

VSI OpenVMS

VSI OMNI Application Programmer's Guide

Document Number: DO-DOMAPG-01A

Publication Date: May 2024

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 Version 4.1

VSI OMNI Application Programmer's Guide



VMS Software

Copyright © 2024 VMS Software, Inc. (VSI), Boston, Massachusetts, USA

Legal Notice

Confidential computer software. Valid license from VSI required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for VSI products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. VSI shall not be liable for technical or editorial errors or omissions contained herein.

HPE, HPE Integrity, HPE Alpha, and HPE Proliant are trademarks or registered trademarks of Hewlett Packard Enterprise.

Intel, Itanium and IA64 are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java, the coffee cup logo, and all Java based marks are trademarks or registered trademarks of Oracle Corporation in the United States or other countries.

Kerberos is a trademark of the Massachusetts Institute of Technology.

Microsoft, Windows, Windows-NT and Microsoft XP are U.S. registered trademarks of Microsoft Corporation. Microsoft Vista is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Motif is a registered trademark of The Open Group.

UNIX is a registered trademark of The Open Group.

Preface	vii
1. About VSI	vii
2. Intended Audience	vii
3. Document Structure	vii
4. Associated Documents	vii
5. Related ISO Documents	vii
6. OpenVMS Documentation	vii
7. VSI Encourages Your Comments	viii
8. Typographical Conventions	viii

Part I. VSIomni Introduction

Chapter 1. Introduction to MMS and VSIomni/VMS	3
1.1. The Manufacturing Message Specification	3
1.2. MMS Objects	3
1.3. MMS Services	4
1.4. MMS Service Provider	5
1.5. MMS Clients and Servers	5
1.6. MMS Companion Standards	6
1.7. VSIomni	6
1.7.1. VSIomni Definition Facility	6
1.7.2. VSIomni Programming Interfaces	7
1.8. OmniView	7
1.9. VSIomni Command Language	8

Part II. VSIomni Programmer's Guide

Chapter 2. VSIomni Application Interface	11
2.1. VSIomni Procedures	12
2.2. User Include Files	13
2.3. Synchronous and Asynchronous Completion Notification	14
2.4. I/O Status Block	14
2.5. Control Structure for Asynchronous Completion	15
2.6. Receiving Unsolicited Indications	15
2.7. Obtaining Object Attributes	15
2.8. Invoke Identifiers	17
2.9. Quotas and Privileges	17
2.10. VSIomni and MMS Companion Standards	18
Chapter 3. Programming with VSIomni	19
3.1. Initializing VSIomni	19
3.2. Loading VMD Object Definitions and Obtaining a VMD Handle	19
3.3. Creating Definitions at Run-time	20
3.3.1. Run-time Routines	20
3.3.2. Supported Definition Classes	21
3.3.3. Variable Type Descriptions	22
3.3.3.1. Alternate Access	22
3.3.3.2. Named Types and Type Specifications	23
3.3.3.3. Creating Simple and Complex Variable Definitions	23
3.3.3.4. Reading and Writing Complex Variables with Full and Partial Access	38
3.4. Obtaining Definition Handles	38
3.4.1. Obtaining a Handle by Name	39

