VMS Software

VSI OMNI

OMNI Network Manager's Guide

Revision Update Information: This is a revised document.

Operating System and Version: VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher

Software Version: VSI OMNI API Version 4.1

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

OMNI Network Manager's Guide

Legal Notice

Table of Contents

Preface

The *VSI OMNI Network Manager's Guide* describes management functions that monitor, define, and control select data within the VSI OMNI system, which is based on the MMS (Manufacturing Message Specification) ISO 9506. MMS specifies the semantics and syntax for communications between applications running on computers and on dedicated factory floor processors such as robots and programmable logic controllers.

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.

2. Intended Audience

This document is for an audience experienced in network management.

The task of modifying VSI OMNI API data should be attempted only by managers who have knowledge of MMS and OSI (Open Systems Interconnection) concepts and hands-on network management experience. If you do not have these prerequisites, it is recommended that you run VSI OMNI API using the default attributes initially set for the software.

3. Document Structure

The *VSI OMNI Network Manager's Guide* is structured as follows:

- *Chapter 1, "Introduction to VSI OMNI Network Management"* presents introductory and startup information.
- *Chapter 2, "The OMNI Definition Facility"* describes the OMNI Definition Facility (ODF) and its functions. ODF controls system management and configuration tasks and enables a system manager to create local definitions of remote VMD objects.
- *Chapter 3, "OMNI Definition Facility (ODF) Commands"* describes ODF commands.
- *Chapter 4, "OMNI Command Language"* presents OMNICL information and OMNICL management commands. OMNICL commands monitor data within the VSI OMNI system.
- *Appendix A, "ODF Predefined Types"* lists ODF predefined types.
- *Appendix B, "ODF Error Messages"* provides a list of ODF error messages.
- *Appendix C, "Supported Mappings"* gives information about the VSI OMNI API network provider, VAX DEC/MAP V3.

4. Associated Documents

This document is part of the following online documentation set:

VSI OMNI Application Programmer's Guide

VSI OMNI API Guide to Using OmniView

VSI OMNI API for OpenVMS Installation Guide

5. OpenVMS Documentation

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

6. 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 VSI OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

7. Conventions

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

Ctrl/x indicates that you must press the key labeled Ctrl while you simultaneously press another key.

A vertical series of periods, or ellipsis, mean that some of the sample text is missing. The point of the example is made without displaying all of the sample text.

Attributes enclosed in brackets [ ] are optional.

Attributes or values enclosed in braces { } are a choice. At least one of the choices must be supplied. Braces around a single choice indicates that the enclosed syntax is mandatory.

A word enclosed by angle brackets < > is either the name or the value of an attribute that a user provides.

A comma may be used as a delimiter between optional parameters.

A horizontal series of periods, or ellipsis, indicates that an element may be repeated.

Chapter 1. Introduction to VSI OMNI Network Management

The user interfaces to VSI OMNI network management are the OMNI Control Language (OMNICL) and the OMNI Definition Facility (ODF).

1.1. VSI OMNI Network Management Functions

OMNICL consists of a set of commands that enable you to read and monitor system-wide data on the OMNI system. ODF controls system management and configuration tasks and enables you to create local definitions of remote VMD objects.

ODF commands are explained in *Chapter 3, "OMNI Definition Facility (ODF) Commands"*, and OMNICL commands are explained in *Chapter 4, "OMNI Command Language"*.

1.2. Network Provider Requirements

There are configuration requirements that VSI OMNI API and its network provider must meet before communication with a Virtual Manufacturing Device (VMD) can occur.

Also, for further information about VSI OMNI API installation and network and system considerations see the *VSI OMNI API for OpenVMS Installation Guide*.

Chapter 2. The 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 API and can be defined by using ODF.

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

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:

```
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:

```
(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:

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

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

```
 $ ODF
  ODF>
```

Once invoked, the ODF prompt appears. 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:

```
$ 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:

```
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 following items in *Table 2.1, "VMD Definitions"*.

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 *Section 2.10, "Committing Definitions to the ODF Database"*), for example:

```
ODF> DEFINE VMD myvmd
(DES,APP SIM NAM,VER,NES LEV,MAX SER CALLE/CALLI,VEN,MOD,PAR CBB,SER
 SUPP,ATT)
```

```
_ODF> APPLICATION SIMPLE NAME mycountry@myorg@myunit@myvmd,
(DES,APP SIM NAM,VER,NES LEV,MAX SER CALLE/CALLI,VEN,MOD,PAR CBB,SER
 SUPP,ATT)
_ODF> DESCRIPTION "Example VMD"
(',', ';' )
_ODF> VERSION 1,
(DES,APP SIM NAM,VER,NES LEV,MAX SER CALLE/CALLI,VEN,MOD,PAR CBB,SER
 SUPP,ATT)
_ODF> NESTING LEVEL 7,
_ODF> MAXIMUM SERVICES CALLED 5,
_ODF> MAXIMUM SERVICES CALLING 2,
_ODF> MAXIMUM SEGMENT SIZE 512,
_ODF> VENDOR "me",
_ODF> MODEL "A",
_ODF> REVISION "first",
_ODF> PARAMETER CBB ( NOVALT, NO UNNAMED VARIABLES, TPY ),
_ODF> SUPPORTED SERVICES
_ODF> ( NO INFORMATION REPORT,
_ODF> RENAME );
```

2.6. Creating a Definition of a Domain

A complete VSI OMNI definition for a domain consists of the following elements in *Table 2.2, "Domain Definitions"*.

Table 2.2. Domain Definitions

Item	Description
vmd_name:domain_name	The name of the remote domain object and its associated VMD
[NO] DELETABLE	A value indicating whether or not the domain can be deleted from the VMD
[NO] SHARABLE	A value indicating whether or not the domain can be shared by multiple program invocations
CONTENT FILE	An OpenVMS file containing the domain
CAPABILITY FILE	An OpenVMS 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 (**COMMIT** is described in *Section 2.10, "Committing Definitions to the ODF Database"*), for example:

```
 ODF> DEFINE DOMAIN myvmd:mydom
  (CAP FIL,CON FIL,[NO]DEL,DES,[NO]SHA)
  _ODF> CAPABILITY FILE my_domains:mydom.cap CONTENT FILE
my_domains:mydom.dom
  (',', ';' )
  _ODF> , DELETABLE, NOSHARABLE;
```

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, "PI Defintion"*.

Table 2.3. PI Defintion

Item	Description
vmd_name:pi_name	The name of the program invocation and its associated VMD.
[NO] DELETABLE	A value indicating whether or not the PI can be deleted from the VMD.
[NO] REUSABLE	A value indicating whether or not 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 mon- itoring event condition. MONITOR PERMANENT indicates that the PI has a monitoring event condition that exists throughout program execu- tion. 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.
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 in it. 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 (**COMMIT** is described in *Section 2.10, "Committing Definitions to the ODF Database"*), for example:

```
ODF> DEFINE PROGRAM INVOCATION myvmd:mypi DOMAIN LIST ( myvmd:mydom )
  (',', ';' )
  _ODF> , DELETABLE, REUSABLE, XA STRING "/DEBUG", NOMONITOR;
```

2.8. Creating a Definition of a Variable

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

- 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, "Named Variable Definition"*.

