " + F$ELEMENT(0, "#", RECORD)
$ OMI$RECORD2 = "Count : " + F$ELEMENT(1, "#", RECORD)
$ OMI$DISPLAY_INFO
$ OMI$WAIT
$ IF $STATUS .EQ. OMI$_CANCELLED THEN $ GOTO END
$ GOTO LOOP
$!
$END:
$!
$ CLOSE INPUT
$ OMI$REFRESH
$ EXIT
```

In this example, information, read from an input file, is displayed record by record in a loop. When the user has seen enough, she can cancel the loop by pressing Ctrl/Z.

6.3 OMI symbols

6.3.1 Symbols defined by OMI

When OMI starts, several symbols are defined that can be used in OMI modules.
These symbols are all local.
A list of these symbols is provided in table 6.1.

OMI$_OK	This is an exit status that indicates a normal successful completion.
OMI$_WARNING	This is an exit status that can be used for warnings. If an OMI module exits with a warning, OMI takes no action, but it might be useful when OMI modules call other procedures.
OMI$_ERROR	This is an exit status that can be used for errors. If an OMI module exits with an error, OMI takes no action, but it might be useful when OMI modules call other procedures.
OMI$_CANCELLED	This is an exit status that can be used for special purposes.
OMI$NODENAME	The nodename on which OMI is currently running.
OMI$CURRENT_USER	The name of the user currently running the menu is stored in this symbol.
OMI$MENU_FILE	The filename of the menu that's currently in use, without the directory specification.
OMI$MENU_LOCATION	The location of the menu file that is currently in use.

OMI$_MESSAGE	This symbol is used to store the environmental value of 'message' in. Before an OMI module is called, all messages are switched off with
	`$ SET MESSAGE /NOFACILITY /NOSEVERITY -`
	`/NOIDENTIFICATION /NOTEXT`
	To switch it on, use the command
	`$ SET MESSAGE ´OMI$_MESSAGE`
	This is especially useful when debugging OMI modules. In interactive subprocesses (after the *SPAWN* command), the original message state is automatically restored.

OMI$_NOMATCH	When using the VMS SEARCH command in OMI modules to search for values in files, the return status can be compared with this symbol to check if the value was found, as in:
	`$ IF $STATUS .EQ. OMI$_NOMATCH THEN -`
	`$ GOTO NOTFOUND`

OMI$OPTION	This symbol contains the last numeric value that has been entered by the user, or the command line.
OMI$OPTION_TYPE	This symbol contains the option type of the latest item that was selected by the user. This can be *SUBMENU*, *COMMAND* or *CALL*.

Table 6.1: Local symbols in OMI

6.3.2 Global symbols set by OMI commands

The commands, listed in table 6.2, are available after the OMI command is called that sets the symbol. These commands are described in section 6.2.
All symbols will automatically be removed when OMI exits.

Symbol	**Set by the command**
OMI$CALCULATED	*OMI$CALC*
OMI$CONFIRMED	*OMI$CONFIRM*
OMI$DAYNUMBER	*OMI$DATE_INFO*
OMI$DECRYPTED	*OMI$DECRYPT*
OMI$ENCRYPTED	*OMI$ENCRYPT*
OMI$MAILBOX[a]	*OMI$CREATE_MBX*
OMI$MONTHNUMBER	*OMI$DATE_INFO*
OMI$RESPONSE	*OMI$ASK*
OMI$REVERSED	*OMI$REVERSE*
OMI$WEEKNUMBER	*OMI$DATE_INFO*

Table 6.2: Symbols set by OMI commands

[a]this is the default logical name if none was specified (see section 6.2.8)

Chapter 7

Adding Toolboxes

All commands that can be used in OMI modules (the commands described in section 6.2) are defined in a toolbox. The standard toolbox is called *OMI$TOOLBOX*, and is defined by *OMI$TOOLBOX.INI*.

Figure 7.1 shows how the file *OMI$TOOLBOX.INI* looks like by default, without the comments.