3.4.2. Obtaining a List of Handles	39
3.5. Environment and General Management Services	41
3.5.1. Initiating an Association	41
3.5.2. Concluding an Association	42
3.5.3. Aborting an Association	42
3.5.4. Receiving an Association Request	42
3.5.5. Responding to an Association Request	43
3.5.6. Receiving a Conclude Request	43
3.5.7. Responding to a Conclude Request	44
3.6. VMD Support Services	44
3.6.1. Obtaining VMD Status	44
3.6.2. Receiving Unsolicited Status	44
3.6.3. Obtaining a List of Objects	45
3.6.4. Requesting VMD Identification	45
3.7. Domain Services	45
3.7.1. Downloading a Domain	46
3.7.2. Uploading a Domain	46
3.7.3. Deleting a Domain	46
3.7.4. Obtaining the Attributes of a Domain	47
3.7.5. Receiving Download and Upload Client Requests	47
3.8. Program Invocation Services	47
3.8.1. Creating a Program Invocation	48
3.8.2. Deleting a Program Invocation	48
3.8.3. Starting a Program Invocation	48
3.8.4. Stopping a Program Invocation	48
3.8.5. Resuming a Program Invocation	48
3.8.6. Resetting a Program Invocation	48
3.8.7. Killing a Program Invocation	49
3.8.8. Getting Program Invocation Attributes	49
3.9. Variable Access Services	49
3.9.1. Operating on Unnamed Variable Objects	50
3.9.2. Operating on Named Variable Objects	50
3.9.3. Operating on Named Variable List Objects	50
3.9.4. Receiving Client Requests for Variable Access Services	51
3.9.4.1. Receiving and Fulfilling a Read Request	51
3.9.4.2. Receiving and Fulfilling a Write Request	51
3.10. File Services	52
Chapter 4. VSIomni Procedure Calls	53
OMNI\$ABORT	56
OMNI\$ACCEPT_CONCLUDE	57
OMNI\$ACCEPT_CONNECT	58
OMNI\$CANCEL	59
OMNI\$CONCLUDE	60
OMNI\$CONNECT	62
OMNI\$CREATE	64
OMNI\$DEFINE	65
OMNI\$DELETE	66
OMNI\$DOWNLOAD	68
OMNI\$END_LIST	70
OMNI\$FDELETE	71
OMNI\$FDIR	72
OMNI\$FGET	74

OMNI\$FPUT	77
OMNI\$FRENAME	79
OMNI\$GET_ATTRIBUTE	81
OMNI\$GET_DEFINITION	83
OMNI\$GET_HANDLE_BY_NAME	88
OMNI\$GET_HANDLE_LIST	90
OMNI\$GET_INDICATIONS	92
OMNI\$GET_REMOTE_ATTRIBUTES	94
OMNI\$GET_VALUE	96
OMNI\$GROUP_VARIABLES	98
OMNI\$INITIALIZE	101
OMNI\$KILL	102
OMNI\$LISTEN	104
OMNI\$LOAD_DEFINITIONS	106
OMNI\$MODIFY_DEFINITION	107
OMNI\$PUT_VALUE	119
OMNI\$REJECT	121
OMNI\$REJECT_CONCLUDE	122
OMNI\$RESET	123
OMNI\$RESUME	125
OMNI\$START	127
OMNI\$STOP	128
OMNI\$OMNI_TO_VMS_TIME	130
OMNI\$UPLOAD	131
Chapter 5. VSIomni MMSI	133
5.1. Overview of the MMSI Implementation	133
5.2. Requirements for Using MMSI	134
5.3. Compiling and Linking User Programs with MMSI	134
5.4. Example Code	135
Appendix A. VSIomni Messages	139
Appendix B. Data Type Passing Mechanisms	151
Appendix C. MMS Attributes and VSIomni Constants	153
Appendix D. Supported Mappings	157

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.

2. Intended Audience

This document is intended for programmers with experience in OpenVMS programming for distributed systems applications and knowledge of manufacturing applications and the Manufacturing Message Specification

3. Document Structure

The *VSI OMNI Application Programmer's Guide* is structured as follows:

Chapter 1 is an overview of MMS concepts and terminology and introduction to the VSImni product set.

Chapter 2 describes the features of the VSImni Application Program Interface (API)

Chapter 3 describes the way a program uses the API to request MMS client service and to perform MMS server functions.

Chapter 4 is a reference section describing the calling format for VSImni procedures.

Chapter 5 describes VSImni MMSI.

4. Associated Documents

This document is part of the following online documentation set:

VSI OMNI Application Programmer's Guide

VSI OMNI Guide to Using OmniView

VSI OMNI API for OpenVMS Installation Guide

VSI OMNI Network Manager's Guide

5. Related ISO Documents

ISO 9506, Manufacturing message specification – Service definition

ISO 9506, Manufacturing message specification – Protocol specification

6. OpenVMS Documentation

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

7. 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.