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 (**COMMIT** is described in *Section 2.10, "Committing Definitions to the ODF Database"*), for example:

```
DEFINE NAMED VARIABLE Foo:X
    DESCRIPTION "VMD Foo X Coordinate"
    APPLICATION TYPE %:OMNI$LONG;
```

2.8.2. Unnamed Variables

A complete VSI OMNI definition of an unnamed variable contains the items listed in *Table 2.5, "Unnamed Varaible Definition "*.

Table 2.5. Unnamed Varaible Definition

Item	Description
vmd_name:domain_ name.variable_name	The name of the remote unnamed variable and its associated VMD (and, option- ally, 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 can 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 (**COMMIT** is described in *Section 2.10, "Committing Definitions to the ODF Database"*), for example:

```
ODF> DEFINE UNNAMED VARIABLE myvmd:X APPLICATION TYPE %:OMNI$LONG
  (DES,TYP,APP TYP)
  _ODF> NUMERIC ADDRESS %X4000, DESCRIPTION "Example of an unnamed
 variable";
  ODF> DEFINE UV myvmd:AT %:OMNI$LONG SYMBOLIC ADDRESS "$n0:0";
```

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, "MMS Named Type Definitin"*.

Table 2.6. MMS Named Type Definitin

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.

Item	Description
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, "Committing Definitions to the ODF Database"*), for example:

```
DEFINE MMS NAMED TYPE Foo:Point
              DESCRIPTION "Point in threespace"
              STRUCTURE
                  x INTEGER 32;
                  y INTEGER 32;
                  z INTEGER 32;
              END;
  DEFINE MMS NAMED TYPE Foo:Position
              DESCRIPTION "Position and Orientation"
              STRUCTURE
                  pos Foo:Point;
                  r FLOAT FORMAT WIDTH 32 EXPONENT 9;
                  o FLOAT FORMAT WIDTH 32 EXPONENT 9;
                  h FLOAT FORMAT WIDTH 32 EXPONENT 9;
              END;
```

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, "Application Named Type Definition"*.

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 can also be defined on a domain.

You must specify the application type specification. To create a VSI OMNI 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, "Committing Definitions to the ODF Database"*), for example:

```
DEFINE APPLICATION NAMED TYPE Foo:Point
                        DESCRIPTION "Point in threespace"
                        APPLICATION TYPE FROM MMS NAMED TYPE Foo:Point
                        STRUCTURE
                        (x,x) INTEGER 32;
                        (y,y) INTEGER 32;
                        (z,z) INTEGER 32;
                        END;
```

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 and/or MMS Type definition.

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. Following is an example of the type definitions that can be used under these circumstances:

```
DEFINE MMS NAMED TYPE Foo:String VISIBLE STRING 100;
        DEFINE APPLICATION NAMED TYPE Foo:String
              FROM MMS NAMED TYPE Foo:String, STRING 100;
        DEFINE APPLICATION NAMED TYPE Foo:Alt_String_NT
              FROM MMS NAMED TYPE Foo:String, NULL TERMINATED STRING 100;
        DEFINE APPLICATION NAMED TYPE Foo:Alt_String_WC
              FROM MMS NAMED TYPE Foo:String, WORD COUNTED STRING 100;
        DEFINE NAMED VARIABLE Foo:String_Var APPLICATION TYPE Foo:String;
```

Another reason for alternate access is to support applications which 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. Following is an example of the definitions that can be used in circumstances where the default application type references the entire array:

```
DEFINE MMS NAMED TYPE Foo:Int_Array ARRAY [20] of INTEGER 32;
        DEFINE APPLICATION NAMED TYPE Foo:Int_Array
            FROM MMS NAMED TYPE Foo:Int_Array, ARRAY [20] of INTEGER 32;
        DEFINE APPLICATION NAMED TYPE Foo:Alt_Int_Array_0_9
            FROM MMS NAMED TYPE Foo:Int_Array, ARRAY [0..9] of INTEGER 32;
        DEFINE APPLICATION NAMED TYPE Foo:Alt_Int_Array_5_14
            FROM MMS NAMED TYPE Foo:Int_Array, ARRAY [5..14] of INTEGER 32;
```

```
        DEFINE NAMED VARIABLE Foo:Int_Array_Var APPLICATION TYPE
 Foo:Int_Array;
```

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 refer to the *VSI OMNI Application Programmer's Guide*.

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

2.10. Committing Definitions to the ODF Database

An ODF session consists of the following steps:

1. The user enters a series of **DEFINE** and/or **DELETE** commands to describe the objects in the MMS environment. ODF saves these definitions in a special area allocated for the ODF session.

2. The user enters 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, or reports on any errors. If committing the changes will produce inconsistencies in the database, referred to as a database constraint error, ODF reports an error and does not make any of the modifications.

   For example, if you have entered 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.

If, as an ODF user, you arrange your transactions so a **COMMIT** is issued after each **DEFINE** command, you can reduce the ambiguity of constraint error messages.

The constraint error message identifiers have the following general format:

DUP<def> - A definition with the name specified in a DEFINE <def> command already exists in the database. To modify a definition, DELETE it, DEFINE it, and then COMMIT it. If modification is not wanted, and the existing database entry is correct, use ROLLBACK to cancel the DEFINE request.

<def1>NO<def2>- An attempt was made to create a definition that is dependent on the existence of another definition. For example, DOMNOVMD means that a **DEFINE DOMAIN** command was issued for a domain, and the VMD specified for that domain does not exist.

Either create the required definition or roll back the request. Remember that all database names are case sensitive.

<def1>REF<def2>- A definition refers to another definition that does not exist. For example, NVREFAT means that a **DEFINE NAMED VARIABLE** command was issued with an **APPLICATION TYPE** reference to a nonexisting Application Named Type.

Either create the required definition or roll back the request. Remember that all database names are case sensitive.

Classes of definitions are abbreviated as follows:

- VMD: Virtual Manufacturing Device
- DOM: Domain
- PI: Program Invocation
- NV: Named Variable
- UV: Unnamed Variable
- VAR: Simple variable-named or unnamed (NR or UV)
- PID: Entry in a PI list of domains
- MT: MMS Named Type
- AT: Application Named Type

A completes list of ODF error messages can be found in *Appendix B, "ODF 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. In the following command sequence, for example, domain "Bar" cannot be deleted while there is an existing definition for a named variable "Baz" within the "Bar" scope:

```
DEFINE DEFINITION VMD Foo ...
  DEFINE DEFINITION DOMAIN Foo:Bar ...
  DEFINE DEFINITION NAMED VARIABLE Foo:Bar.Baz ...
  DELETE Foo:Bar;
  COMMIT;! Will fail because of Foo:Bar.Baz
  DELETE DEFINITION Foo:Bar(NAMED VARIABLE:*);! Delete all variables in
 Domain Bar of Vmd Foo
  DELETE DEFINITION DOMAIN Foo:Bar;
  COMMIT; ! Will succeed
  DELETE DEFINITION VMD Foo;
```

