VMS Software

VSI OpenVMS

VSI OMNI Pocket Guide

Document Number: DO-DOMNPG-01A

Publication Date: June 2021

Revision Update Information: This is a new guide.

Operating System and Version: VSI OpenVMS Alpha Version 8.4-2L1
VSI OpenVMS Integrity Version 8.4-1H1

Software Version: VSI OMNI Version 4.1

VMS Software, Inc. (VSI)
Burlington, Massachusetts, USA

VSI OMNI Pocket Guide

Legal Notice

Preface

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

VSI seeks to continue the legendary development prowess and customer-first priorities that are so closely associated with the OpenVMS operating system and its original author, Digital Equipment Corporation.

2. Intended Audience

This guide is intended for VSI OMNI users with experience in OpenVMS programming for distributed systems applications and knowledge of manufacturing applications and the Manufacturing Message Specification.

3. Document Structure

This guide consists of four chapters:

Chapter 1 summarizes API commands.

Chapter 2 describes ODF concepts.

Chapter 3 summarizes ODF commands.

Chapter 4 describes OMNICL concepts and commands.

4. Associated Documents

This guide is intended to complement the online VSI OMNI documentation set that provides more detailed information about API, ODF, and OMNICL.

The online documentation set includes:

VSI OMNI Application Programmer's Guide

VSI OMNI Guide to Using OmniView

VSI OMNI Software Installation Guide

VSI OMNI Network Manager's Guide

The online documentation can be found under the following VSI OMNI V1.1 file names:

- OMNI$APPL_PROG_GUIDE
- OMNI$NETWORK_MNGR_GUIDE
- OMNI$INSTALLATION_GUIDE
- OMNI$GUIDE_TO_OMNIVIEW

5. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

6. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com

7. Typographical Conventions

The conventions found in the following table are used in this document.

Convention	Meaning
Return	Press the Return key.
UPPERCASE TYPE	All uppercase letters in a command line indicate keywords that must be entered. You can enter them in either uppercase or lowercase. You can use the first three characters to abbreviate command keywords, or you can use the minimum unique abbreviation.
lowercase ital- ics	Lowercase italics in command syntax or examples indicate variables for which either you or the system supplies a value.
[ ]	In examples showing VMS directory specifications, square brackets are a necessary part of the specification, [*directory-name*]. In a procedure, square brackets in an inquiry enclose the default response for the inquiry.
key	Press the specified key.
CTRL/*x*	While holding down the **Ctrl** key, press the key specified by *x*.
⋮	Vertical ellipses (dots) in examples represent data that has been omitted.

Chapter 1. Programming with VSI OMNI

An application can use the VSI OMNI programming interface to perform the following operations:

- Initialize VSI OMNI.
- Load VMD object definitions and obtain a VMD definition handle.
- Obtain handles for object definitions associated with a VMD.
- Establish associations with remote applications and request other MMS environment and general management services.
- Request VMD support services.
- Request domain services and receive client requests for domain services.
- Request program invocation services.
- Request variable access services and receive client requests to read and write local variables.
- Request file management services.
- Create, modify, and retrieve definitions using the VSI OMNI run-time facility.

See the online *VSI OMNI Application Programmer's Guide* for details about these operations.

VSI OMNI Procedure Call Format

The format section describes the syntax of the procedure call – that is, the call elements in their proper sequence. The general format for a call with multiple arguments is:

```
status=OMNI$ procedure [_A] arg1 ,[arg2] . . . ,[argn]
```

The elements are defined in Table 1.1

Table 1.1. Elements

Element	Meaning
status	A location to receive a longword condition value that the procedure returns to the caller.
procedure	A VSI OMNI procedure.
_A	A suffix to specify asynchronous operation of the requested service.
arg1 ,[arg2] . . . ,[argn]	A list of required and optional arguments.
[. . .]	Square brackets, used to indicate that the enclosed element is optional. In the general format example, *arg2* and *argn* are optional.
,	A comma, used to separate arguments in an argument list. Omitted arguments must be indicated by 0.

Note

All omitted arguments must be indicated by 0. Omitted arguments include both optional arguments and placeholder arguments reserved for use in future versions of VSI OMNI.

VSI OMNI Return Values

The VSI OMNI procedure calls return the information listed in Table 1.2

Table 1.2. Return Values

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value in R0**

See the online *VSI OMNI Application Programmer's Guide* for information about specific, procedure call return values.

VSI OMNI Procedure Calls

The following section defines each VSI OMNI procedure call and provides the command line format of each procedure. See the section called “VSI OMNI Procedure Call Argument Definitions” for information about procedure call argument definitions.

VSI OMNI Procedure Call Argument Definitions

The following section defines arguments used by VSI OMNI procedure calls.

`attraddress`

type: OMNI$L_ENUMERATION_CONST

access: read only

mechanism: by reference

Attraddress specifies the address of a variable whose value is the attribute to retrieve.

`called_vmd_handle`

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The handle of the VMD that the called application will make available to the remote peer. (On an OMNI $LISTEN, the called application is the local application that has issued the OMNI$LISTEN request.)

`calling_vmd_handle`

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide* The handle of a VMD that the application wants to make available during the association. If the *calling_vmd_handle* is included, the application can receive client requests from the remote peer to operate on the VMD. In addition, the handle also enables VSI OMNI to service network objects – such as variables – for the user.

`class`

type: OMNI$L_ENUMERATION_CONST

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide Class* identifies the class of definition to create.

The value of the class parameter is one of the values listed in Table 1.3

Table 1.3. Class Constants

Constant	Meaning
OMNI$K_CLS_VMD	VMD
OMNI$K_CLS_DOM	Domain
OMNI$K_CLS_PI	Program Invocation
OMNI$K_CLS_NAMED_VAR	Named Variable
OMNI$K_CLS_UNNAMED_VAR	Unnamed Variable
OMNI$K_CLS_MSG	Message
OMNI$K_CLS_MMS_NAMED_TYPE	MMS Named Type
OMNI$K_CLS_MMS_TYPE_SPECIFICATION	MMS Type Specification
OMNI$K_CLS_APP_NAMED_TYPE	Application Named Type
OMNI$K_CLS_APP_TYPE_SPECIFICATION	Application Type Specification
OMNI$K_CLS_MMS_STRUCT_COMP	MMS Structure Component
OMNI$K_CLS_APP_STRUCT_COMP	Application Structure Component

`conclude_flag`

type: longword

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

Takes one of the values listed in Table 1.4

Table 1.4. Conclude Values

Value	Meaning
non 0	VSI OMNI delivers all conclude indications to the calling application for processing.

Value	Meaning
0	VSI OMNI automatically accepts conclude requests.

context

type: OMNI$R_CONTEXT

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

A location for use by VSI OMNI. The *context* is the same value that was returned by OMNI $_GET_REMOTE_ ATTRIBUTES.

contextaddress

type: OMNI$L_CONTEXT

access: modify

mechanism: by reference

Contextaddress is the address of a variable. This parameter is used only if you modify a multivalued attribute.

The value of contextaddress should be initialized to NULL. If a multivalued attribute is modified, reset the value of the context to zero before another multivalued attribute is modified.

After you specify values for the attribute, call OMNI$END_ LIST. Do not modify the value of the context until OMNI$END_LIST has been called.

ctrl_struc

type: OMNI$R_CTRL

access: read only

mechanism: by reference

A control structure to handle an event flag, AST routine, and AST parameter.

The *ctrl_struc* parameter is the address of the control structure.

domain_file

type: OMNI$T_FILE_NAME

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The file specification for a VMS file containing the domain contents. If you omit this parameter, OMNI uses the domain contents file name associated with the ODF definition of the domain.

domain_handle

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The identifier of a loaded domain definition.

`defhandle`

type: OMNI$L_HANDLE

access: read only

mechanism: by value

Defhandle specifies the handle of the definition to modify. The value of this parameter is one of the following:

- VMD Handle
- Domain Handle
- PI Handle
- Named Variable Handle
- Unnamed Variable Handle
- MMS Named Type Handle
- Application Named Type Handle
- MMS Type Specification Handle
- Application Type Specification Handle
- MMS Structure Component
- Application Structure Component
- Message Handle

`def_name`

type: character-coded text string

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The name of the definition to search for.