8. Typographical Conventions

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

Convention	Meaning
UPPERCASE	Capital letters indicate MAPI function names that you have to enter as shown.
<i>Italic</i>	Words in <i>italic</i> indicate variable information that you supply.
Boldface print	Words in boldface print indicate key terms.
[]	In a procedure call, square brackets enclose parts of the command that you can omit.

Part I. VSlomni Introduction

This part provides an introduction to the concepts of the Manufacturing Message Specification (MMS) and MMS terminology and an overview of VSlomni/VMS, Digital's implementation of MMS for VMS systems.

Chapter 1. Introduction to MMS and VSlomni/VMS

This chapter contains an overview of the OSI Manufacturing Message Specification (MMS) and an introduction to VSlomni, a Digital product that allows a VMS application to take part in an MMS system.

1.1. The Manufacturing Message Specification

The Manufacturing Message Specification (MMS) is an ISO International Standard for a messaging service to support application programs that control and monitor manufacturing devices in a distributed multi-vendor processing environment.

Using the services and protocol defined by MMS, applications can exchange processing requests and responses with numerical control (NC) machines, programmable logical controllers (PLCs), and other devices on the plant floor.

1.2. MMS Objects

In the MMS programming environment, each real manufacturing device on the plant floor is modeled as an abstract object called a **virtual manufacturing device**. A VMD in turn is made up of other objects that represent the specific resources of the device. Table 1.1 lists and describes the classes of objects that can belong to a VMD.

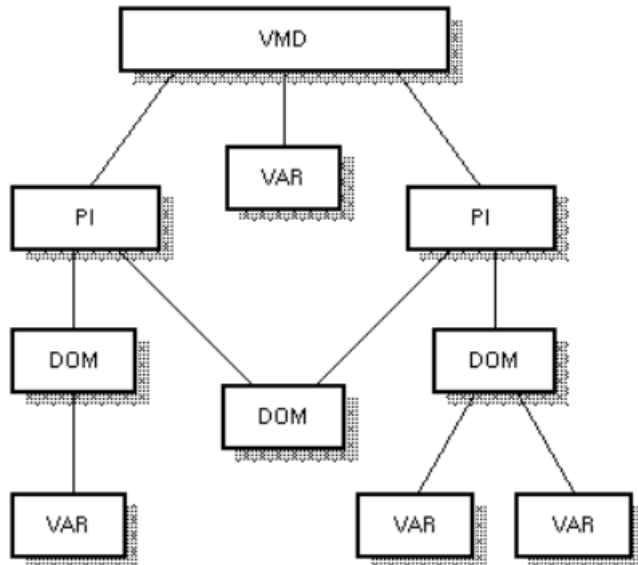
Table 1.1. MMS Objects

Object	Function
Domain	Represents a subset of resources on a physical device. These resources can take the form of executable instructions or data contained in a file, or both instructions and data.
Program invocation	Represents an executable program on the device. A PI consists of one or more domains.
Variable	Represents a variable on the device and provides the mechanism by which the application reads data from the device and writes data to the device.
Named Types	Represents a type description (such as an integer or structure) on the device. Named Types can be shared by variables and other Named Types.
Event object	Defines a condition that results in an action.
Journal object	Records the activities of the device. A journal object is typically a log file.
Semaphore object	Indicates whether a resource on the device is currently being used.

MMS objects possess attributes. In MMS terminology, an attribute is a data element with a defined meaning and a specified set of possible values. For example, each VMD object possesses a physical status attribute. According to the definition in the MMS service specification, the VMD physical status

attribute indicates the current operational state of the hardware represented by the VMD. The physical status attribute can take the following values: operational, partially-operational, inoperable, and needs-commissioning.