```
[OMI$TOOLBOX] ! Default location is OMI$:, default file type is .COM
   ASK
   CONFIRM
   CHECK
   CREATE_MBX
   DECRYPT
   ENCRYPT
   INPUT_VALIDATE
   REVERSE
   WAIT
```

Figure 7.1: The OMI Toolbox initialization file

If you want to write you own OMI commands, it is possible to add a toolbox to this initialization file.

To do this, add a section, with the name of your toolbox file, enclosed by square brackets, as shown in figure 7.2. By default, OMI looks for a file with that name and file type *.COM*, in the *OMI$:* directory, but this can be overridden by specifying the full path and filename.

Next, create a file called MY_TOOLBOX.COM, and store it in the directory where OMI is located. The file should start with the following two statements:

```
$ GOSUB 'p1'$ ! Or use GOTO if all subroutines contain an EXIT
$ EXIT
```

Make sure the command file contains subroutines for all commands with the label "MY-COMMAND$". You can *not* change this label layout, since OMI checks for the existence of this label during startup!

```
[MY_TOOLBOX]
   MY-EASY-COMMAND
   MY-USEFULL-COMMAND
   MY-OTHER-TOOL
```

Figure 7.2: Initializing a toolbox

To write the code for the commands defined in figure 7.2, the file MY_TOOLBOX.COM should look like something displayed in figure 7.3

```
$ EXIT_STATUS = OMI$_OK ! Routines can change this to OMI$_WARNING
$!                 or OMI$_ERROR when errors show up.
$ GOSUB 'p1'$
$ EXIT 'EXIT_STATUS
$!
$ MY-EASY-COMMAND$:
$!
$ <Write your DCL code here>
$ IF <any-condition> THEN $ EXIT_STATUS = OMI$_ERROR
$ RETURN
$!
$ MY-USEFULL-COMMAND$:
$!
$ <Write your DCL code here>
$ IF <any-condition> THEN $ EXIT_STATUS = OMI$_ERROR
$ RETURN
$!
$ MY-OTHER-TOOL$:
$!
$ <Write your DCL code here>
$ IF <any-condition> THEN $ EXIT_STATUS = OMI$_ERROR
$ RETURN
```

Figure 7.3: Write Toolbox code

The examples in this section would define the following three commands that are available in all OMI modules:

OMI$MY-EASY-COMMAND
OMI$MY-USEFULL-COMMAND
OMI$MY-OTHER-TOOL

NOTE: It is *not* recommended to add commands to the file *OMI$TOOLBOX.COM*, since they might get lost in a future release!
You should add a new section to the initialization file, and a new toolbox command file.

Chapter 8

Creating on-the-fly menus

OMI modules can create *on-the-fly* menus. They are used to setup a temporary submenu.

To do so, all menu items have to be specified in the module as global symbols, with the menu-name *OTF_MENU*, e.g.:

```
$ OTF_MENU$ITEM1 == "Leave this menu#COMMAND#BACK NOEXIT_MODULE"
$ OTF_MENU$ITEM2 == "Do something, then leave#COMMAND#BACK"
$ OTF_MENU$INPUT1 == "Some input please#VARIABLE#This is default"
$ OTF_MENU$ON_EXIT == MY_EXIT_MODULE
```

At least one item element and one input element are required. The global symbols are cleaned up automatically when leaving the menu.

When all menu elements are defined, use the command *OMI$CREATE_OTF* to invoke the menu.

8.1 Limitations

8.1.1 Menu elements

All elements that can be specified in menu files, can be specified in on-the-fly menus, except those listed in table 8.1.

NAME	Names are used for jumping, which is not allowed to and from on-the-fly menus
PASSWORD	Security checks are not performed on
SECURITY	on-the-fly menus

Table 8.1: Elements not allowed in on-the-fly menus

8.1.2 OMI Commands

Since jumping is not allowed, and security checks are not performed, several OMI commands, like *JUMP* and *(RE)SET PASSWORD* are not allowed in on-

the fly menus. The commands or options that are not allowed are marked with a † in chapter 9.

8.2 Tag- and select lists

Tag- and select lists can be used and setup in on-the-fly menus on the same way as they are used in regular menus, like in:

$ OTF_MENU$INPUT1 == "Make a selection#{SEL|MY_LIST}VARIABLE#VALUE1"

In this example, the symbol VARIABLE will be filled with a value that is selected from MY_LIST. This can be an existing section in the menu file, or a section created dynamically.

NOTE: All sections in an on-the-fly menu, *except* the menu section itself, have to be defined as *local* symbols!
E.g.:

$ MY_LIST$VALUE1 = "Yes"
$ MY_LIST$VALUE2 = "No"
$ MY_LIST$VALUE3 = "Maybe"
$ MY_LIST$DELIMITER = "/" ! So it can be used as a taglist as well

A useful way to use tag- and select lists in on-the-fly menus, is with a FILENAME element combined with an *ON_INIT* element to create a value file, as in the figure 8.1.

This example shows an OMI module that sets up an on-the-fly menu. When the menu is invoked, the *ON_INIT* module WRITE_VALUE_FILE.OMI is executed. Let's say that one reads the parameter (which contains a filename), and creates a list of values, that's called by the select list in input element 2.

Selecting input element 2 causes OMI to read all possible values from the specified file and creating a select list with it. When the user leaves the menu, using element 2 ("Write changes..."), the *ON_EXIT* module is called. This module is user specific (which means it should be there for all users in this example).

8.3 Saving an on-the-fly menu

On-the-fly menus are volatile envinroments, that disappear immediatly after leaving the menu.

It might be useful however to save on *on-the-fly* menu so a user can return to the menu at any time.

To do so, the temporary file – used by OMI to collect all – can be saved. An example of this trick can be seen in figure 8.2[1].

[1] I still need to sort a few things out here. The example given here doesn't work just like that, but with small modifications it works. What I still need to sort out, is how it can be documented easily.
If you're reading this note, that means I forgot to update the docs—at the time of writing this I don't have the time to continue, so I'll have to do it later.
Sorry for that, Oscar van Eijk, April 19, 2004.