`domain_file`

type: OMNI$T_FILE_NAME

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The file specification for a VMS file to receive the domain. If you omit this parameter, OMNI uses the file name associated with the domain.

`execution_argument`

type: OMNI$T_EXEC_ARG_STR

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

A character string appropriate to the execution of the program invocation. This field overrides the ODF execution argument definition.

`handle`

type: OMNI$L_HANDLE

access: write only

mechanism: see the *VSI OMNI Application Programmer's Guide*

A location to receive the definition handle.

`handleaddress`

type: OMNI$L_HANDLE

access: write only

mechanism: see the *VSI OMNI Application Programmer's Guide*

Handleaddress specifies the return address of the definition handle.

`incoming_vmd_struc`

type: OMNI$R_VMD_DEF

access: write only

mechanism: see the *VSI OMNI Application Programmer's Guide*

Service parameters proposed by the calling (remote) application.

`invoke_id`

type: longword

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The identifier assigned by VSI OMNI to the service the application wants to cancel.

last_modified

type: OMNI$L_LAST_MOD_DATE

access: write only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The date on which the file was last modified on the remote system. Last modified dates before January 1, 1970 are not supported.

local_file_name

type: OMNI$T_FILE_NAME

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The file specification for the local file to receive the copy.

method_handle

type: OMNI$HANDLE

access: read only

mechanism: sse the *VSI OMNI Application Programmer's Guide*

Method_handle modifies the default presentation of a variable.

model

type: character-coded text string

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The name of the MMS software service provider. VSI OMNI uses the name when replying to an Identify Request. The default name is OMNI.

modifier_object

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

Reserved for future use.

negotiated_vmd_struc

type: OMNI$R_VMD_DEF

access: write only

mechanism: see the *VSI OMNI Application Programmer's Guide*

A VMD data structure to receive negotiated service parameters.

new_remote_file_name

type: OMNI$T_FILE_NAME

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The file specification (in native format) for the new name.

object_attribute

type: longword (unsigned)

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

object_handle

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The identifier of a loaded definition for the object to be created.

omni_iosb

type: OMNI$R_IOSB

access: write only

mechanism: by reference

The VSI OMNI I/O status block. For a description of the codes that VSI OMNI returns to the IOSB, see the *VSI OMNI Application Programmer's Guide* .

The *omni_iosb* parameter is the address of the status block.

original_remote_file_name

type: OMNI$T_FILE_NAME

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The file specification (in native format) for the remote file to rename.

pdata

type: depends on value

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

Pdata is a pointer to a data structure that receives the value of the object or to a data structure that contains the value of the object.

`pi_handle`

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The identifier of a loaded program definition.

`reason`

type: condition value

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The reason the conclude request is being rejected. Reserved for future use.

`receive_struct`

type: depends on attribute specified

access: write only

mechanism: see the *VSI OMNI Application Programmer's Guide*

VSI OMNI uses the code to construct a location to contain the returned attribute value.

`remote_file_name`

type: OMNI$T_FILE_NAME

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The file specification (in native format) for the remote file to receive the copy.

`req_method_handle`

type: OMNI$L_HANDLE

access: read only

mechanism: by reference

Identifier of a defined and loaded access method. If present, this method overrides the method associated with the Response Data object in ODF.

The *req_method_handle* parameter is the address of the handle.

revision

type: character-coded text string

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The version number of OMNI software. The default version is V1.0.

scope

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

A handle indicating the scope of the search. The *scope* parameter is typically the handle of a VMD or a domain. VSI OMNI limits its search to the specified VMD or domain.

The *scope* parameter is the handle of a VMD (obtained using OMNI$LOAD_DEFINITIONS) or a domain (obtained by a previous call to OMNI$GET_HANDLE_BY_NAME).

size

type: OMNI$L_SIZE_OF_FILE

access: write only

mechanism: see the *VSI OMNI Application Programmer's Guide*

Size of the file on the remote device.

translate_flag

type: longword

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

One of the values in Table 1.5 to specify the way VSI OMNI handles initation indications received from a remote VMD.

Table 1.5. Values

Value	Meaning
0	VSI OMNI rejects the initiation if the calling application specifies a VMD whose definition is not currently loaded.
non 0	If the calling application specifies a VMD that is not currently loaded, VSI OMNI returns the initiation indication, creates a dummy VMD definition, and passes the handle of the dummy definition to the user.

value

type: see table

access: write only

mechanism: by reference

Value is the address of a buffer in which the attribute value is returned.

value_structure

type: depends on value

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

A data structure containing the value of the object.

vmd_handle

type: OMNI$L_HANDLE

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The longword identifier of a loaded VMD definition. The *vmd_handle* is returned by the OMNI $LOAD_ DEFINITIONS procedure.

vmd_name

type: character-coded text string

access: read only

mechanism: see the *VSI OMNI Application Programmer's Guide*

The name of the VMD whose local definitions you want to load.

vendor

type: character-coded text string

access: read only

mechanism: by reference

The name of the system vendor. VSI OMNI uses the *vendor* name when replying to an Identify-Request. The default name is VSI.

OMNI$ABORT

OMNI$ABORT — Immediately terminates an association with a remote VMD. All pending requests return with status OMNI$ABORT.

Format

```
status=OMNI$ABORT[_A] vmd_handle, [omni_iosb], [ctrl_str]
```

OMNI$ACCEPT_CONCLUDE

OMNI$ACCEPT_CONCLUDE — Accepts an association conclude request from a remote application.

Format

```
status=OMNI$ACCEPT_CONCLUDE[_A] vmd_handle, [omni_iosb], [ctrl_str]
```

OMNI$ACCEPT_CONNECT

OMNI$ACCEPT_CONNECT — Accepts an association request from a remote VMD.

Format

```
status=OMNI$ACCEPT_CONNECT[_A] vmd_handle, reserved,
[conclude_flag], [omni_iosb], [ctrl_struc]
```

OMNI$CANCEL

OMNI$CANCEL — Cancels a request previously issued but not yet completed.

Format

```
status=OMNI$CANCEL[_A]invoke_id, [omni_isob], [ctrl_struc]
```

OMNI$CONCLUDE

OMNI$CONCLUDE — Brings an association with a remote VMD to an orderly conclusion. It is valid to conclude an association only when all requests have been satisfied. If there are pending operations on the association, VSI OMNI does not accept an OMNI$CONCLUDE request.

Format

```
status=OMNI$CONCLUDE[_A] vmd_handle, [omni_iosb], [ctrl_str]
```

OMNI$CONNECT

OMNI$CONNECT — Initiates an association with a remote VMD. The calling VMD specifies network objects.

Format

```
status=OMNI$CONNECT[_A] vmd_handle,reserved, [calling_vmd_handle],
[negotiated_vmd_struc], [conclude_flag], [omni_iosb], [ctrl_struc]
```

OMNI$CREATE

OMNI$CREATE — Creates an object on a VMD.

Format

```
status=OMNI$CREATE[_A] [invoke_id], object_handle,
[modifier_object], [omni_iosb], [ctrl_struc]
```

OMNI$DELETE

OMNI$DELETE — Deletes a specific object on a VMD. Only objects with no dependencies can be deleted. For example, a domain with an associated program invocation cannot be deleted until the program invocation is deleted.

Format

```
status=OMNI$DELETE[_A] [invoke_id], object_handle,
[modifier_object], [omni_iosb], [ctrl_struc]
```

Description

OMNI$DELETE does not delete the definition obtained by OMNI$LOAD_DEFINITIONS. The only way to delete an object definition is through the use of ODF.

OMNI$DOWNLOAD

OMNI$DOWNLOAD — Initiates the downloading of a domain to a remote VMD.