Objects are defined within the context of the VMD to which they belong. Variable objects can also be domain-specific (defined within the context of a domain). Domains can be defined as shareable (used in multiple program invocations) or nonshareable (used by one PI only).



VMD = Virtual Manufacturing Device
PI = Program Invocation
DOM = Domain
VAR = Variable

MR-4781-RA

1.3. MMS Services

MMS defines a set of services that an application can request to operate on VMDs and other objects. These services are described in Table 1.2

Table 1.2. MMS Services

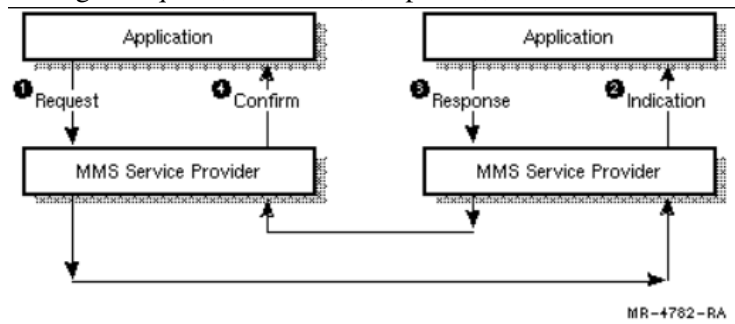
Service	Description
Environment and general management	Allows an MMS application to establish and terminate a logical connection – called an association – with a remote VMD.
VMD support	Allows an application to obtain information about the capabilities and status of a VMD object. VMD support services also allow an application to change the names of objects and receive unsolicited messages from a server about the status of a VMD.
Domain management	Allows an application to request upload and download operations on a domain, obtain the attributes of a domain, and delete a domain.
Program invocation management	Allows an application to create a program invocation object and start, stop, delete, kill, and resume operation of a program invocation.
Variable access	Allows an application to read, write, define, delete, and get the attributes of variable objects in a VMD.

Service	Description
Semaphore management	Allows multiple applications to synchronize, control, and coordinate the resources of a VMD.
Event management	Allows an application to define and manage event objects at a VMD and to obtain notifications of event occurrences.
Journal management	Allows an application to retrieve chronologically ordered information concerning events.

1.4. MMS Service Provider

Each system in the MMS environment includes an MMS service provider. Logically, the MMS service provider is located in layer 7 of the OSI reference model and depends on communications services located in the lower OSI layers.

Applications communicate through their MMS service providers by sending and receiving four types of messages: requests, indications, responses, and confirmations.



1. The application issues a service **request** to the local service provider.
2. The service provider transmits the request over the network. The remote application receives the service request from its local MMS service provider in the form of an **indication**.
3. An application that receives an indication is required to perform a specified operation and return a **response**.
4. The application that issued the original request receives the response in the form of a **confirmation**.

1.5. MMS Clients and Servers

Once the association has been established, applications exchange messages in a relationship of **server** and **client**.

An MMS server is an application that makes available the resources of one or more VMDs to other applications in the MMS environment. The server receives indications from its local MMS service provider, performs the requested operations, and returns the appropriate responses.

An MMS client is an application that issues requests to operate on a remote VMD and its objects. The client receives responses in the form of positive and negative confirmations.

An MMS application can be designed to function as both server and client – that is, it can issue client service requests and receive and fulfill service requests.

1.6. MMS Companion Standards

The MMS specification defines a common set of objects and services used by all MMS applications. MMS also allows for the definition of additional objects and services to support special classes of devices. An extension to the set of common MMS objects and services is defined as an MMS companion standard (CS).

1.7. VSIomni

VSIomni is a Digital product that allows a VMS application to take part in an MMS environment.

VSIomni provides the following user interfaces:

- OMNI Definition Facility (ODF)
- VSIomni Application Programming Interface (API)
- VSIomni MMSI
- OMNIview
- OMNI Command Language (OMNACL)

1.7.1. VSIomni Definition Facility

To an application, the MMS environment consists of objects – VMD objects and the objects belonging to VMDs. The VSIomni Definition Facility (ODF) allows a system manager to create a definition of every object in the environment. An ODF definition consists of the object's attributes and any local information required to access and operate on the object.