```
$!  First, define a symbol containing a filename
$!
$ VALUE_FILE = SYS$SCRATCH:OMI$VALUES_'F$CVTIME(,,"WEEKDAY")'._TMP$
$!
$!  Global symbols to define the menu
$!
$ OTF_MENU$ON_INIT == WRITE_VALUE_FILE 'VALUE_FILE
$ OTF_MENU$ON_EXIT == 'OMI$CURRENT_USER'_MODULE$
$ OTF_MENU$ITEM1 == "Exit without changes#COMMAND#BACK NOEXIT_MODULE"
$ OTF_MENU$ITEM2 == "Write changes and exit#COMMAND#BACK"
$ OTF_MENU$INPUT1 == "Record ID#UPDATE_REC#''CURRENT_VALUE'#FRM_RECID"
$ OTF_MENU$INPUT2 == "Select value#{SEL|REC_VALUES}#NEW_VALUE"
$!
$!  Local symbols to define additional sections
$!
$ REC_VALUES$FILENAME = "''VALUE_FILE'" !  Read values from this file
$!
$ FRM_RECID$TYPE = "INTEGER"
$ FRM_RECID$MIN = 100
$ FRM_RECID$MAX = 999
$!
$!  Invoke the menu
$!
$ OMI$CREATE_OTF
$!
$ EXIT $STATUS
```

Figure 8.1: Creating on-the-fly menus

```
$ IF F$SEARCH ("SYS$SCRATCH:OMI$CHECK_OTF_MENU._TMP$") .NES. "" THEN -
   $ COPYY SYS$SCRATCH:OMI$CHECK_OTF_MENU._TMP$ -
   SYS$SCRATCH:OMI$CHECK_OTF_MENU_SAVED$._TMP$