Format

```
status=OMNI$DOWNLOAD[_A] [invoke_id],domain_handle, [domain_file],
[modifier_object], [omni_iosb], [ctrl_struc]
```

Description

- You cannot download a domain that already exists on the VMD.
- If a domain file specification is not included in the OMNI$DOWNLOAD call or in the ODF definition, VSI OMNI returns an error code.
- If, on completion of the download service, the domain has been discarded by the remote application, the user is notified in the IOSB.
- VSI OMNI supports only one download to a domain at a time.

OMNI$END_LIST

MNI$END_LIST — Terminates the use of a list context. OMNI$GET_REMOTE_ATTRIBUTES allocates memory. OMNI$END_LIST frees that same memory. Failure to call OMNI$END_LIST results in an increase of memory usage.

Format

```
status=OMNI$END_LISTcontext
```

OMNI$FDELETE

OMNI$FDELETE — Deletes a file from a remote system. Wildcards are delivered to the remote device as specified. See PIC for remote device specification.

Format

```
status=OMNI$FDELETE[_A] [invoke_id], vmd_handle, remote_file_name,
[modifier_object], [omni_iosb], [ctrl_struc]
```

OMNI$FDIR

OMNI$FDIR — Obtains a list of file specifications from a remote directory. To get the file specifications in the directory, call the OMNI$GET_ATTRIBUTE procedure for each filespec. OMNI$FDIR allocates space for a buffer to contain the directory. To release the buffer, call OMNI $END_LIST.

Format

```
status=OMNI$FDIR[_A] [invoke_id], vmd_handle, remote_directory,
context, [modifier_object], [omni_iosb], [ctrl_struc]
```

OMNI$FGET

OMNI$FGET — Copies a file from the remote system to the local system.

Format

```
status=OMNI$FGET[_A] [invoke_id], vmd_handle, remote_file_name,
local_file_name, [size], [last_modified], [modifier_object],
[omni_iosb], [ctrl_struc]
```

Description

- If an error occurs while writing the file, the file is deleted.
- Wildcards must reduce to one file name.
- If the local file already exists, the record attributes are inherited from the previous version.
- If the local file does not exist, the file created is compatible with an FTAM-3 document type. OMNI $FGET creates a file with RMS record format undefined and RMS record attributes of none.

OMNI$FRENAME

OMNI$FRENAME — Renames a file on the remote system.

Format

```
status=OMNI$FRENAME[_A] [invoke_id], vmd_handle,
original_remote_file_name, new_remote_file_name, [modifier_object],
[omni_iosb], [ctrl_struc]
```

OMNI$GET_ATTRIBUTE

OMNI$GET_ATTRIBUTE — Obtains: 1. A specific object attribute from a list of attributes obtained by the OMNI$GET_REMOTE_ATTRIBUTE procedure. 2. A file specification from a remote directory obtained by the OMNI$FDIR procedure.

Format

```
status=OMNI$GET_ATTRIBUTE context, object_attribute, receive_struc
```

Description

The receive structure is based on the type of attribute. The OMNI$GET_ATTRIBUTE procedure reads the first value of the specified type. To read the next value of the same type, call OMNI $GET_ATTRIBUTE with a NULL attribute.

OMNI$GET_DEFINITION

OMNI$GET_DEFINITION — Retrieves the value of a specified attribute of a specified definition and inserts the value in an address specified by the caller.

Format

```
tatus=OMNI$GET_DEFINITION (defhandle,attraddress, contextaddress,
value)
```

Description

OMNI$GET_DEFINITION retrieves the values of both single-valued and multivalued attributes.

In the case of multivalued attributes, OMNI$GET_ DEFINITION acts similar to OMNI $GET_HANDLE_LIST. Each call to either of the routines, returns one value.

The value of the ATTRIBUTE parameter should specify the address of the attribute on the first call, and should be NULL thereafter. A value of NULL for the ATTRIBUTE parameter indicates that the next value should be specified or retrieved. In the case of OMNI$GET_DEFINITION, when the last value has been retrieved, OMNI$_ENDOFLIST is returned as the status value. The OMNI$END_LIST routine should be called after a list of values has been retrieved.

OMNI$GET_HANDLE_BY_NAME

OMNI$GET_HANDLE_BY_NAME — Locates the handle of a specified object definition. The name used when the definition was created with ODF is supplied to identify the definition.

Format

```
status=OMNI$GET_HANDLE_BY_NAME scope, class, def_name, handle
```

OMNI$GET_HANDLE_LIST

OMNI$GET_HANDLE_LIST — Returns a definition handle of the specified class for the specified scope.

Format

```
status=OMNI$GET_HANDLE_LIST scope, class, context, handle
```

Description

When used as part of a loop, OMNI$GET_HANDLE_LIST can retrieve all of the handles of a specified class for the specified scope. OMNI$GET_HANDLE_LIST sets the value pointed to by the receiving handle to NULL before assigning it a valid value. Even if OMNI$GET_HANDLE_LIST returns an error, the value pointed to by the receiving handle can still be zero.

OMNI$GET_INDICATIONS

OMNI$GET_INDICATIONS — Receives the following indications from a remote application:Read/ write indications, Unsolicited status, Conclude indications, Abort indications, Information reports.

Format

```
status=OMNI$GET_INDICATIONS[_A] vmd_handle, def_handle, context,
indication_type, reserved, [omni_iosb], [ctrl_struc]
```

Description

Issue one OMNI$GET_INDICATIONS call per remote VMD.

OMNI$GET_REMOTE_ATTRIBUTES

OMNI$GET_REMOTE_ATTRIBUTES — Obtains a list of current attribute values for an object on a remote MMS system.

Format

```
status=OMNI$GET_REMOTE_ATTRIBUTES[_A][invoke_id], def_handle, class,
context, [modifier_object], [omni_iosb], [ctrl_struc]
```

OMNI$GET_VALUE

OMNI$GET_VALUE — As a client procedure, obtains the value of a variable on a remote VMD. As a server procedure, OMNI$GET_VALUE obtains the value referred to by a write indication.

Format

```
status=OMNI$GET_VALUE[_A] [invoke_id], object_handle,
[method_handle], receiving_struc, [modifier_object], [omni_iosb],
[ctrl_struc]
```

OMNI$GROUP_VARIABLES

OMNI$GROUP_VARIABLES — Allows to read or write multiple variables.

Format

```
status=OMNI$GROUP_VARIABLES(contextaddress, object_handle,
method_handle, pdata,modifier_object,omni_iosb);
```

Description

You can use this procedure by following these steps:

1. Set the context variable to be used to zero.
2. Call the OMNI$GROUP_VARIABLES with the pointer to the context, variable handle, alternate access handle, pointer to data, modifier handle, and pointer to ISOB for one variable.
3. Repeat step 2 as many times as necessary for different variables, using the same context.
4. Call the OMNI$GET_VALUE[_A] or OMNI$PUT_ VALUE[_A], substituting the context value for the variable handle, and omitting the object handle, method handle, pointer to data, and modifier handles.
5. The group context is in effect until an OMNI$END_LIST with the context specified is done.

The IOSB that is passed in each time to the OMNI$GROUP_VARIABLES function indicates whether the transaction was successful on the variable.

If variables are grouped together and the user passes to OMNI$GET_VALUE[_A] or OMNI $PUT_VALUE[_A] a pointer to an IOSB, that IOSB represents the general transaction completion.

OMNI$INITIALIZE

OMNI$INITIALIZE — Sets up VSI OMNI data structures and specifies values for the following operating parameters: 1. Vendor name (the default is VSI). 2. Model name (the default is VSI OMNI). 3. Revision name.(the default is V1.0).

Format

```
tatus=OMNI$INITIALIZE [vendor], [model], [revision]
```

OMNI$KILL

OMNI$KILL — Ends a program invocation on a VMD by causing it to transition to the unrunnable state.

Format

```
status=OMNI$KILL[_A] [invoke_id], pi_handle, [modifier_object],
[omni_iosb], [ctrl_struc]
```

OMNI$LISTEN

OMNI$LISTEN — Receives an association request from a remote application.