To create an object definition, the manager invokes ODF at the terminal and enters the ODF commands described in Table 1.3

Table 1.3. VSIomni ODF Commands

Command	Description
DEFINE VMD	Creates a definition for a VMD
DEFINE [UNNAMED] VARIABLE	Creates local definitions of simple and complex variables associated with a VMD or a domain
DEFINE DOMAIN	Creates a local definition of a remote domain and associates the definition with a VMD
DEFINE PROGRAM INVOCATION	Creates a program invocation and associates it with a VMD
DELETE	Deletes the definition of an object
ENABLE	Logs the user commands of the current ODF session to a file for later use
DISABLE	Stops the logging of the current ODF session
DO	Executes a series of stored commands, such as those in a log file
SET	Sets session defaults such as the type mapping and the scope of a definition

Command	Description
SHOW	Displays session settings
WRITE	Displays the definition of an object formatted as a DEFINE command

ODF stores the definitions in a relational database. At run time, definitions are loaded into the user process area. VSIomni assigns an identifier – called a handle – to each definition. An application specifies an object by passing VSIomni the handle of the definition.

Stored ODF object definitions relieve the programmer of the need to supply the information at run time. Under VSIomni, the application simply supplies handles. VSIomni uses the handles to obtain the information it needs to perform the requested information.

ODF is described in the online *VSI OMNI Network Manager's Guide*.

Definitions can also be created at run time through the use of the routines OMNI\$DEFINE and OMNI\$MODIFY_DEFINITION. See Chapter 4 for details on these routines.

1.7.2. VSIomni Programming Interfaces

VSIomni includes two programming interfaces to MMS services:

- The VSIomni Application Program Interface (API)
- The MMSI Interface

The VSIomni API is a high-level interface that frees the programmer from many of the details of MMS primitives.

Specifically, VSIomni procedures allow a VMS application to:

- Create definitions and retrieve their attributes at run time.
- Load ODF object definitions and obtain object handles.
- Establish and manage logical connections – called associations – with remote VMD objects.
- Receive association requests from remote applications.
- Issue MMS client requests to operate on remote objects.
- Receive client requests from remote applications to operate on local objects.

The VSIomni API is described in Chapter 2, Chapter 3 and Chapter 4

The VSIomni MMSI is an implementation of the programming interface to MMS services defined in the MMS Application Interface Specification. The VSIomni MMSI is described in Chapter 5

1.8. OmniView

OmniView is a diagnostic tool that uses DECwindows and the VSIomni Application Programmer's Interface (API) to display data associated with a Virtual Manufacturing Device (VMD) and to write the values of variables that reside on a VMD.

For OmniView to access data from a VMD, that VMD must be defined using the Omni Definition Facility (ODF). See the ODF chapter of the *VSI OMNI Network Manager's Guide*.

1.9. VSlomni Command Language

The OMNI Command Language (OMNACL) consists of a set of commands that enable the system manager to read and monitor data on the VSlomni system.

OMNACL is described in the *VSI OMNI Network Manager's Guide*.

Part II. VSIomni Programmer's Guide

This part is a guide to two VSIomni programming interfaces: the VSIomni Application Program Interface (API) and VSIomni MMSI, which implements the interface defined by the MAP MMS Interface specification.

Chapter 2. VSImni Application Interface

The VSImni Application Program Interface (API) is a set of procedures that allows a VMS application program to take part in an MMS environment. By issuing calls to VSImni procedures, the program can:

- Establish peer-to-peer associations with remote MMS applications.
- Issue client service requests to operate on VMDs and related objects located on remote MMS servers. In these operations, the VSImni application functions as an MMS client.
- Receive client requests from remote MMS applications. In these operations, the VSImni application functions as an MMS server, providing access to one or more VMDs and related objects.