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:

```
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 specifed a file, ODF creates a file `OMNI$ODF.LOG` in the current default directory and logs the session there.

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

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

2.15. Displaying Definitions and Current Settings

ODF provides the commands listed in *Table 2.8, "ODF Commands"* that you can use to display the current settings of ODF session attributes or the values of definitions in the database.

Table 2.8. ODF Commands

Command	Action
SHOW SCOPE	Shows the default VMD and domain
SHOW COMPANION STANDARD	Shows the current companion standard
SHOW ODF LOGFILE	Shows the current output file
SHOW ODF LOGGING	Shows the current logging state
SHOW DEFINITION	Displays a definition or set of definitions to SYS $OUTPUT
SHOW VERSION	Displays the version of ODF and the database level

2.16. 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, for example:

```
File COMMANDS.COM looks like:
  +--------------------------------+
  | DEFINE NAMED VARIABLE Bar ... |
  | DEFINE NAMED VARIABLE Baz ... |
  | EXIT |
  +--------------------------------+
```

```
$ ODF = "$SYS$SYSTEM:OMNI$ODF.EXE"
$ ODF
ODF> DEFINE VMD Foo;
ODF> SET SCOPE Foo;
ODF> ENABLE ODF LOGGING
ODF> sho sco
Scope is Foo:
ODF> DEFINE NAMED VAR X ...
ODF> DO COMMANDS.COM;
ODF> EXIT
$ TYPE OMNI$DEF.LOG
sho sco
! Scope is Foo:
DEFINE NAMED VAR X ...
! DO COMMANDS.COM;
! Invoking Script File DISK1:[GUEST]COMMANDS.COM;1
DEFINE NAMED VARIABLE Bar ...
DEFINE NAMED VARIABLE Baz ...
! End of Script File DISK1:[GUEST]COMMANDS.COM;1
EXIT
```

2.17. 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 there.

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:*);
```

2.18. 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, "Conventions"*.

Table 3.1. Conventions

[ ]	Brackets enclose optional expressions.
{ }	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: it is 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).

Table 3.2, "Naming Format" shows the valid formats for referring to a definition. These formats are used with the **`DELETE DEFINITION`**, **`SHOW DEFINITION`**, and **`WRITE DEFINITION`** commands.

Table 3.2. Naming Format

Definition Naming	**Format and Examples**
VMD	*`vmd_name`* vmd
Domain	[*vmd_name* :] *domain_name* vmd:dom, :dom, dom [*vmd_name*](DOMAIN: *domain_name*) v(DOMAIN:d), (DOM:d)
Program Invocation	[*vmd_name*](PROGRAM INVOCATION: *pi_name*) v(PROGRAM INVOCATION:p), (PI:p)
Named Variable	[*vmd_name*][: *dom_name*](NAMED VARIABLE: *var_name*) v:d(NAMED VARIABLE:n), :d(NV:n), :(NV:n), (NV:n)
Unnamed Variable	[*vmd_name*](UNNAMED VARIABLE: *uvar_name*) v(UNNAMED VARIABLE:n), (UV:n)
MMS Named Type	[*vmd_name*][: *dom_name*](MMS NAMED TYPE: *type_name*) v:d(MMS NAMED TYPE:n), :d(MT:n), :(MT:n), (MT:n)
Application Named Type	[*vmd_name*][: *dom_name*](APPLICATION NAMED TYPE: *type_name*)

Definition Naming	Format and Examples
	v:d(APPLICATION NAMED TYPE:n), :d(AT:n), :(AT:n), (AT:n)

COMMIT

COMMIT — The **COMMIT** command commits 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. See *Appendix B, "ODF Error Messages"* for the list of constraint errors.

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 — The **DEFINE DOMAIN** command 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>];
```

Attributes and Values

<vmd_name>

The name of the VMD definition with which the domain definition is associated. A <vmd_name> is an MMS identifier.

<domain_name>

The name of the domain.

A <domain_name> is an MMS identifer.

[NO] DELETABLE

Indicates whether or not the domain can be deleted from the VMD. The DELETABLE attribute can be set by a server only. The default is DELETABLE.

[NO] SHARABLE

Indicates whether or not the domain can be shared by multiple program invocations. The default is NO SHARABLE. ODF does not prevent PI definitions from sharing a domain that is marked NO SHARABLE.

CONTENT FILE <*content_filespec*>

A file containing the domain.

The <content_filespec> is an OpenVMS file specification or logical name. The default is "".

CAPABILITY FILE <*capability_filespec*>

A file specifying the capabilities of the domain.

The <capability_filespec> is an OpenVMS file specification or logical name. The default is OMNI $DOMAINS:[<vmd_name>]<domain_name>.cap

---

Note

If the OpenVMS file specification contains a semicolon (**;**), the specification string must be enclosed in quotes.

DESCRIPTION <*text*>

Any information identifying the domain. This is not communicated. The default is "".

The <text> is a quoted character string with a maximum length of 128 characters.

DEFINE MESSAGE

DEFINE MESSAGE — The **DEFINE MESSAGE** command 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>];
```

Attributes and Values

<vmd_name>

The name of the VMD the message belongs to. If omitted, ODF uses the default VMD that you have set with the SET SCOPE command. The <vmd_name> is an MMS identifier.

<msg_name>

The name of the message object being defined. Only one message can be defined for each VMD.

The <msg_name> is an MMS identifier.

LENGTH <*msg_length*>

The maximum length of the message data in bytes.

The <msg_length> is any positive integer in the range from 1 to 4096.

DESCRIPTION <*text*>

Information identifying the variable.

The <text> is a quoted character string with a maximum length of 80 characters. The default is "".

DEFINE PROGRAM INVOCATION

DEFINE PROGRAM INVOCATION — The **DEFINE PROGRAM INVOCATION** command 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>];
```

Attributes and Values

<vmd_name>

The name of the VMD definition with which the domain definition is associated. The <vmd_name> is an MMS identifier.

<pi_name>

The name of the program invocation.

The <pi_name> is an MMS identifer.

DOMAIN LIST <dom_id>[,<dom_id>]...

A list of one or more domain references.

A <domain_id> is an MMS identifier. Separate the IDs with commas and enclose the list in parentheses. At least one domain must be specified. The order of the list is not significant.

DESCRIPTION <*text*>

Any information identifying the PI.

The <text> is a quoted character string. The default is "". The maximum length is 128 characters.

[NO] DELETABLE

Indicates whether or not the program invocation can be deleted from the VMD. The default is DELETABLE.

[NO] RESUABLE

Indicates whether or not the program invocation can be reused. The default is NO REUSABLE.

[NO] EXECUTION ARGUMENT STRING <*text*>

An execution argument for the program invocation. If supplied, the value becomes the default for both START and RESUME service requests for the program invocation. The value can be overridden on the call to either the OMNI START or RESUME function. The default is NO EXECUTION ARGUMENT STRING.

The <text> is a symbol or a quoted text string. The maximum length is 128 characters.

DEFINE NAMED VARIABLE

DEFINE NAMED VARIABLE — The DEFINE NAMED VARIABLE command 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>;
```

Attributes and Values

<vmd>

The name of the VMD the variable belongs to. If omitted, ODF uses the default VMD that you have set with the SET SCOPE command. The <vmd> is an MMS identifier.

<dom>

The name of the domain that the variable belongs to. If not specified, ODF uses the default scope that you have set with the SET SCOPE command. See the SET SCOPE command for details.