Format

```
status=OMNI$LISTEN[_A] called_vmd_handle, translate_flag,
calling_vmd_handle, [incoming_vmd_struc], [omni_iosb], [ctrl_str]
```

OMNI$LOAD_DEFINITIONS

OMNI$LOAD_DEFINITIONS — Loads the definitions that have been created by ODF for one VMD object and related objects.

Format

```
status=OMNI$LOAD_DEFINITIONS vmd_name, vmd_handle
```

OMNI$MODIFY_DEFINITION

OMNI$MODIFY_DEFINITION — Modifies the value of a specified attribute of a specified definition. The address of the new attribute value is passed as a parameter to the routine.

Format

```
status=OMNI$MODIFY_DEFINITION(defhandle, attraddress,
contextaddress, value)
```

Description

OMNI$MODIFY_DEFINITION modifies the values of both single-valued and multivalued attributes. In the case of multivalued attributes, OMNI$MODIFY_DEFINITION is used much like OMNI $GET_HANDLE_LIST. Each call to OMNI$MODIFY_DEFINITION specifies one value. The value of the ATTRIBUTE parameter should specify the address of the attribute on the first call, and should be NULL thereafter. A value of NULL for the ATTRIBUTE parameter indicates that the next value should be specified.

OMNI$END_LIST should be called after modifying a list to free space allocated for bookkeeping by VSI OMNI.

A definition is not usable until the value of its SCOPE attribute has been modified. The scope of a definition can be modified only once, and each class of definition must have a particular set of attributes modified before its scope can be modified. Modification of the value of a definition's scope is equivalent to the committal of that definition in ODF.

OMNI$PUT_VALUE

OMNI$PUT_VALUE — As a client procedure, modifies the value of a variable on a remote VMD. As a server procedure, OMNI$PUT_VALUE transmits the value of the variable specified by a read indication.

Format

```
status=OMNI$PUT_VALUE[_A] [invoke_id], object handle,
[method_handle], value_struc, [modifier_object], [omni_iosb],
[ctrl_struc]
```

OMNI$REJECT

OMNI$REJECT — Rejects an indication you do not want.

Format

```
status=OMNI$REJECT[_A] context,[reason], [omni_iosb], [ctrl_struc]
```

OMNI$REJECT_CONCLUDE

OMNI$REJECT_CONCLUDE — Rejects an association conclude request from a remote application. Call the OMNI$REJECT_CONCLUDE procedure in response to an indication returned by OMNI $GET_INDICATIONS.

Format

```
status=OMNI$REJECT_CONCLUDE[_A] vmd_handle, [reason], [omni_iosb],
[ctrl_struc]
```

OMNI$REJECT_CONNECT

OMNI$REJECT_CONNECT — Rejects an association request from a remote VMD. Call the OMNI $REJECT_CONNECT procedure in response to an association request returned by OMNI$LISTEN.

Format

```
status=OMNI$REJECT_CONNECT[_A] vmd_handle, [reason], [omni_iosb],
[ctrl_struc]
```

OMNI$RESET

OMNI$RESET — Resumes execution of a stopped program on the VMD. It causes a program invocation that is in the stopped state to transition to either the idle or unrunnable state. If the PI is reusable, it transitions to the idle state; otherwise, it transitions to the unrunnable state.

Format

```
status=OMNI$RESET[_A] [invoke_id], pi_handle, [modifier_object],
[omni_iosb], [ctrl_struc]
```

OMNI$RESUME

OMNI$RESUME — Causes a program invocation to transition from the stopped state to the running state.

Format

```
status=OMNI$RESUME[_A] [invoke_id], pi_handle, [execution_arg],
[modifier_object], [omni_iosb], [ctrl_struc]
```

OMNI$START

MNI$START — Causes a program invocation to transition from the idle to the running state.

Format

```
status=OMNI$START[_A] [invoke_id], pi_handle, [execution_arg],
[modifier_object], [omni_iosb], [ctrl_struc]
```

OMNI$STOP

OMNI$STOP — Causes a program invocation to transition from the running state to the stopped state.

Format

```
status=OMNI$STOP[_A] [invoke_id], pi_handle, [modifier_object],
[omni_iosb], [ctrl_struc]
```

OMNI$OMNI_TO_VMS_TIME

OMNI$OMNI_TO_VMS_TIME — Converts an OMNI time to a VMS time.

Format

```
status=OMNI$OMNI_TO_VMS_TIME[_A] OMNI_Time, VMS_Time
```

OMNI$UPLOAD

OMNI$UPLOAD — Preforms the uploading of a domain from a remote VMD. Two files are created, one with the list of capabilities and one with the domain contents. The list of capabilities file name is a mandatory field in ODF and is the name that is used for the upload. If an error occurs during the upload process, the files are deleted.

Format

```
status=OMNI$UPLOAD[_A] [invoke_id], domain_handle, [domain_file],
[modifier_object], [omni_iosb], [ctrl_struc]
```

Chapter 2. OMNI Definition Facility

The OMNI Definition Facility (ODF) enables you to create and manage locally stored definitions of MMS objects. Specifically, ODF provides a set of commands that perform the following operations:

- Create definitions of VMDs.
- Create definitions of MMS domains and associate the definitions with a locally defined VMD.
- Create definitions of MMS program invocations and associate the definitions with a locally defined VMD.
- Create definitions of variables and associate the definitions with a locally defined domain or VMD.
- Create data type definitions.
- Display the local definitions of an MMS object.
- Delete a locally created definition or set of definitions.
- Log the current ODF session to a file for later use.
- Write (export) definition commands for backup or convenience.
- Execute a series of stored commands – for example, commands saved in a log file.
- Set and display the defaults for an ODF session.

Note

The definitions you create with ODF are local to VSI OMNI but are not necessarily local to the system running ODF or using the definitions.

2.1. ODF and Companion Standards

A Companion Standard (CS) can function as an integral part of VSI OMNI and can be defined by using ODF.

Note that if a CS exists with VSI OMNI, it can affect the behavior of the VSI OMNI procedure calls, since a CS can support objects and attributes that are different from those supported by VSI OMNI.

See your applicable companion standard's guide for details about the objects and attributes supported by that companion standard.

2.2. ODF Command Language Interface

The Command Language Interface (CLI) guides you through the correct syntax of each ODF command by supplying prompts and a list of options.

For example, suppose you want to use the SET command, but you cannot remember the exact syntax or choices of the command. Simply type in the SET command followed by a carriage return, as shown:

```
ODF>SET Return
```

Because the command has been entered in incomplete form, CLI automatically prompts for the next word in the command. Only those that support the SET command are listed as options. All options are enclosed within parentheses, as follows: (COMPANION STANDARD, ODF LOGFILE, SCOPE) _ODF>

2.2.1. Level-by-Level Prompting

You can specify the entire command without using CLI, or you can specify part of the command and have CLI prompt only for those words that you miss.

Because CLI displays only supported options, prompting for options is a good way to check the syntax of a command after receiving a parser error. Any attribute or keyword you specify that is not in the CLI list of options is not supported for that command.

2.2.2. Short Lines and Abbreviations

You can shorten the command line by shortening the number of letters in each word. You can abbreviate any word to three characters or the number of characters that makes it unique.

2.3. Invoking and Exiting ODF

You can issue ODF commands one at a time using Digital Command Language (DCL), or you can invoke ODF and issue as many commands as you want before returning to the DCL prompt ($).

To use ODF for single line commands, first define ODF as follows:

```
$ ODF :== $OMNI$ODF
```

ODF executes the command and returns you to the DCL system prompt. To invoke ODF through DCL for an interactive session, type:

```
$ ODF
ODF>
```

Once invoked, the ODF prompt is displayed on the screen. Issue the first command next to the prompt. If you leave out required component-ids or attributes, ODF prompts for them.

To invoke ODF for an interactive session without using DCL, use the RUN command as follows:

```
$ RUN SYS$SYSTEM:OMNI$ODF
ODF>
```

To exit ODF and return to the DCL system prompt, either issue the EXIT command or press **Ctrl/z**.