Call other menus or modules here

$ IF F$SEARCH ("SYS$SCRATCH:OMI$CHECK_OTF_MENU_SAVED$._TMP$") .NES. ""
$    THEN
$       COPYY SYS$SCRATCH:OMI$CHECK_OTF_MENU_SAVED$._TMP$ -
         SYS$SCRATCH:OMI$CHECK_OTF_MENU._TMP$
$       OMI$CURRENT_MENU = "OTF_MENU"
$       DELETEE SYS$SCRATCH:OMI$CHECK_OTF_MENU_SAVED$._TMP$;*
$ ENDIF
```

Figure 8.2: Save and restore an on-the-fly menu

Chapter 9

OMI Command Reference

This command reference describes the commands that can be entered by the user on the OMI prompt, or as the third argument in the *COMMAND* item type (see section 5.5.2).

Commands marked with a † are not available in *on-the-fly* menus.

9.1 ALL

Format: **ALL**

In a window where you can specify input fields, this command enables you to enter all values, without the need to select all options separately.

In those windows, the last option will always perform this command. The way you are prompted for it, can be modified in the OMI configuration file by changing the value of *ALL_OPTIONS* in the section *[QUESTIONS]*.

9.2 BACK

Format: **BACK [NOEXIT_MODULE]**

With this command, you can go up one level. When you are at the top level, this will exit the menu.

This command performs the same function as the keystroke <Ctrl/Z>, or entering "0" (zero), followed by <Return>.

Parameter

The option *NOEXIT_MODULE* can optionally be specified, which causes the *ON_EXIT* module not to be executed.

By default, if an *ON_EXIT* module is specified, it will always be executed if the current (sub)menu exits.

9.3 CALC

Format: **CALC** ***calculation***

This command calls the internal OMI calculator. This calculator is very simple, but it can work with floating points and simple functions, and supports parentheses.

Parameter
The calculation is entered as one up to eighth parameters (integers, operator and parentheses can be entered as one parameter, or with blanks between them).

Integers can have a floating point. Dots (.) and commas (,) are recognized as floating points.

Operations currently supported are add (e.g. "-12.4 + 8.53"), subtract (e.g. *14 - 68.032*), multiply (e.g. "1.4 * 7") and divide (e.g. *156 / -2.56*).

When using more complex functions, like: "(2+5)*((8-3)*4/2)/2", OMI will just display the result: "35". To view all steps as they are calculated by OMI, simply put a question mark (?) somewhere in the formula (e.g.: "(2+5)*((8-3)*4/2)/2?"). This will cause OMI to show all separate calculations as they are performed and substituted.
The question mark itself will be removed before calculation starts.

NOTE: The maximum size of the numbers (without the floating point) is 9 digits. This is a limitation in DCL[1].

9.4 CLS

Format: **CLS**

This command removes all text from the window, leaving the layout intact.

9.5 DCL

Format: **DCL** ***dcl_command***
or: **$** ***dcl_command***

Enter any DCL command. The output of the DCL command will be displayed your terminal, and you'll have to press <Return> when execution is completed.

If you don't want the output on your screen, you can use the command *SILENT_DCL*.

In stead of the command *DCL*, the dollar sign (*$*) can also be used.

If the current user is not authorized for interactive DCL, a warning message will be displayed.

[1]The maximum value of an integer is 2,147,483,647. For several internal checks, the maximum value has to be reduced to 999,999,999 in OMI)

Parameter

Enter any valid DCL command. If you omit a parameter, you will be prompted for the command.

When the command DCL is used in a *.MNU* file, OMI can prompt for additional input for the command, if the string *~?* is used in the parameter (see also section 5.5.1 on page 33).

9.6 DELETE

This command deletes values from the current menu environment.

9.6.1 DELETE TEXTAREA

Format: **DELETE TEXTAREA** ***[sequence-number]***

Delete the value of a text area from the current menu and the file in which the value is stored. If the current menu has more than 1 text area, a sequence number is required, indicating which text area should be displayed. This sequence number is not the same as the option number; if internal variable *INPUT2* is a textarea, and *INPUT1* is not, *INPUT2* has sequence number 1!

A text area can only be deleted if the format section of the input element has the option *KEEP* set to *TRUE* (see section 5.6.2 on page 35).

By default, this command will ask for a confirmation. This can be changed with the *CONFIRM* element in the configuration file (see section 4.2.3 on page 21).

This command requires *WRITE* privilege to the current menu.

9.7 EDIT

This command modifies menu elements. All edit commands require write privilege to the menu in which it is invoked.

9.7.1 EDIT ELEMENT

Format: **EDIT ELEMENT** ***option_nr***†

Use this command to edit one of the elements in the current menu. It starts an *on-the-fly* menu, in which modifications to the selected element can be made. When the menu file is updated, changes are in effect immediately.

Parameter

The parameter should be the option number as displayed on the screen, so it's the same number users should enter to select the element.

9.7.2 EDIT MENU_FILE

Format: **EDIT MENU_FILE**†

This command starts an interactive edit session with the editor specified in the user's configuration file, in which the current menu file can be modified. This command is only available in the top-level menu.

9.7.3 EDIT VALUE_FILE

Format: **EDIT VALUE_FILE** ***option_nr***

If one of the options in the current menu has an input element with a tag- or select list that reads the values from a file (see section 5.6.6), that file can be modified with this command.

When this command is entered, the editor, specified in the user's configuration file, is started to edit the file.

Parameter

The parameter should be the option number as displayed on the screen, so it's the same number users should enter to select the input element.

9.8 ENCRYPT

Format: **ENCRYPT** ***section-name element-name key-name***†