The <dom> is an MMS identifer.

<var>

The name of the named variable being defined.

The <var> is an MMS identifier.

<type>

There is one variable type: **APPLICATION TYPE <app_type_reference>** where <app_type_reference> is a reference to a predefined application type, such as %:OMNI$LONG or to a user- defined application type. (See ***DEFINE APPLICATION NAMED TYPE***.)

You must enter a type. However, the type you specify can be overridden at run time. See *Appendix A, "ODF Predefined Types"* for a list of available predefined types.

[NO] DELETABLE

Indicates whether the variable can be deleted from the VMD. The default is **DELETABLE**.

DESCRIPTION <text>

Information identifying the variable

The <text> is a quoted character string with a maximum length of 128 characters. The default is "".

DEFINE UNNAMED VARIABLE

DEFINE UNNAMED VARIABLE — The DEFINE UNNAMED VARIABLE command 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>];
```

Attributes and Values

<vmd>

The name of the VMD the variable belongs to. If omitted, ODF uses the default VMD that you have set with the SET SCOPE command. The <vmd> is an MMS identifier.

<var>

The name of the unnamed variable being defined.

The <var> is an MMS identifier.

<type>

APPLICATION TYPE <app_type_reference> where <app_type_reference> is a reference to a predefined type such as %:OMNI$LONG or to a user-defined application type. (See ***DEFINE APPLICATION NAMED TYPE***.)

You must enter a type. However, the type you specify can be overridden at run time. See *Appendix A, "ODF Predefined Types"* for a list of available predefined types.

[NO] Supply Type Spec

Indicates whether the variable's type description is to be sent to the remote VMD with requests to access this variable. The default is NO Supply Type Spec.

DESCRIPTION <*text*>

Information identifying the variable

The <text> quoted character string with a maximum length of 128 characters. The default is "".

<address>

One of the following to indicate the address of the variable:

NUMERIC ADDRESS <longword_value>

<longword_value> can be a decimal number (the default) or a hexadecimal number. A hexadecimal number has the format *% hhhhhhhh* where each *h* is a hex digit (0-9, a-f, or A-F)

SYMBOLIC ADDRESS <address_string>

<address string> should be a string enclosed in double quotation marks (").

DEFINE MMS NAMED TYPE

DEFINE MMS NAMED TYPE — The DEFINE MMS NAMED TYPE command creates an MMS named type definition. An MMS type definition de- scribes the attributes of a variable that can be communicated to an MMS peer.

Format

```
DEFINE MMS NAMED TYPE [<vmd>]:[<dom>.]<type>
                                          <mms_type_specification>
                                          [[NO]DELETABLE]
                                          [DESCRIPTION <text>];
```

Attributes and Values

<vmd>

The name of the VMD the named type definition belongs to. If omitted, ODF uses the default VMD that you have set with the **SET SCOPE** command. <vmd> is an MMS identifier.

<dom>

The name of the domain the type belongs to. If blank, the type's scope is VMD specific. If not specified, ODF uses the default domain that you have set with the **SET SCOPE** command.

To override the default domain:

```
:<mms_named_type>.
<dom> is an MMS identifer.
```

<type>

The name of the MMS type being defined.

<type> is an MMS identifier.

<mms_type_specification>

One of the following values indicating that the variable is a structure, an array, or a simple variable:

- STRUCTURE {<mms_type_component_name> <mms_ type_specification>;}...END;

  Indicates that the value is constructed from an ordered list of one or more components, each of which can have a distinct type.

  <mms_type_component_name> is an MMS identifier.

  <mms_type_specification> describes the type of this component. The type can be a structure, an array, or a simple type.

- ARRAY <mms_array_bounds> OF <mms_type_ specification>

  Indicates that the value is an ordered sequence of elements.

  <mms_array_bounds> is a positive integer enclosed in brackets ([ ])

  <mms_type_specification> describes the type of one element in the array The type can be a structure, an array, or a simple type.

- BOOLEAN

  Indicates the type is a simple boolean.

- [VARYING] BIT STRING <cell_size_bits>

  Indicates that the type is a simple bitstring. <cell_size_ bits> is the number of bits in the bit string.

- INTEGER [<cell_size_bits>]

  Indicates that the type is a simple integer. <cell_size_ bits> is the number of bits in the largest two's complement number the integer can hold. The cell size value must be 8, 16, or 32. The default is 32.

- UNSIGNED [<cell_size_bits>]

  Indicates that the type is a simple unsigned integer. <cell_size_bits> is the number of bits in the largest binary number the unsigned integer can hold. The cell size value must be 8, 16, or 32. The default is 32.

- FLOAT

  Indicates that the type is a simple floating-point number. The format width is 32 bits and the exponent is 8 bits.

- [VARYING] OCTET STRING <size_in_octets>

  Indicates that the type is a simple octet string. Each octet can hold a value from 0 to 255. <size_in_octets> is the number of octets in the string.

- [VARYING] VISIBLE STRING <size_in_octets>

  Indicates that the type is a simple visible string. <size_ in_octets> is the number of characters in the string. The string should hold a printable ASCII value.

- GENERALIZED TIME

  Indicates that the type is a generalized time value.

- BINARY TIME DATE [NOT] INCLUDED

  Indicates that the type is a binary time value. DATE INCLUDED is the default.

- BCD [<size_in_digits>]

  Indicates that the type is an unsigned binary coded decimal number. <size_in_digits> is the number of decimal digits used to represent the maximum value the variable can hold.

- OBJECT IDENTIFIER

  Indicates that the type is an object_identifier.

- <ref>

  A reference to another MMS Named Type can be used instead of an explicit type description. The VMD scope of the reference must match the VMD scope of the MMS Named Type being defined or must be one of the predefined MMS Named Types.

[NO] DELETABLE

Indicates whether the MMS Named Type can be deleted from the VMD. The default is DELETABLE.

DESCRIPTION <*text*>

Information identifying the type.

<text> is a quoted character string with a maximum length of 128 characters. The default is "".

DEFINE APPLICATION NAMED TYPE

DEFINE APPLICATION NAMED TYPE — The **DEFINE APPLICATION NAMED TYPE** command creates an Application Named Type definition.

Format

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

```
                                        [DESCRIPTION <text>];
```

Attributes and Values

<vmd>

The name of the VMD the Application Named Type belongs to. If omitted, ODF uses the default VMD that you have set with the SET SCOPE command. <vmd> is an MMS identifier.

<dom>

The name of the domain the type belongs to. If blank, the type's scope is VMD specific. If not specified, ODF uses the default domain that you have set with the **SET SCOPE** command. (To override the default domain, enter :<named_ var_name>).

<dom> is an MMS identifer.

<type>

The name of the type being defined.

<type> is an MMS identifier.

FROM MMS NAMED TYPE <ref>

The name of the MMS type to use when using this application type. The MMS type must have the same VMD scope as the application type or must be one of the predefined MMS types. The default is the same name and scope as the application type name.

<ref> is a reference to an MMS named type definition.

<app_type_specification>

A construct specifying local format information. Mapping of an application type specification to an MMS type specification is limited to certain combinations. Structures must be mapped to structures, arrays to arrays, references to references, and simple types to simple types. See *Appendix C, "Supported Mappings"* for details.

- STRUCTURE

  Indicates that the variable is a structure. A structure entry has the format:

  ```
  STRUCTURE
  {<app_component_id> <app_type_specification>;}...
  END STRUCTURE
  ```

  <app_component_id> specifies which component of the corresponding MMS structure is being referenced, and what name the application uses to refer to that component. It has the form:

  ```
  (<app_component_name>, <mms_component_name>)
  ```

- ARRAY

  Indicates that the variable is an array. The entry has the following format:

  ```
  ARRAY
  <app_array_bounds> OF <app_type_specification>
  ```

<app_array_bounds> is a positive integer, enclosed in brackets (for example, [10]) indicating the number of elements in the array or a range (for example, [3...5]) indicating which elements in the array are included in a partial access.

<app_type_specification> indicates that the component type is a structure, an array, or a simple type.

- BOOLEAN

  Indicates the type is a simple boolean with a cell size of eight bits.

- BIT STRING <cell_size_bits>

  Indicates that the type is a simple bitstring. <cell_size_ bits> is the number of bits in the bitstring. Each bit is stored in the low order bit of an eight-bit cell.

- INTEGER [<cell_size_bits>]

  Indicates that the type is a simple integer. <cell_size_ bits> is the number of bits to use to represent the integer in two's complement format. Only cell sizes of 8, 16, or 32 are valid. The default is 32.

- UNSIGNED [<cell_size_bits>]

  Indicates that the type is a simple binary integer <cell_ size_bits> is the number of bits to store the value in. Only cell sizes of 8, 16, or 32 are valid. The default is 32.

- F_FLOAT

  Indicates that the type is a simple floating-point, stored locally in VAX F_Float format.

- STRING <size_in_bytes>

  Indicates that the type is a simple scalar byte string. <size_in_bytes> is the length of the string in bytes.

- WORD COUNTED STRING <size_in_bytes>

  Indicates that the type is a word counted string. <size_ in_bytes> is the maximum number of characters in the string.

- NULL TERMINATED STRING <size_in_bytes>

  Indicates that the type is a null terminated string. <size_ in_bytes> is the maximum number of characters in the string, not including the null terminator.

- OMNITime

  Indicates that the type is a time value stored as six words. It is the time type used internally by the VSI OMNI Application Interface.

- VMS ABSOLUTE TIME

  Indicates that the type is stored as a quadword containing a VMS absolute time value.

- BOOLEAN ARRAY <app_array_bounds>

  Indicates that the type is an array of boolean values, where each value is stored in a cell of eight bits. <app_ array_bounds> specifies the number or range of values.

- <ref>

  A reference may be used instead of an explicit type description. The VMD scope of the reference must match the VMD scope of the application named type being defined or must be one of the predefined application named types.

DESCRIPTION <*text*>

Information identifying the variable.

<text> is a quoted character string with a maximum length of 128 characters.

DEFINE VMD

DEFINE VMD — The DEFINE VMD command 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>]
                      [[NO] MODEL <model>]
                      [[NO] REVISION <revision>]
                      [DESCRIPTION <text>];
```

Attributes and Values

<vmd_name>

The local name of the VMD definition. The <vmd_name> is an MMS identifer. The VMD name is used to identify the VMD in the OMNI data base; it is not used for communications.

APPLICATION SIMPLE NAME <app_simple_name>

The simple name is used to look up the application in Directory Services. The format of the name is determined by the Directory Service Provider you are using. See MAPCL documentation for further details. If you omit the simple name, VSI OMNI uses the VMD name.

The <app_simple_name> is a quoted character string.

VERSION <version_number>

Specifes the protocol used by the application.

The <version_number> is an integer value. The range is determined by the companion standard being used. For the MMS companion standard, the following values are valid: Zero (0)- The application is DIS-compliant. One (1) - The application is IS compliant. This is the default.

NESTING LEVEL <word_value>

The maximum number of levels of nesting that can occur within any data element that is transmitted or communicated over an association with the VMD. A value of zero (0) specifies unlimited nesting. The default is 10.

The <level> is an integer.

MAXIMUM SERVICES CALLING <word_value>

The proposed maximum number of transaction object instances that can be created at the calling MMS-user on the application association. The default is 5.

The <word_value> is an integer.

MAXIMUM SERVICES CALLED <word_value>

The proposed maximum number of transaction object instances that can be created at the called MMS-user on the application association. The default is 5.

The <word_value> is an integer.

MAXIMUM SEGMENT SIZE <integer_value>

The proposed maximum size of an MMS message to exchange with a VMD.

PARAMETER CBB <cbb_list>

The set of conformance building blocks supported by the VMD.

The <cbb_list> consists of one or more of the items listed in *Table 3.3, "CBB Parameters"* separated by commas and enclosed by parentheses.

Table 3.3. CBB Parameters

Parameter	ISO 9506 Designation
[NO] ARRAYS	STR1
[NO] STRUCTURES	STR2
[NO] NAMED VARIABLES	VNAM
[NO] ALTERNATE ACCESS	VALT
[NO] UNNAMED VARIABLES	VADR
[NO] SCATTERED ACCESS	VSCA
[NO] THIRD PARTY	TPY
[NO] NAMED VARIABLE LIST	VLIS
[NO] REAL	REAL
[NO] ACKNOWLEDGEMENT EVENT CONDITION	AKEC
[NO] EVALUATION INTERVAL	CEI

SUPPORTED SERVICES <supported_service_list>

The set of services supported by the calling MMS user for the association.

The <supported_service_list> consists of one or more of the services listed in *Table 3.4, "Supported Services"* separated by commas and enclosed by parentheses.

Table 3.4. Supported Services

Service
[NO] STATUS
[NO] GET NAME LIST
[NO] IDENTIFY
[NO] RENAME
[NO] READ
[NO] WRITE
[NO] GET VARIABLE ACCESS ATTRIBUTES
[NO] DEFINE NAMED VARIABLE
[NO] DEFINE SCATTERED ACCESS
[NO] GET SCATTERED ACCESS ATTRIBUTES
[NO] DELETE VARIABLE ACCESS
[NO] DEFINE NAMED VARIABLE LIST
[NO] GET NAMED VARIABLE LIST ATTRIBUTES
[NO] DELETE NAMED VARIABLE LIST
[NO] DEFINE NAMED TYPE
[NO] GET NAMED TYPE ATTRIBUTES
[NO] DELETE NAMED TYPE
[NO] INPUT
[NO] OUTPUT
[NO] TAKE CONTROL
[NO] RELINQUISH CONTROL
[NO] DEFINE SEMAPHORE
[NO] DELETE SEMAPHORE
[NO] REPORT SEMAPHORE STATUS
[NO] REPORT POOL SEMAPHORE STATUS
[NO] REPORT SEMAPHORE ENTRY STATUS
[NO] INITIATE DOWNLOAD SEQUENCE
[NO] DOWNLOAD SEGMENT
[NO] TERMINATE DOWNLOAD SEQUENCE
[NO] INITIATE UPLOAD SEQUENCE
[NO] UPLOAD SEGMENT
[NO] TERMINATE UPLOAD SEQUENCE
[NO] REQUEST DOMAIN DOWNLOAD
[NO] REQUEST DOMAIN UPLOAD
[NO] LOAD DOMAIN CONTENT
[NO] STORE DOMAIN CONTENT