2.4. Getting Help Through ODF

After you invoke ODF, you can use the HELP command to display quick reference information about individual commands. Type HELP and the name of the command you want information about as shown in the following example, or type HELP followed by a carriage return to receive a menu of options. For example:

```
ODF> HELP SET SCOPE
```

A display of HELP information about the SET SCOPE command is returned when this command executes.

2.5. Creating a Definition of a VMD

A complete VSI OMNI definition of a VMD consists of the items in Table 2.1

Table 2.1. VMD Definitions

Item	Description
vmd_name	The local name of the VMD definition. This name is used to reference the definition; it is not used in communications.
APPLICATION SIMPLE NAME	The name used to look up the application in Directory Services.
VERSION	The version of the MMS protocol to use.
NESTING LEVEL	The maximum number of levels of nesting that can occur within any data element over an association with the VMD.
MAXIMUM SERVICES CALLED	The proposed maximum number of transaction object instances that can be created at the called MMS-user on the association.
MAXIMUM SERVICES CALLING	The proposed maximum number of transaction object instances that can be created at the calling MMS-user on the association.
MAXIMUM SEGMENT SIZE	The proposed maximum size of an MMS message to exchange with the VMD.
PARAMETER CBB	A list specifying the set of conformance building blocks (CBBs) supported by the VMD.
SUPPORTED SERVICES	A list of services supported by the VMD for the association.
VENDOR	The vendor of the system supporting the VMD.
MODEL	The model of the system supporting the VMD.
REVISION	A string describing the software, firmware, or hardware revision level of the VMD.
DESCRIPTION	Information describing the defined VMD. This is not used in communication.

To create a VSI OMNI definition of a VMD, enter the DEFINE VMD COMMAND, specify the name of the VMD, and supply the values that describe the VMD. To add the definition to the permanent ODF database, enter the COMMIT command. COMMIT is described in the *VSI OMNI Network Manager's Guide* .

2.6. Creating a Definition of a Domain

A complete VSI OMNI definition for a domain consists of the elements in Table 2.2

Table 2.2. Domain Definitions

Item	Definition
vmd_name:domain_name	The name of the remote domain object and its associated VMD.
[NO] DELETABLE	A value indicating whether the domain can be deleted from the VMD.
[NO] SHARABLE	A value indicating whether the domain can be shared by multiple program invocations.
CONTENT FILE	A VMS file containing the domain.
CAPABILITY FILE	A VMS file specifying the capabilities of the domain.
DESCRIPTION	Information describing the defined domain.

A domain definition must include the name of the VMD to which the domain belongs. ODF will reject any domain definition that does not specify an existing VMD and a capabilities file.

To create a VSI OMNI definition of a domain, enter the DEFINE DOMAIN command in response to the ODF prompt and supply the information you need to describe the domain. To add the definition to the permanent ODF database, enter the COMMIT command.

2.7. Creating a Definition of a Program Invocation

A complete VSI OMNI definition of a Program Invocation (PI) contains the information listed in Table 2.3

Table 2.3. PI Definition

Item	Description
vmd_name:pi_name	The name of the program invocation and its associated VMD.
[NO] DELETABLE	A value indicating whether the PI can be deleted from the VMD.
[NO] REUSABLE	A value indicating whether the PI can be reused.
EXECUTION ARGUMENT STRING	An execution argument that becomes the default for START and RESUME requests for the PI.
monitor_type	One of three values. NO MONITOR indicates that the PI has no monitoring event condition. MONITOR PERMANENT indicates that the PI has a monitoring event condition that exists throughout program execution. MONITOR CURRENT indicates that the PI has a monitoring event condition that exists only for the life of the association.
DOMAIN LIST	A list of references to the domains that make up this program invocation.

Item	Description
DESCRIPTION	Information describing the defined Program Invocation.

A PI definition must include the name of the VMD to which the domain belongs. ODF will reject any PI definition that does not specify an existing VMD.

Each PI definition must also specify a domain list with at least one domain. ODF will reject the definition if the listed domains are not defined.

To create a VSI OMNI definition of a program invocation, enter the DEFINE PROGRAM INVOCATION command in response to the ODF prompt and supply the values that describe the PI. To add the definition to the permanent ODF database, enter the COMMIT command.

2.8. Creating a Definition of a Variable

ODF enables you to create VSI OMNI definitions for the following types of variables:

- Named variables
- Unnamed variables

2.8.1. Named Variables

A complete VSI OMNI definition of a Named Variable contains the items listed in Table 2.4

Table 2.4. Named Variable Definition

Item	Description
vmd_name:domain_name.variable_name	The name of the remote named variable object and its associated VMD and (optionally) domain.
type	A reference to a predefined application type.
[NO] DELETABLE	A value that indicates whether the named variable can be deleted from the VMD.
DESCRIPTION	Information that describes the named variable.

A variable definition must include the name of the VMD to which the variable belongs. ODF will reject any definition that does not specify an existing VMD. If a variable is defined as being on a domain, you must also define the domain.

You must specify the type of the variable.

To create a VSI OMNI definition for a named variable, enter the DEFINE NAMED VARIABLE command in response to the prompt and supply the required information. To add the definition to the permanent ODF database, enter the COMMIT command.

2.8.2. Unnamed Variables

A complete VSI OMNI definition of an Unnamed Variable contains the items listed in Table 2.5.

Table 2.5. Unnamed Variable Definition

Item	Description
vmd_name:domain_name.variable_name	The name of the remote unnamed variable and its associated VMD (and, optionally, its domain).
type	A reference to a predefined application type.
<address>	The address of the unnamed variable.
[NO] Supply Type Spec	A value that indicates whether the variable's type specification is to be sent to the remote VMD to access the variable.
DESCRIPTION	Information describing the unnamed variable.

A variable definition must include the name of the VMD to which the variable belongs, the variable type, and the address. ODF will reject any definition that does not specify an existing VMD, the variable type, and the address.

The variable address may be specified as a NUMERIC ADDRESS or a SYMBOLIC ADDRESS. A numeric address value is entered as a decimal number by default or as a hexadecimal number using the %X prefix. A symbolic address value is entered as a quoted string.

To create a VSI OMNI definition for an unnamed variable, enter the DEFINE UNNAMED VARIABLE command in response to the prompt and supply the required information. To add the definition to the permanent ODF database, enter the COMMIT command.

2.9. Defining Variable Types

An ODF variable definition includes two variable type definitions: an MMS type definition and an application type definition.

- The MMS type definition provides information about the variable that is communicated through the MMS protocol when the variable is read or written.
- The application type definition provides information about the way the application views the variable. Application type information cannot be communicated through the MMS protocol – it is specific to the local programming environment.

ODF provides two commands that you can use to create variable type definitions:

- DEFINE MMS NAMED TYPE. Creates an MMS type definition.
- DEFINE APPLICATION TYPE. Creates an application type definition and associates the definition with a corresponding MMS type definition that you have created.

The DEFINE TYPE commands are useful for creating commonly-used type definitions that many variables will reference. When a number of variables refer to the same type definition, all of the variables can be changed by changing the one type definition.

2.9.1. Creating an MMS Named Type Definition

A complete VSI OMNI definition of an MMS Named Type contains the items listed in Table 2.6.

Table 2.6. MMS Named Type Definition

Item	Description
vmd_name:domain_name.mms_type_name	The name of the MMS Named Type specification and its associated VMD (and, optionally, its domain).
mms_type_specification	A structure, array or simple type specification or a reference to another MMS Named Type.
[[NO]DELETABLE]	Indicates whether or not the MT can be deleted from the VMD.
DESCRIPTION	Information describing the defined MMS Named Type.

An MMS Named Type definition must include the name of the VMD to which the named type belongs. ODF rejects any definition that does not specify an existing VMD. If a named type is defined as being on a domain, you must also specify the domain.

You must specify the MMS type specification.

To create a VSI OMNI definition for an MMS Named Type, enter the DEFINE MMS Named Type command in response to the prompt and supply the required information. To add the definition to the permanent database, enter the COMMIT command. COMMIT is described in Section 2.10

2.9.2. Creating an Application Named Type Definition

A complete VSI OMNI definition of an Application Named Type contains the items listed in Table 2.7.