Encrypt the specified element in the menu file. The encrypted new value is written to the menu file. Therefore, *WRITE* privilege to the current menu is required.

Since v1.1, encrypting an element requires a key, so you must have generated an own key using the *SET KEY* command.

This command also changes the value of the selected element in the current OMI session, so the value can only be used again if the OMI module in which the value is required, calls *OMI$DECRYPT* (see section 6.2.11).

Parameters

This command requires three parameters. If they are omitted, OMI will prompt for them.

The first parameter specifies the section which holds the element that should be encrypted. Square brackets can be omitted.

The second parameter is the element name.

The third parameter should be the name of the key with which the string has to be encrypted. It has to exist at the keyring of the user's configuration file. If it's not there, it can be imported with the OMI command *IMPORT KEY*, or created with the *SET KEY* command.

Please note that encrypting an element that was already encrypted can cause unpredictable behavior!

9.9 EXIT

Format: **EXIT**†

Use this command to leave OMI. This will return control to your previous interface (e.g. the DCL prompt), from any menu level.

If you started a new menu using the *MENU* command, control will *not* return to the previous menu.

EXIT and *QUIT* are synonymous.

9.10 EXPORT KEY

Format: **EXPORT KEY *key-name***

Copy a key from the user specific keyring to the global keyring. This requires write privilege to the *OMI$:* directory.

Parameter

Specify the name of the key as a parameter. This has to be an existing key at the user specific keyring.

9.11 HELP

Format: **HELP *[topic]***

This displays the online help file of OMI.

9.12 IMPORT KEY

Format: **IMPORT KEY *key-name***

Copy a key from the global keyring to the user specific keyring.

Parameter

Specify the name of the key as a parameter. This file must be made available at the global keyring. Keys with the same name at the user specific keyring are not allowed to exist.

9.13 INCREASE

Format: **INCREASE** [**REFRESH**]
Increase the counter for the current menu with 1. If the current menu has no counter, a message will be displayed.

The counter is automatically increased, every time the menu is accessed.

REFRESH

If this option is specified, the menu screen will be refreshed after the increase.

This option can be used if input fields with the previous counter value have to be cleared, or a comment line containing the counter value should be updated.

9.14 INFO

Format: **INFO**†

If the current menu has an info section in the menu specific help file[2], the *INFO* command can be used to display the information. This way it is possible to add menu specific help.

For more information on writing help files, please refer to rection 5.10

9.15 JUMP

Format: **JUMP** ***menu-name***†

Jump immediately to the submenu with the name that's specified as a parameter of this command.

The name of submenus can be displayed using the *SHOW NAME* command, or automatically by setting the option *DISPLAY_NAMES* in the *[SCREEN]* section of the configuration file to *TRUE*.

9.16 MAIN

Format: **MAIN**†

This command will always return to the top- level menu, from anywhere in the structure.

9.17 MENU

Format: **MENU** ***menu_name***†

Select another menu to work with. This command cleans up the current environment completely, causing the menu that you are working with, to be erased from memory.

NOTE: All values that have been entered during the current OMI session will be lost!

Parameter

Enter a valid menu file. If the parameter is omitted, you will be prompted to enter one. The new menu will be loaded in memory, overwriting the current menu.

To retrieve a listing of all available menu files in *OMI$:* and *OMI$MENU_DIRECTORY:*, enter a question mark (*?*).

[2] *OMI$MENU_DIRECTORY:menu-file-name.OMH*, where *menu-file-name* is the same as the filename of the current menu file

9.18 QUIT

Format: **QUIT**†

Use this command to leave OMI. This will return control to your previous interface (e.g. the DCL prompt), from any menu level.

If you started a new menu using the *MENU* command, control will NOT return to the previous menu.

QUIT and *EXIT* are synonymous.

9.19 REFRESH

Format: **REFRESH**
Refresh the menu screen.

9.20 RESET

This command resets or removes the value of the variable, specified by a keyword.

9.20.1 RESET AUTO_REFRESH

Format: **RESET AUTO_REFRESH**

Use this command to disable the automatic screen refresh is disabled.

9.20.2 RESET COUNTER

Format: **RESET COUNTER**

Set the counter of the current menu to 0 (zero) (see also footnote 1 on page 29). If the current menu has no counter, a message will be displayed.

9.20.3 RESET NAME

Format: **RESET NAME**†

Remove the name of the current menu. A name can be used to jump between menus, or to startup in a submenu when the name is specified on the DCL prompt. This command requires *WRITE* privilege to the current menu.

9.20.4 RESET ORDER

Format: **RESET ORDER**

If the current menu has a required order (see section 5.2.8), a list is maintained in the background, which remembers the input elements that have already been selected. The list changes every time one of the required inputs is selected for the first time. If all required inputs have been selected, the list is empty, and

will remain empty during the current OMI session, unless the command *RESET ORDER* is issued in the menu having the list.

9.20.5 RESET PASSWORD

Format: **RESET PASSWORD**†

This command removes the password for the current menu, if one was set. This requires *WRITE* privilege to the current menu.

9.20.6 RESET VARIABLES

Format: **RESET PASSWORD [BACKGROUND]**

Use this command to reset all variables in the current menu to their default values as specified in the menu file.

The *BACKGROUND* option can be added to prevent OMI from refreshing the screen and displaying a message if the command is called in the background.

9.21 SET

The *SET* command can be used to change the value of an internal OMI variable.