Service
[NO] DELETE DOMAIN
[NO] GET DOMAIN ATTRIBUTES
[NO] CREATE PROGRAM INVOCATION
[NO] DELETE PROGRAM INVOCATION
[NO] START
[NO] STOP
[NO] RESUME
[NO] RESET
[NO] KILL
[NO] GET PROGRAM INVOCATION ATTRIBUTES
[NO] OBTAIN FILE
[NO] DEFINE EVENT CONDITION
[NO] DELETE EVENT CONDITION
[NO] GET EVENT CONDITION ATTRIBUTES
[NO] REPORT EVENT CONDITION STATUS
[NO] ALTER EVENT CONDITION MONITORING
[NO] TRIGGER EVENT [NO] DEFINE EVENT ACTION
[NO] DELETE EVENT ACTION
[NO] GET EVENT ACTION ATTRIBUTES
[NO] REPORT EVENT ACTION STATUS
[NO] DEFINE EVENT ENROLLMENT
[NO] DELETE EVENT ENROLLMENT
[NO] ALTER EVENT ENROLLMENT
[NO] REPORT EVENT ENROLLMENT STATUS
[NO] GET EVENT ENROLLMENT ATTRIBUTES
[NO] ACKNOWLEDGE EVENT NOTIFICATION
[NO] GET ALARM SUMMARY
[NO] GET ALARM ENROLLMENT SUMMARY
[NO] READ JOURNAL
[NO] WRITE JOURNAL
[NO] INITIALIZE JOURNAL
[NO] REPORT JOURNAL STATUS
[NO] CREATE JOURNAL
[NO] DELETE JOURNAL
[NO] GET CAPABILITY LIST
[NO] FILE OPEN
[NO] FILE READ
[NO] FILE CLOSE
[NO] FILE RENAME

Service
[NO] FILE DELETE
[NO] FILE DIRECTORY
[NO] UNSOLICITED STATUS
[NO] INFORMATION REPORT
[NO] EVENT NOTIFICATION
[NO] ATTACH TO EVENT CONDITION
[NO] ATTACH TO SEMAPHORE
[NO] CONCLUDE
[NO] CANCEL

VENDOR <vendor_name>

The name of the vendor of the system that supports this VMD, enclosed in double quotes (''). The default is NO VENDOR. VSI OMNI uses the default vendor name. The vendor name is relevant only when you are defining a server VMD.

The <vendor_name> is a character string of as many as 128 characters.

MODEL <model>

The model of the system supported by the VMD, enclosed in double quotes ('').

The default is NO MODEL. VSI OMNI uses the default model name. The model name is relevant only when you are defining a server VMD.

The <model> is a character string of as many as 128 characters.

REVISION <revision>

The name of the revision of the system that supports this VMD, enclosed in double quotes (''). The default is NO REVISION. VSI OMNI uses the default revision. The revision is relevant only when you are defining a server VMD.

The <revision> is a character string of as many as 128 characters.

DESCRIPTION <text>

Any information identifying the VMD.

The <text> is a quoted character string of as many as 128 characters.

DELETE DEFINITION

DELETE DEFINITION — The **DELETE DEFINITION** command 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>]...;
```

Attributes and Values

<def_ref>

The reference to a definition. For the form of a reference, see *Table 3.2, "Naming Format"*. The **DELETE DEFINITION** command supports the special characters in definition references in *Table 3.5, "DELETE DEFINITION Special Characters"*.

Table 3.5. DELETE DEFINITION Special Characters

Character	Meaning
*	Asterisk wildcard. Matches zero (0) or more characters in a name. For example, "a * z" matches "az", "abz", "abcz", and so forth.
.	Period wildcard. Matches exactly one character in a name. For example, "a.z" matches "abz", but not "az" or "abcz".
def_ref >	Right arrow. Deletes the definition and all definitions that are dependent on the definition. Can be used with a VMD or Domain reference only.

DISABLE

DISABLE — The **DISABLE** command 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 — The **DO** command 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>;
```

Attributes and Values

<script_file>

An OpenVMS file specificiation or logical name pointing to the script file. The default file extension for a script file is .COM.

ENABLE

ENABLE — The **ENABLE** command enables OMNI logging to the logfile specified in the most recent **SET ODF LOGFILE** command. If no logfile 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 — The **EXIT** command 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 — The **QUIT** command 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 — The **ROLLBACK** command 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 — The **SET ODF LOGFILE** command specifies the file to which ODF logs the session.

Format

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

Attributes and Values

<vms_file_specification>

An OpenVMS file specification for a file to receive the session log. The default specification is `OMNI$ODF.LOG`.

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 — The **SET COMPANION STANDARD** command 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>;
```

Attributes and Values

<companion_standard_name>

The following companion standard name:

```
MMS (default)
```

SET SCOPE

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

Format

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

Attributes and Values

<vmd_name>

The VMD to use as the default.

<domain_name>

The domain to use as the default. If you leave the domain name blank, the default scope is VMD-specific. To change to a different domain in the same VMD, type **SET SCOPE :<domain_name>;**, where <domain_name> is the name of the new domain.

To blank out the scope setting, type **SET SCOPE;**.

The following examples provide Named Variable definitions with scopes set.

```
SET SCOPE v;   ! default scope is now set to VMD "v";
DEF NV x ...   ! defines a variable on VMD "v";
                 equivalent to the command "DEF NV v:x..."
DEF NV a.y ... ! defines a variable on domain "a" of VMD "v";
                 equivalent to the command "DEF NV v:a.y..."
DEF NV w:z ... ! overrides the default VMD scope and defines
                 a variable on VMD "w";
                 equivalent to the command "DEF NV w:z..."
SET SCOPE v:a; ! default scope is now set to VMD "v", domain "a"
DEF NV x ...   ! defines a variable on domain "a";
                 equivalent to "DEF NV v:a.x...";
DEF NV b.x ... ! overrides the default domain scope and defines
                 a variable on domain "b";
                 equivalent to the command "DEF NV v:b.x..."
DEF NV :x ...  ! overrides the default domain scope and defines
                 a variable on VMD "v";
                 equivalent to the command "DEF NV v:x..."
```

SHOW

SHOW — The **SHOW** command displays current ODF session settings.

Format

```
SHOW <setting> ;
```

Attributes and Values

<setting>

One of the following settings for the ODF session:

- COMPANION STANDARD
- ODF LOGFILE
- ODF LOGGING
- SCOPE
- DEFINITION
- VERSION

SHOW DEFINITION

SHOW DEFINITION — The **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>]...;
```

Attributes and Values

<def_ref>

The reference to a definition. For the form of a reference, see *Table 3.2, "Naming Format"*.

The **SHOW DEFINITION** command supports special characters in definition references as shown in *Table 3.6, "Special Characters"*.

Table 3.6. Special Characters

Character	Meaning
*	Asterisk wildcard. Matches zero (0) or more characters in a name. For exam- ple, "a * z" matches "az", "abz", "abcz", and so forth.
.	Period wildcard. Matches exactly one character in a name. For example, "a.z" matches "abz", but not "az" or "abcz".
def_ref >	Right arrow. Deletes the definition and all definitions that are dependent on the definition. Can be used with a VMD or Domain reference only.