Table 2.7. Application Named Type Definition

Item	Description
vmd_name:domain_name.application_type_name	The name of the Application Named Type specification and its associated VMD (and, optionally, its domain).
FROM MMS NAME TYPE	The name of the MMS Named Type associated with the application named type. The default is the same name and scope as the application type.
application_type_specification	A structure, array or simple type specification or a reference to another Application Named Type.
DESCRIPTION	Information describing the defined Application Named Type

An Application Named Type definition must include the name of the VMD to which the named type belongs. ODF will reject any definition that does not specify an existing VMD. The named type may also be defined on a domain.

You must specify the application type specification.

To create a DECOmni definition for an Application Named Type, enter the DEFINE Application Named Type command in response to the prompt and supply the required information. To add the definition to the permanent database, enter the COMMIT command. COMMIT is described in Section 2.10.

2.9.3. Creating Application Type Definitions for Alternate Access

Every VSI OMNI variable definition specifies a default Application Type Definition, which in turn refers to an MMS Type Definition.

Simple applications would generally access the variable's data using the default Application Type. Other applications may need to perform alternate access by referring to the variable using some other Application or MMS Type Definition or both.

One reason for alternate access would be to support applications which store internal data differently. For example, suppose two applications access a variable whose MMS Type Definition is a Visible String. One application may need to store this visible string internally as a null terminated string while another application type may need to store it internally as a word counted string. In both cases, since all elements of the array would be accessed, there is a 1:1 correspondence between array components of the MMS Type Definition and the Application Type Definition.

Another reason for alternate access is to support applications that may not need to access all of the data in a variable. This type of alternate access is called partial access. For example, a device can define a portion of its memory as a large array.

An application can read the portion of the memory it is interested in by creating an Application Type Definition that specifies a subrange of the array to be read into an application buffer which is only large enough to hold the data in that subrange.

In both examples of alternate access (full and partial), an application accomplishes alternate access on a variable by providing a method handle in calls to the variable access procedures, OMNI$GET_VALUE and OMNI$PUT_ VALUE. A method handle is an object identifier handle of an Application Named type. For information on method handles see the online *VSI OMNI Application Programmer's Guide*.

You can also define the default application type as an alternate access type. In this case, it is not necessary to supply a method handle to perform alternate access. Instead, alternate access is by default whenever the variable is accessed.

2.10. Committing Definitions to the ODF Database

An ODF session consists of the following steps:

1. Enter a series of DEFINE or DELETE commands, or both to describe the objects in the MMS environment. ODF saves these definitions in a special area allocated for the ODF session.

2. Enter the COMMIT command. ODF examines the batched definitions (that is, all the definitions entered since the last COMMIT command or since the beginning of the session), writes all valid definitions into the permanent database, and reports on any errors. If committing the changes will produce inconsistencies in the database, ODF reports an error and does not make any of the modifications.

   For example, if you enter a variable definition that includes a reference to a nonexistant VMD, ODF will reject the definition and return an error code.

ODF does not discard the batch of definitions if the COMMIT operation fails. Thus, you can correct the error and COMMIT again.

To erase a batch of modifications from the temporary storage area, type the ROLLBACK command. ODF discards all the definitions that you have created since your last COMMIT command. (Note how ROLLBACK differs from DELETE. The DELETE command removes a definition that has been committed to the permanent ODF database and/or exists in temporary storage; the ROLLBACK COMMAND simply discards actions from temporary storage.)

In addition to batching DEFINE commands, ODF batches all commands that modify the database (for example, the DELETE command) until you enter a COMMIT command.

Note

The EXIT command causes ODF to attempt a COMMIT before exiting. The QUIT command causes ODF to attempt a ROLLBACK before exiting.

You should try to arrange your transactions so that a COMMIT can be issued after each DEFINE command, which should reduce the ambiguity of constraint error messages.

2.11. Setting the Default Scope

ODF enables you to set the default VMD and domain for dependent objects that you want to define. To specify a default VMD and domain, enter the SET SCOPE command and the name of the VMD and domain. (If you omit the domain name, the scope is VMD-specific.)

For example, the following SET SCOPE command specifies Foo as the default VMD for the session and Bar as the default domain. The DEFINE command creates a variable definition named *X* :

```
ODF> SET SCOPE Foo:Bar
ODF> DEFINE NAMED VARIABLE X APPLICATION TYPE %OMNI$LONG;
```

ODF creates the definition Foo:Bar.X.

2.12. Deleting a Definition

The DELETE DEFINITION command deletes a definition from the permanent ODF database and/or temporary storage.

A definition cannot be deleted until all its dependencies are deleted.

A right arrow character (>) in the command line causes ODF to delete the specified object and all objects that are dependent on that object. For example, the following command deletes VMD Foo and all the objects it contains:

```
DELETE DEFINITION Foo>;
```

To delete the entire database, type:

```
DELETE DEFINITION *>;
```

This command line is not recommended.

The DELETE command supports the wildcard asterisk (*). For example, the following command deletes all named variables in domain Foo:Bar:

```
DELETE DEFINITION Foo:Bar(NV:*);
```

2.13. Creating, Opening, and Closing a Log File

ODF enables you to create and open a log file for the ODF session. To create a log file, enter the SET ODF LOGFILE command and specify the name of the log file.

Once the file is open, you can start session logging with the ENABLE ODF LOGGING command. You can also write definition commands to the log file using the WRITE DEFINITION command.

To close the log file, reenter the SET ODF LOGFILE command with a different filename or the null device name (NL:).

2.14. Enabling and Disabling Logging

To create a log of the ODF session, enter the ENABLE ODF LOGGING command.

If you have already specified a log file with the SET ODF LOGFILE command, ODF logs the session to that file. If you have not specified a file, ODF creates a file OMNI$ODF.LOG in the current default directory and logs the session.

To disable logging, enter the DISABLE ODF LOGGING command. Logging will stop, but the log file will remain open until you exit the ODF session or enter another SET ODF LOGFILE command.

The ENABLE and DISABLE ODF LOGGING commands can be used to selectively log portions of an ODF command session.

2.15. Executing Stored Commands

The DO command (or @) enables you to read ODF commands stored in a script file.

You can create script files in any of the following ways:

- Use the SET ODF LOGFILE and ENABLE ODF LOGGING commands to trace a session.
- Use the WRITE DEFINITION TO command to write declaration commands to a file. This is helpful when creating script files to rebuild portions of the database.

  Issuing a WRITE DEFINITION command without a TO specifier causes a DEFINE command to be written to the current log file.
- Use any editor to create a text file containing the commands.

If logging is enabled when the script file is invoked, the invocation command is commented out in the trace, and the individual commands in the script file appear in the trace output. A comment is inserted at the end of the trace file. If the script file contains an EXIT command, that command does not appear in the trace file.

DO commands can be nested (a script file can issue a DO command). There is no limit to how many DO commands can be issued from a particular script file; however, ODF must open each script file, so the open file limit (FILLM) quota determines the maximum nesting allowed.

2.16. Creating a Command to Repeat a Definition

The WRITE DEFINITION command enables you to write out definitions to a file, where each definition is written as a valid ODF DEFINE command. A reference to a definition, list of definitions, or a wildcard specification can be specified. An asterisk (*) used as a wildcard character matches zero or more characters, and a period (.) used as a wildcard character matches exactly one character.

If you include a file specification by using the TO clause, ODF opens that file, writes the definitions to it, and closes the file. If there is no file specification, ODF appends the definitions to the current log file. If no log file is open, ODF opens a new version of OMNI$DEF.LOG and writes the definitions.

The following command writes out all definitions in the database to a file named BACKUP.LOG:

```
WRITE DEFINITION *> TO BACKUP.LOG;
```

The following example writes out domain Bar of VMD Foo and its dependent objects to file DOMAIN.LOG:

```
WRITE DEFINITION Foo:Bar> TO DOMAIN.LOG;
```

The following example writes out named variables defined in domain Bar to the current log file:

```
WRITE DEFINITION Foo:Bar(NV:*);
```

2.17. Exiting and Quitting an ODF Session

The EXIT command attempts to perform a COMMIT before ending the ODF session. If there are unresolved dependencies, ODF does not EXIT. Enter additional DEFINE commands to satisfy the dependencies, and reenter the EXIT command.