9.21.1 SET AUTO_REFRESH

Format: **SET AUTO_INCREASE *value***

Use this command to set or change the interval in seconds with which the menu screen is automatically refreshed. The value should be between 0 and 255. When set to 0, the automatic refresh is disabled.

9.21.2 SET COUNTER

Format: **SET COUNTER**

Use this command to set the counter for the current menu to another value. If the current menu has no counter, a message will be displayed.

9.21.3 SET KEY

Format: **SET KEY *value***

This command defines a personal key. This key will be used for the *ENCRYPT* command in OMI, and for the *OMI$ENCRYPT* and *OMI$DECRYPT* command in OMI modules.

When menu items are encrypted using any of these commands, they can only be decrypted by the same user, or by users who have the same key defined.

Keys cannot be shared between users (except in shared installations, see section 1.2.2). Every user can define only one key.

Parameter

The parameter must be an integer value between 1 and 1,000,000.

9.21.4 SET NAME

Format: **SET NAME *menu-name*†**

Modify the name of the current menu, or define one if the current menu has no name. A name can be used to jump between menus, or to startup in a submenu when the name is specified on the DCL prompt.

This command requires *WRITE* privilege to the current menu.

NOTE: If you want a menu to have the name *RESET*, you need to set this name using the editor, this won't work with the *SET* command, since this name conflicts with OMI internal names.

Parameter

The new menu name can optionally be specified as a parameter. If omitted, OMI will prompt for a new menu name.

The parameter cannot be *RESET*.

9.21.5 SET PASSWORD

Format: **SET PASSWORD**†

Change the password for the current menu. If the current menu has no password, it will be defined using this command. The password must be at least five characters long.

This command requires *WRITE* privilege to the current menu.

NOTE: The password can not have the value *RESET*!

9.21.6 SET WIDTH

Format: **SET WIDTH *value***

Use this command to modify the screen width. The value can be 80 or 132.

9.22 SHOW

This command displays the value of an internal OMI variable.

9.22.1 SHOW COUNTER

Format: **SHOW COUNTER**

Show the current value of the counter for this menu. If the current menu has no counter, a warning message will be displayed.

9.22.2 SHOW NAME

Format: **SHOW NAME**

Display the name of the current menu, if one has been set. These names can be used to jump to the menu immediately from anywhere in the menu structure,

or by specifying the menu name as the second parameter on the DCL command line when OMI is started.

9.22.3 SHOW ORDER

Format: **SHOW ORDER**

If the current menu has a required order (see section 5.2.8), this command can be used to display the list which is maintained in the background, to see which input elements still have to be selected, and in which order.

9.22.4 SHOW TEXTAREA

Format: **SHOW TEXTAREA [sequence-number]**

Display the value of a text area from the current menu. If the current menu has more than 1 text area, a sequence number is required, indicating which text area should be displayed. This sequence number is not the same as the option number; if internal variable *INPUT2* is a textarea, and *INPUT1* is not, *INPUT2* has sequence number 1

If the option *LARGE* in the format section of the variable is set to *TRUE* (see section 5.6.2), it is possible that this command does not display anything without warning.

9.22.5 SHOW VERSION

Format: **SHOW VERSION**

Show the current version of OMI.

9.22.6 SHOW VMS_MESSAGE

Format: **SHOW VMS_MESSAGE *status-code***

This command translates a VMS status code to a message and displays it on the OMI message line.

Parameter

A valid status code is required. If omitted, OMI will prompt for a status code. The value can be specified as decimal or hexadecimal integer value.

9.23 SILENT_DCL

Format: **SILENT_DCL *dcl_command***

Enter any DCL command. The output of the command will not be displayed on the screen, leaving your layout intact.

The destination of the output, including errors, is defined in the OMI configuration file, element *SILENT_OUTPUT* in section *[MAIN]*. By default, this will be NLA0:

The final status code will be displayed on the OMI message line.

Parameter

Enter any valid DCL command. If you omit a parameter, you will be prompted for the command.

When the command *SILENT_DCL* is used in a menu file, OMI can prompt for additional input for the command, if the string *~?* is used in the parameter.

If the current user is not authorized for interactive DCL, a warning message will be displayed.

9.24 SPAWN

Format: **SPAWN *dcl_command***

Spawn an interactive subprocess without leaving OMI. Any VMS command can optionally be given as a parameter. If the command is entered, control is returned to the calling process when the execution of the command is completed.

If the current user is not authorized for interactive DCL, a warning message will be displayed.

Parameter

Enter any valid DCL command. If you omit a parameter, you will be prompted for the command.

When the command SPAWN is used in a menu file, OMI can prompt for additional input for the command, if the string *~?* is used in the parameter.

9.25 SUBMIT

Format: **SUBMIT *omi_module***

With this command, an OMI module can be started in the background. It works the same as using the qualifier */BACKGROUND=BATCH* from the DCL prompt.

When starting a background process from the OMI prompt, a module can only be started as a batch job; not as a detached process.

All defaults for the Submit command (batch queue, log file) are taken from the *[BGRPROCESS]* section in the user's configuration file.

Parameter

The parameter is the name of the OMI module to start (*name*.OMI). It has to be located in the *OMI$MENU_DIRECTORY:* search path. This module should be created to run unattended (e.g. using Mailboxes for communication).

Appendix A

Example OMI Menu

This example is included in the distribution as *OMI$EXAMPLE.MNU*.