WRITE DEFINITION

WRITE DEFINITION — The **WRITE DEFINITION** command 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>];
```

Attributes and Values

<def_ref>

A reference to a definition or set of definitions. If you enter multiple references, separate them with commas. See *Table 3.2, "Naming Format"* for examples of definition references.

<filespec>

An OpenVMS file specification for a file to contain the definition. If not specified, the command is written to the log file.

The **WRITE DEFINITION** command supports special characters in definition references as shown in *Table 3.7, "Special Characters"*.

Table 3.7. Special Characters

Character	Meaning
*	Asterisk wildcard. Matches zero (0) or more characters in a name. For exam- ple, "a * z" matches "az", "abz", "abcz", and so forth.
.	Period wildcard. Matches exactly one character in a name. For example, "a.z" matches "abz", but not "az" or "abcz".
def_ref >	Right arrow. Deletes the definition and all definitions that are dependent on the definition. Can be used with a VMD or Domain reference only.

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, **OMNICL** features and commands are described.

4.1. Summary of OMNICL Commands

This section lists all the **OMNICL** commands in the order in which they appear in this chapter. Page numbers of individual commands are given in the Table of Contents.

SET Commands

SET EVENT LOGGING

SET OMNICL LOGGING

SET DEFAULT

SHOW Commands

SHOW DEFAULT

SHOW VERSION

SHOW EVENT LOGGING

SHOW OMNICL LOGGING

SHOW APPLICATION_ENTITY

SHOW ASSOCIATIONS

ENABLE Commands

ENABLE EVENT LOGGING

ENABLE OMNICL LOGGING

DISABLE Commands

DISABLE EVENT LOGGING

DISABLE OMNICL LOGGING

DO Commands

DO

4.2. 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."

4.2.1. 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:

```
OMNICL > SET Return
```

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:

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

4.2.1.1. 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.

4.2.1.2. 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.

4.3. 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.

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 wish before returning to the DCL prompt ($).

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

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

To issue a single OMNICL command to DCL:

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

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

```
$ OMNICL
OMNICL>
```

Once invoked, the OMNICL prompt appears. (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:

```
$ 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**.

4.4. 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:

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

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

4.5. SET Command Descriptions

There are three **SET** commands:

- **SET DEFAULT**
- **SET EVENT LOGGING**
- **SET OMNICL LOGGING**

SET DEFAULT

SET DEFAULT — This command 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 — This command 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 — This command 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.

4.6. SHOW Command Descriptions

There are six **SHOW** commands:

- **SHOW DEFAULT**
- **SHOW VERSION**
- **SHOW EVENT LOGGING**
- **SHOW OMNICL LOGGING**
- **SHOW APPLICATION_ENTITY**
- **SHOW ASSOCIATIONS**

SHOW DEFAULT

SHOW DEFAULT — This command displays the default application entity that is currently active.

Format

```
SHOW DEFAULT
```

Attributes

None

Values

None

SHOW VERSION

SHOW VERSION — This command displays the current version of VSI OMNI.

Format

```
SHOW VERSION
```

Attributes

None

Values

None

SHOW EVENT LOGGING

SHOW EVENT LOGGING — This command 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
```

Attributes

None

Values

None

SHOW OMNICL LOGGING

SHOW OMNICL LOGGING — This command 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
```

Attributes

None

Values

None

SHOW APPLICATION_ENTITY

SHOW APPLICATION_ENTITY — This command 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

None

Values

<application_entity>

Specifies the name of the application entity.

SHOW ASSOCIATIONS

SHOW ASSOCIATIONS — This command displays associations.

Format

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

Attributes

None

Values

<Sys ID>

Specifies a unique association-id.

4.7. ENABLE Command Descriptions

There are two **ENABLE** commands:

- **ENABLE EVENT LOGGING**
- **ENABLE OMNICL LOGGING**

ENABLE EVENT LOGGING

ENABLE EVENT LOGGING — This command initiates event logging.

Format

```
ENABLE EVENT LOGGING
```

Attributes

EVENT LOGGING

Activates event logging.

Values

None

ENABLE OMNICL LOGGING

ENABLE OMNICL LOGGING — This command initiates scripting.

Format

```
ENABLE OMNICL LOGGING
```

Attributes

OMNICL LOGGING

Initiates script production.

Values

None

4.8. DISABLE Command Descriptions

There are two **DISABLE** commands:

- **DISABLE EVENT LOGGING**
- **DISABLE OMNICL LOGGING**

DISABLE EVENT LOGGING

DISABLE EVENT LOGGING — This command discontinues event logging.

Format

```
DISABLE EVENT LOGGING
```

Attributes

EVENT LOGGING

Deactivates event logging.

Values

None

DISABLE OMNICL LOGGING

DISABLE OMNICL LOGGING — This command discontinues scripting.

Format

```
DISABLE OMNICL LOGGING
```

Attributes

OMNICL LOGGING

Deactiviates script production.

Values

None

4.9. DO Command Description

There is one **DO** command.

DO

DO — This command invokes command files. 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.

Format

```
DO <script_filename>
```

Attributes

None

Values

<script_filename>

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

Appendix A. ODF Predefined Types

This appendix contains a list of ODF predefined types.

A.1. ODF Predefined Types

ODF supports predefined types in *Table A.1, "Predefined Types"*.

Table A.1. Predefined Types

Predefined Types	Type Description
OMNI$BIT8	8-bit bitstring transmitted as bitstring
OMNI$BIT16	16-bit bitstring transmitted as bitstring
OMNI$WC_STR4	4-byte word-counted string, transmitted as varying octet string
OMNI$WC_STR8	8-byte word-counted string, transmitted as varying octet string
OMNI$WC_STR10	10-byte word-counted string, transmitted as varying octet string
OMNI$WC_STR16	16-byte word-counted string, transmitted as varying octet string
OMNI$WC_STR18	18-byte word-counted string, transmitted as varying octet string
OMNI$WC_STR32	32-byte word-counted string, transmitted as varying octet string
OMNI$WC_FIXED_STR4	4-byte word-counted string, transmitted as fixed octet string
OMNI$WC_FIXED_STR6	6-byte word-counted string, transmitted as fixed octet string
OMNI$WC_FIXED_STR8	8-byte word-counted string, transmitted as fixed octet string
OMNI$WC_FIXED_STR10	10-byte word-counted string, transmitted as fixed octet string
OMNI$WC_FIXED_STR16	16-byte word-counted string, transmitted as fixed octet string
OMNI$WC_FIXED_STR18	18-byte word-counted string, transmitted as fixed octet string
OMNI$WC_FIXED_STR32	32-byte word-counted string, transmitted as fixed octet string
OMNI$LONG	32-bit signed integer
OMNI$WORD	16-bit signed integer
OMNI$BYTE	8-bit signed integer
OMNI$ULONG	32-bit unsigned integer