The QUIT command rolls back any batched DEFINE or DELETE DEFINITION commands and ends the ODF session.

Chapter 3. OMNI Definition Facility (ODF) Commands

This chapter describes the set of commands you issue to create and manage local VSI OMNI definitions of remote MMS objects.

These sections use the documentation conventions listed in Table 3.1

Table 3.1. Conventions

Convention	Meaning
[ ]	Square brackets enclose optional expressions.
8 < : <val1> <val2> <val3> 9 = ;	
	Large braces enclose choices from a group of items. Braces around a single item indicate that this item is mandatory.
< >	Angle brackets enclose tokens that must be expanded.
. . .	Ellipsis indicates an expression that can be repeated.

A local ODF definition name has the same format as an MMS identifier: a string of 1 to 32 characters. All alphanumeric characters, the dollar sign ($), and the underscore (_) are valid. The identifier cannot begin with a numeric character. Also, ODF definition names, like MMS identifiers, are case-sensitive (foo is not equal to FOO).

COMMIT

COMMIT — Makes changes to the database. All changes made in an ODF session since the last COMMIT become permanent and are made visible to other users of the ODF database.

Format

```
COMMIT;
```

Description

When you enter COMMIT, ODF processes all of the DEFINE and DELETE DEFINITION commands you have entered since your last COMMIT command or since the beginning of the session.

Before the modifications are made visible, ODF verifies that those modifications leave the database in a consistent state. If committing the command would cause an inconsistency, ODF reports a constraint violation, and the changes are not added to the database.

To recover from a constraint violation, either roll back the commands or enter additional commands to correct the problem. The SHOW DEFINITION command is useful for pinpointing the cause of the problem. SHOW DEFINITION shows any uncommitted changes as if they had already been applied.

DEFINE DOMAIN

DEFINE DOMAIN — Creates a definition of a domain and associates the domain with a VMD definition.

Format

```
DEFINE DOMAIN [<vmd_name>:]<domain_name> [[NO] DELETABLE] [[NO]
SHARABLE] [CONTENT FILE <content_filespec>] [CAPABILITY FILE
<capability_filespec>] [DESCRIPTION <text>];
```

DEFINE MESSAGE

DEFINE MESSAGE — Creates a local VSI OMNI definition of a message object and associates it with a VMD previously defined.

Format

```
DEFINE MESSAGE [<vmd_name>:]<msg_name> LENGTH <msg_length>
[DESCRIPTION <text>];
```

DEFINE PROGRAM INVOCATION

DEFINE PROGRAM INVOCATION — Creates a definition of a program invocation and associates the PI with a VMD definition.

Format

```
DEFINE PROGRAM INVOCATION [<vmd_name>:]<pi_name> [DESCRIPTION
<text>] DOMAIN LIST <dom_id>[,<dom_id>] . . . [[NO] DELETABLE] [[NO]
REUSABLE] [[NO]EXECUTION ARG STRING <text>];
```

DEFINE NAMED VARIABLE

DEFINE NAMED VARIABLE — Creates a VSI OMNI definition of a named variable and associates the definition with a defined domain or VMD.

Format

```
DEFINE NAMED VARIABLE [<vmd>:][<dom>.]<var> {<type>} [[NO]
DELETABLE] DESCRIPTION <text>;
```

DEFINE UNNAMED VARIABLE

DEFINE UNNAMED VARIABLE — Creates a VSI OMNI definition of an unnamed variable object and associates the definition with a defined domain or VMD.

Format

```
DEFINE UNNAMED VARIABLE [<vmd>:]<var> <type> <address> [[NO] Supply
Type Spec] [DESCRIPTION <text>];
```

DEFINE APPLICATION NAMED TYPE

DEFINE APPLICATION NAMED TYPE — Creates an application named type definition.

Format

```
DEFINE APPLICATION NAMED TYPE [<vmd>]: [<dom>.]<type> [FROM MMS
NAMED TYPE <ref>] <app_type_specification> [DESCRIPTION <text>];
```

DEFINE VMD

DEFINE VMD — Creates a local VSI OMNI definition of a VMD.

Format

```
DEFINE VMD <vmd_name> APPLICATION SIMPLE NAME <app_simple_name>
[VERSION <version_number>] [NESTING LEVEL <word_value>] [MAXIMUM
SERVICES CALLED <word_value>] [MAXIMUM SERVICES CALLING
<word_value>] [MAXIMUM SEGMENT SIZE <integer_value>] [PARAMETER
CBB <cbb_list>] [SUPPORTED SERVICES <supported_service_list>] [[NO]
VENDOR <vendor_name>]
```

```
DEFINE VMD <vmd_name> [[NO] MODEL <model>] [[NO] REVISION
<revision>] [DESCRIPTION <text>];
```

DELETE DEFINITION

DELETE DEFINITION — Removes a definition from the database. A definition cannot be deleted until all its dependent definitions have been deleted.

Format

```
DELETE DEFINITION <def_ref>[,<def_ref>] . . . ;
```

DISABLE

DISABLE — Stops logging of the current ODF session. The logfile is not closed. To close the file, issue a SET ODF LOGFILE command or exit the session.

Format

```
DISABLE ODF LOGGING;
```

DO

DO — Executes a series of stored commands, such as those saved in a script file by the ENABLE command. DO is a synonym for @.

Format

```
DO <script_file>;
```

ENABLE

ENABLE — Turns on OMNI logging to the log file specified in the most recent SET ODF LOGFILE command. If no log file has been set since the start of the ODF session, ODF tries to create a file OMNI $ODF.LOG in the current default directory and log commands to that file.

Format

```
ENABLE ODF LOGGING;
```

EXIT

EXIT — Commits any outstanding changes to the database and exits ODF. If the outstanding changes are invalid, ODF reports an error and does not exit.

Format

```
EXIT;
```

QUIT

QUIT — Cancels all the DEFINE and DELETE commands you have entered since the last COMMIT command and exits from the session. No definition data from the cancelled commands is written into the database.

Format

```
QUIT;
```

ROLLBACK

ROLLBACK — Cancels all the DEFINE and DELETE DEFINITION commands you have entered since the last COMMIT command. No definition data is written into the database from the commands within the range of the rollback.

Format

```
ROLLBACK;
```

SET ODF LOGFILE

SET ODF LOGFILE — Specifies the file to which ODF logs the session.

Format

```
SET ODF LOGFILE <vms_file_specification>;
```

Description

The log file is opened immediately, but logging is disabled until you enter an ENABLE ODF LOGGING command. If a log file is already open, ODF closes the file before attempting to open the new file. If logging is currently enabled, ODF issues an implicit DISABLE ODF LOGGING command.

SET COMPANION STANDARD

SET COMPANION STANDARD — Sets the ODF command syntax (if the companion standard extends the syntax) and the default companion standard name for definitions created under ODF. A VMD and its dependencies must all be defined under the same companion standard.

Format

```
SET COMPANION STANDARD <companion_standard_name>;
```

SET SCOPE

SET SCOPE — Specifies the definitions that ODF uses as the default scope. The scope of an ODF command is the VMD or domain definition or both with which the command is associated.

Format

```
SET SCOPE [<vmd_name>] [:<domain_name>];
```

SHOW

SHOW — Displays current ODF session settings.

Format

```
SHOW <setting> ;
```

SHOW DEFINITION

SHOW DEFINITION — Displays definitions from the database on the terminal. If modifications have been made, but not committed, they will also be visible.

Format

```
SHOW DEFINITION <def_ref> [, <def-ref>]<hellipsis>;
```

WRITE DEFINITION

WRITE DEFINITION — Writes out definitions to a file. Each definition is written as a valid ODF command. If you include a file specification, ODF opens the file, writes the definitions, and closes the file. If you omit the file specification, ODF appends the definitions to the current log file. (If there is no open log file, ODF opens a new version of OMNI$ODF.LOG and writes the definitions.)