```
[MENU_MENU]
  owner = system
  title = OMI Example Menu
  name  = Main                                    ! Menu name for shortcut
  item1 = Print the menu file#command#silent_dcl print -
  /queue=~?{Printer queue:} /notify omi$:omi$example.mnu
                                                  ! Print this menu file to read
                                                  ! on paper what's happening.
  item2 = Exit#command#exit                       ! Command item, EXIT the menu
  item3 = Protected Menu#submenu#pwdtest ! Submenu item, calls PWDTEST
  item4 = Get some inputs#submenu#inptest ! Submenu item, calls INPTEST
  item5 = Counter menu#submenu#cnttest ! Submenu item, calls CNTTEST

[MENU_PWDTEST]
   password =
!  password_level = 2                           ! If not outcommented, nobody
                                                  ! can change the password.
  name  = Protected                               ! Menu name for shortcut
  comment = This menu is password protected     ! Put a comment line on screen
  security = grant_me_write                       ! Define authorisation
  title = Dummy Menu                              ! Give this one another title
  item1 = Main Menu#command#back ! Command item, one level BACK


[MENU_INPTEST]
  name  = Input                                   ! Menu name for shortcut
  item1 = Main Menu#command#back ! Command item, one level BACK
  item2 = Formatted inputs#submenu#frmtest ! Submenu item, calls FRMTEST
  item3 = Display the inputs#call#omi$:omi$example
! Call a test procedure to
                                                  ! display the inputs.
  input1 = Select a weekday#{SEL|weekdays}inp_weekday#value1
! Select from a predefined
```

```
                                                ! list. Default is "Monday"
  input2 = Select some months#{TAG|months}inp_months
                                                ! Select one ore more months.

[MENU_FRMTEST]
  name  = Format                                ! Menu name for shortcut
  item1 = Main Menu#command#main ! Command item, back to MAIN
  item2 = Previous Menu#command#back ! Command item, one level BACK
  input1 = Enter a date#inp_date#today#date_frm ! Input item, should be a date
! Default is "TODAY"
  input2 = Enter a valid filename#inp_file##fname_frm
      ! Input item, should be a filename
  input3 = Enter an int between 0 and 100#inp_int##int_frm
      ! Input item, should be an integer

[MENU_CNTTEST]
  counter = just_a_counter          ! Define a counter for this menu
  comment = The counter is now {counter$just_a_counter}
  name  = Counter1                              ! Menu name for shortcut
  item1 = Main Menu#command#back       ! Command item, one level BACK
  item2 = More counters#submenu#cnttest2      ! Submenu item, calls CNTTEST2