Predefined Types	Type Description
OMNI$UWORD	16-bit unsigned integer
OMNI$UBYTE	8-bit unsigned integer
OMNI$BOOLEAN	8-bit boolean, transmitted as boolean
OMNI$BIT32	32-bit bitstring, transmitted as bitstring
OMNI$F_FLOAT	F_FLOATING transmitted as FLOAT
OMNI$NT_STR4	4-byte null-terminated string, transmitted as varying visible string
OMNI$NT_STR6	6-byte null-terminated string, transmitted as varying visible string
OMNI$NT_STR8	8-byte null-terminated string, transmitted as varying visible string
OMNI$NT_STR10	10-byte null-terminated string, transmitted as varying visible string
OMNI$NT_STR16	16-byte null-terminated string, transmitted as varying visible string
OMNI$NT_STR18	18-byte null-terminated string, transmitted as varying visible string
OMNI$NT_STR32	32-byte null-terminated string, transmitted as varying visible string
OMNI$NT_FIXED_STR4	4-byte null-terminated string, transmitted as a fixed visible string
OMNI$NT_FIXED_STR6	6-byte null-terminated string, transmitted as a fixed visible string
OMNI$NT_FIXED_STR8	8-byte null-terminated string, transmitted as a fixed visible string
OMNI$NT_FIXED_STR10	10-byte null-terminated string, transmitted as a fixed visible string
OMNI$NT_FIXED_STR16	16-byte null-terminated string, transmitted as a fixed visible string
OMNI$NT_FIXED_STR18	18-byte null-terminated string, transmitted as a fixed visible string
OMNI$NT_FIXED_STR32	32-byte null-terminated string, transmitted as a fixed visible string

Appendix B. ODF Error Messages

This appendix provides a list of ODF error messages.

B.1. ODF Error Messages

ODF provides the error messages listed in the table below.

Table B.1. ODF Error Messages

ODF Messages	Meaning
ATCDUPNAME	Application Type structure contains duplicate component names
ATCNOATS	App Type Comp depends on nonexistent App Type Spec
ATCREFMISS	Cannot resolve Application Type structure component reference to MMS Named Type
ATCSTRORD	Ordering of Application Type structure components does not match MMS Named Type
ATNODOM	The Domain an Application Named Type was defined on does not exist
ATNOMT	The MMS Named Type referred to in the FROM clause of an Application Named Type definition does not exist
ATCNOMTC	Cannot resolve Application Type structure component reference to MMS Named Type component
ATNOVMD	The VMD an Application Named Type was defined on does not exist
ATSNOAT	App Named Type depends on nonexistent App Named Type
ATSMAPMTS	Mapping of Application type to MMS Named Type is not supported
ATSREFINV	Type of Application Type reference does not match type of MMS Named Type reference
ATSREFMTS	Cannot resolve Application Type reference to MMS Named Type
ATCREFATS	App Type Component refers to nonexistent App Type Specification
ATCREFMTC	App Type Component refers to nonexistent MMS Type Component
DUPVRS	(Internal)
DUPVMD	Duplicate VMD Definition
DUPDOM	Duplicate Domain Definition
DUPPI	Duplicate Program Invocation Definition

ODF Messages	Meaning
DUPPID	Duplicate entry in a Program Invocation list of domains
DUPMT	Duplicate MMS Named Type
DUPMTS	Duplicate MMS Type Specification
DUPMTC	Duplicate MMS Type Component
DUPAT	Duplicate Application Named Type
DUPATS	Duplicate Application Type Specification
DUPATC	Duplicate Application Type Component
DUPNV	Duplicate Named Variable
DUPUV	Duplicate Unnamed Variable
DUPVLS	Duplicate Variable List
DUPVLE	Duplicate Variable List Entry
DUPAPP	Duplicate Application (OSAP only)
DUPUDF	(Internal)
DUPUDA	(Internal)
DUPUDC	(Internal)
DOMNOVMD	The VMD a domain was defined for does not exist
MTCREFMTS	MMS Type Component refers to nonexistent MMS Type Specification
MSGNOVMD	(Internal)
MTSNOMT	MMS Type Spec depends on nonexistent MMS Named Type
MTCNOMTS	MMS Type Comp depends on nonexistent MMS Type Spec
MTNOVMD	The VMD an MMS Named Type was defined on does not exist
MTNODOM	The Domain an MMS Named Type was defined on does not exist
NVNOVMD	The VMD a Named Variable was defined on does not exist
NVNODOM	The Domain a Named Variable was defined on does not exist
NVREFAT	A Named Variable definition refers to an Application Named Type which does not exist
ONEMSGVMD	(Internal)
PINOVMD	The VMD a program invocation was defined on does not exist
PIDNOPI	(Internal)
PIREFDOM	One or more Domains listed in a PI Domain List is not defined
UVNOVMD	The VMD an Unnamed Variable was defined on does not exist

ODF Messages	Meaning
UVNODOM	The Domain an Unnamed Variable was defined on does not exist
UVREFAT	An unnamed Variable definition refers to an Application Named Type which does not exist
VLSNOVMD	(Internal)
VLSNODOM	(Internal)
VLENOVLS	(Internal)
VLSREFAT	(Internal)
VLENOVLS	(Internal)
VLSREFVAR	(Internal)

Appendix C. Supported Mappings

This appendix provides a list of mappings that are supported between MMS and Application Types.

Table C.1. Supported Mappings

MMS Type	Application Type
BOOLEAN	BOOLEAN
BOOLEAN	INTEGER 8
INTEGER n, n <= 8	INTEGER 8
INTEGER n, n <= 16	INTEGER 16
INTEGER n, n <= 32	INTEGER 32 (default)
UNSIGNED n, n <= 8	UNSIGNED 8
UNSIGNED n, n <=16	UNSIGNED 16
UNSIGNED n, n <=32	UNSIGNED 32 (default)
FLOAT (exponent 8, format 32)	F_FLOAT
BIT STRING n	BIT STRING x, x = n
BIT STRING n	BOOLEAN ARRAY x, x = n
[VARYING] BIT STRING n	WORD COUNTED STRING x, x >=n, x <=65535
OCTET STRING n	STRING x, x = n
[VARYING] OCTET STRING n	WORD COUNTED STRING x, x >= n, x <=65535
VISIBLE STRING n	STRING x, x = n
[VARYING] VISIBLE STRING n	NULL TERMINATED STRING x, x >= n
[VARYING] VISIBLE STRING n	WORD COUNTED STRING x, x >= n, x <=65535
GENERALIZED TIME	VMS ABSOLUTE TIME
GENERALIZED TIME	OMNI TIME
BINARY TIME DATE INCLUDED	VMS ABSOLUTE TIME
BINARY TIME DATE INCLUDED	OMNI TIME
BINARY TIME DATE NOT INCLUDED	VMS ABSOLUTE TIME
BINARY TIME DATE NOT INCLUDED	OMNI TIME
BCD n, n <= 8	UNSIGNED 32
OBJECT IDENTIFIER	STRING n
OBJECT IDENTIFIER	WORD COUNTED STRING n, n <=65535
OBJECT IDENTIFIER	NULL TERMINATED STRING n
ARRAY [n] OF <MMS type x> ARRAY [s] OF	<application type y> where s <= n and x and y are a supported mapping
ARRAY [n] OF <MMS type x> ARRAY [s1..s2] OF	<application type y> where s1 <= n, s2 <= n, s1 <= s2 and x and y are a supported mapping
STRUCTURE	STRUCTURE