Format

```
WRITE DEFINITION <def_ref>[, <def_ref>] . . . [TO <filespec>];
```

Chapter 4. OMNI Command Language

One user interface to VSI OMNI network management is the OMNI Command Language (OMNICL). OMNICL consists of a set of commands that enable you to read and monitor data on the OMNI system. In this chapter, the OMNICL features and commands are described.

4.1. OMNICL Command Syntax

OMNICL commands contain the following elements:

- keyword
- component-id
- attribute
- value

For example:

```
SET OMNICL LOGGING OUTPUT OPCOM
```

The keyword describes the operation you want to perform and the component-id describes the major component affected by the command.

Most commands also have several attribute options to further qualify the action of the command. Also, if you plan to change the value of an attribute, you must enter the new value in the command line.

In this chapter, each command is listed by keyword and component-id. A description of the command and the correct format are given and then any attribute for that command is listed under "Attributes". Variables you need to enter are identified with their associated attributes, but are described separately in the area marked "Values".

4.1.1. OMNICL Command Language Interface

The Command Language Interface (CLI) guides you through the correct syntax of each OMNICL command using prompts and a list of options for each keyword and attribute level.

For example, suppose you want to use the SET command but cannot remember the exact syntax or choices of the command. Simply type in the SET command followed by a carriage return, as shown:

```
OMNICL> SET
```

Because the command has been entered in incomplete form, CLI automatically prompts for the next word in the command, which is the name of the component-id. Only those ids that support the SET command are listed as options. All options are enclosed within parentheses, as follows:

```
* (EVENT, OMNICL, DEFAULT, NODEFAULT)? OMNICL LOGGING
```

4.1.1.1. Level-by-Level Prompting

You can specify the entire command without using CLI, or you can specify part of the command and have CLI prompt only for those words that you miss.

Because CLI displays only supported options, prompting for options is a good way to check the syntax of a command after receiving a parser error. Any attribute or keyword you specify that is not in the CLI list of options is not supported for that command.

4.1.1.2. Short Lines and Abbreviations

You can shorten the command line by shortening the number of words you specify and the number of letters in each word. You can abbreviate any word to the minimum number of letters that make it unique. This is usually three letters. (You will receive an error message if the parser finds the term ambiguous.)

Once unique words are encountered in the command line, you do not have to enter remaining keywords or attributes to execute the command. CLI automatically assumes the missing words.

4.2. Invoking and Exiting OMNICL

Before you can invoke OMNICL, you must confirm that your VSI OMNI license is correctly registered and loaded. If it is not registered and loaded, invocation will fail. Refer to the *VAX License Management Utility Reference Manual* for details about license management.

You can issue OMNICL commands one at a time using Digital Command Language (DCL), or you can invoke OMNICL and issue as many commands as you want before returning to the DCL prompt ($).

To use DCL for single line commands, first define OMNICL as follows:

```
$ OMNICL :== $OMNI$CL
```

To issue a single OMNICL command to DCL, type:

```
$ OMNICL command "attributes . . . "
$
```

OMNICL executes the command and returns you to the DCL system prompt. To invoke OMNICL through DCL for an interactive session, type:

```
$ OMNICL
OMNICL>
```

Once invoked, the OMNICL prompt is displayed on the screen. (When specifying commands directly to OMNICL rather than DCL, do not use foreign command format.) Issue the first command next to the prompt. If you leave out required component-ids or attributes, OMNICL prompts for them.

To invoke OMNICL for an interactive session without using DCL, use the RUN command as follows:

```
$ RUN SYS$SYSTEM:OMNI$CL
OMNICL>
```

To exit OMNICL and return to the DCL system prompt, issue either the EXIT command or press **Ctrl/z**.

4.3. Getting Help Through OMNICL

After you invoke OMNICL, you can use the HELP command to display quick reference information about individual commands. Type HELP and the name of the command if you want information as shown in the following example, or type HELP followed by a carriage return to receive a menu of options. For example:

```
OMNICL> HELP SET OMNICL LOGGING
```

A display of HELP information on the SET OMNICL LOGGING command is returned when this command executes.

OMNI Commands

SET DEFAULT

SET DEFAULT — Establishes the default application entity for subsequent commands. If SET NODEFAULT is entered, the default application entity is cleared.

Format

```
SET [NO]DEFAULT <application_entity>
```

Attributes

None.

Values

<application_entity>

Specifies the application entity name. The name must be known to Directory Services or a fully qualified network address. The name is a character string.

SET EVENT LOGGING

SET EVENT LOGGING — Defines the events to be logged and where those events will be logged. OMNI events that you log are separate from DNA, OSAK, and VOTS events. Events you can log are on a system-wide level; no application or association- specific events can be logged. By default, logging is disabled.

Format

```
SET EVENT LOGGING [OUTPUT= <dest_str>]
```

Attributes

OUTPUT= <dest_str>

Specifies where output is logged.

Values

<dest_str>

Specifies a destination name. The default destination is OPCOM, but a local or remote file can also be the destination.

SET OMNICL LOGGING

SET OMNICL LOGGING — Defines the script file to be logged and where that script file will be logged.

Format

```
SET OMNICL LOGGING OUTPUT= <dest_str>]
```

Attributes

OUTPUT= <dest_str>

Specifies where output is logged.

Values

<dest_str>

Specifies a destination name.

SHOW VERSION

SHOW VERSION — Displays the current version of VSI OMNI.

Format

```
SHOW VERSION
```

SHOW EVENT LOGGING

SHOW EVENT LOGGING — Displays either EVENT logging attributes that are provided by VSI OMNI as default values or are logging values that are set using the SET EVENT LOGGING command.

Format

```
SHOW EVENT LOGGING
```

SHOW OMNICL LOGGING

SHOW OMNICL LOGGING — Displays either OMNICL logging attributes that are provided by VSI OMNI as default values or are logging values that are set using the SET OMNICL LOGGING command.

Format

```
SHOW OMNICL LOGGING
```

SHOW APPLICATION_ENTITY

SHOW APPLICATION_ENTITY — Displays the logging attributes set by using the SET DEFAULT command. Each active association for the specified application entity is displayed along with its associated state. If the KNOWN option is invoked, all known application entities are displayed.

Format

```
SHOW [KNOWN] APPLICATION_ENTITY [<application_entity>]
```

Attributes

<dest_str>

Specifies the destination.

<event_id>

Identifies the event.

Values

<application_entity>

Specifies the name of the application entity.

SHOW ASSOCIATIONS

SHOW ASSOCIATIONS — Displays associations.

Format

```
SHOW [KNOWN] ASSOCIATIONS [<Sys ID>]
```

Attributes

None.

Values

<Sys ID>

Specifies a unique association-id.

ENABLE EVENT LOGGING

ENABLE EVENT LOGGING — Initiates event logging.

Format

```
ENABLE EVENT LOGGING
```

ENABLE OMNICL LOGGING

ENABLE OMNICL LOGGING — Initiates scripting

Format

```
ENABLE OMNICL LOGGING
```

DISABLE EVENT LOGGING

DISABLE EVENT LOGGING — Discontinues event logging.

Format

```
DISABLE EVENT LOGGING
```

DISABLE OMNICL LOGGING

DISABLE OMNICL LOGGING — Discontinues scripting.

Format

```
DISABLE OMNICL LOGGING
```

DO

DO — Invokes command files.

Format

```
DO <script_filename>
```

Attributes

None.

Values

<script_filename>

Specifies the script file. The default file extension is .SCP.

Description

Commands can be stored in text files either by using a text editor or by invoking the logging facility with ENABLE OMNICL LOGGING. These command files, or scripts, are invoked by the DO command and are useful for initialization and other commonly performed activities.

When the DO command invokes a script, OMNICL recognizes that script as an alternative source of standard commands. DO scripts are executed synchronously. Multiple levels of scripts are allowed.