[MENU_CNTTEST2]
  counter = another_counter              ! Define a counter for this menu
  comment = The counter is now {counter$another_counter}
  name  = Counter2                              ! Menu name for shortcut
  item1 = Main Menu#command#main       ! Command item, back to MAIN
  item2 = Previous Menu#command#back       ! Command item, one level BACK
  item3 = Reset the counter#command#reset counter
                                                ! Command item, RESET counter
  item4 = Increase the counter#command#increase refresh
                                                ! Command item, INCREASE counter
                                                ! followed by a screen refresh
                                                ! to update the comment line
  item5 = Display the arrays#command#dcl show symbol inp_array*
                                                ! Command item, call to DCL
                                                ! to display the inputs
  input1 = Enter anything#inp_arraya'counter$another_counter
  input2 = Select a day#{SEL|weekdays}inp_arrayb'counter$another_counter


[DATE_FRM]
  type = date                                   ! Input should be a valid date
  format = absolute                             ! Will be convert. to absolute

[FNAME_FRM]
  type = filespec                               ! Input should be of FILESPEC type
  wildcards = true                              ! Wildcards are allowed
  required = false                              ! The file does not have to exist
```

```
! The arguments below are from version 1.0b2, the FILESPEC type was implemented
! later, but with the arguments below, used in the STRING type, almost
! the same result can be reached.
! The arguments are outcommented in stead of remove, so they can remain here
! as an extra example.
!
!  upcase   = true                          ! String will be conver. to upcase
!  collapse = false                         ! All blanks will be removed
!  ivchars = @#$%^&*()'?/|\+`~{}[]<>        ! These characters are not allowed


[INT_FRM]
  type = integer                           ! Input should be an integer
  min = 0                                  ! Input value should be between
  max = 100                                !  '0' and '100'


[WEEKDAYS]
!
! Define the days of the week from which a selection can be made
! using the SEL input in the INPTEST menu (input1).
!
  value1 = Monday
  value2 = Tuesday
  value3 = Wednesday
  value4 = Thursday
  value5 = Friday
  value6 = Saturday
  value7 = Sunday

[MONTHS]
!
! Define the months that can be tagged using the TAG input in the
! INPTEST menu (input2).
!
  delimiter = /
  value1  = January
  value2  = February
  value3  = March
  value4  = April
  value5  = May
  value6  = June
  value7  = July
  value8  = August
  value9  = September
  value10 = October
  value11 = November
  value12 = December
```

```
[COUNTER]
  just_a_counter  = 0
  another_counter = 0

[GRANT_ME_WRITE]
!
! This security section was added, granting write privilege to my userid,
! to enable me to set a password on the PWDTEST menu, without the need
! to log in as SYSTEM (the owner of the menu)
!
  DE70VEO   = write
  all_users = read, exec

<EOF>
```

The '<EOF>' above will be interpreted as an end-of-file. This means the OMI will never read these lines. This is done to improve performance while initializing; long comments can now be written past the end-of-file.

Appendix B

Example OMI Module

This example is included in the distribution as *OMI$EXAMPLE.OMI*.

```
$!      set message 'omi$_message'       ! Uncomment this for debugging
$       omi$record1 = "The weekday you have selected was:
$       omi$record2 = "  ''inp_weekday'"
$       omi$record3 = "The month(s) you have tagged were:
$       _counter = 0
$       omi$check inp_months "" empty_allowed
$       if $status .ge. omi$_warning
$          then        ! Month input was not yet selected
$              omi$record4 = "  <None>"
$              goto month$end_loop
$       endif
$!
$ month$_loop:
$!
$       _month = f$element(_counter, months$delimiter, inp_months)
$       if _month .eqs. "" .or. _month .eqs. months$delimiter then -
          $ goto month$end_loop
$       _record = '_counter + 4
$       omi$record'_record = "  ''_month'"
$       _counter = _counter + 1
$       goto month$_loop
$!
$ month$end_loop:
$!
$       omi$display_info
$       omi$confirm "More ? " 'questions$answer_yes
$       if .not. omi$confirmed then $ goto end
$       omi$record1 = " The date you entered was     : ''inp_date'"
$       omi$record2 = " The filename you entered was : ''inp_file'"
$       omi$reverse "''inp_file'" "''inp_date'"
$       omi$record3 = " Reversing this results in    : ''omi$reversed'"
$       omi$record4 = " The integer you entered was  : ''inp_int'"
$       omi$record5 = " You called the Counter menu ''counter$another_counter' times"
```

```
$!
$! Make sure the list with months won't be redisplayed, starting with rec. 5
$       if f$type(omi$record6) .nes. "" then -
          deletee/symbol/local omi$record5
$!
$       omi$display_info
$       omi$wait
$       omi$refresh
$!
$ end:
$!
$       exit
```

Index