VSImni procedures provide a high-level interface to MMS services. VSImni handles many MMS primitives in a way that is transparent to the application, freeing the user from many of the details of MMS services and the MMS protocol.

The VSImni API makes use of locally created and stored object definitions. An object definition consists of the object's attributes and other information needed to access and operate on the object. Object definitions are created by the VSImni manager using the VSImni Definition Facility (ODF). ODF assigns an identifier – called a handle – to each definition and stores the definition in a relational database. Definitions can also be created at run time through the use of the routines OMNI\$DEFINE and OMNI\$MODIFY_DEFINITION. See Chapter 4 for details.

ODF definitions free the programmer from the need to supply object attribute information at run time. To operate on an object, the application calls the appropriate VSImni procedure and passes the handle of the object's ODF definition as a parameter. VSImni uses the handle to locate the definition and obtain the attributes. ODF is described in the *VSI OMNI Network Manager's Guide*.

This chapter describes the following features of the VSImni application interface:

- VSImni procedures (see Section 2.1)
- User include files (see Section 2.2)
- Synchronous and asynchronous call completion notification (see Section 2.3)
- I/O status block (see Section 2.4)
- Control structure for asynchronous completion (see Section 2.5)
- Mechanism for receiving unsolicited MMS indications (see Section 2.6)
- Mechanism for obtaining attributes of remote objects (see Section 2.7)
- Invoke ID handling (see Section 2.8)
- Quotas and privileges (see Section 2.9)
- VSImni support for products that implement MMS companion standards (see Section 2.10)

Chapter 3 shows how an application procedure issues VSImni calls to establish peer-to-peer associations and to request MMS client and server functions.

Chapter 4 describes the calling format of VSImni procedures.

2.1. VSlomni Procedures

The VSlomni API provides the following categories of callable procedures:

Procedures that the application calls to request local VSlomni operations. These procedures are shown in Table 2.1

Table 2.1. Procedures for Local Operations

Procedure	Meaning
OMNI\$INITIALIZE	Initializes the VSlomni MMS service provider
OMNI\$LOAD_DEFINITIONS	Loads the ODF definitions for a specified VMD and all objects belonging to the VMD
OMNI\$GET_HANDLE_BY_NAME	Obtains the handle ID for a specified object definition
OMNI\$GET_HANDLE_LIST	Obtains the handles of the objects of a particular type belonging to a specified VMD or domain
OMNI\$DEFINE	Creates a definition at run time
OMNI\$MODIFY_DEFINITION	Modifies the value of a specified attribute of a specified definition
OMNI\$GET_DEFINITION	Retrieves a value of an attribute of a definition

Procedures that the application calls to request association management services. These procedures are shown in Table 2.2

Table 2.2. Procedures Association Management

Procedure	Meaning
OMNI\$CONNECT	Requests an association with a remote MMS application
OMNI\$LISTEN	Receives an association indication from a remote application
OMNI\$ACCEPT_CONNECT	Returns a positive response to an association indication
OMNI\$REJECT_CONNECT	Returns a negative response to an association indication
OMNI\$CONCLUDE	Requests an orderly termination of an association
OMNI\$ABORT	Requests an abrupt termination of an association
OMNI\$ACCEPT_CONCLUDE	Sends a positive response to a conclude indication
OMNI\$REJECT_CONCLUDE	Sends a negative response to a conclude indication

Procedures that the application calls to request MMS client services and perform MMS server functions. These include procedures shown in Table 2.3

Table 2.3. Procedures for MMS Services

Procedure	Meaning
OMNI\$CANCEL	Cancels a request previously issued but not yet completed
OMNI\$CREATE	Creates an object on a remote VMD
OMNI\$DELETE	Deletes an object from a remote VMD
OMNI\$GET_REMOTE_ATTRIBUTES	Obtains attribute values from a remote object and stores the values in a local buffer

Procedure	Meaning
OMNI\$GET_ATTRIBUTE	Reads an attribute from the attribute buffer
OMNI\$GROUP_VARIABLES	Reads or writes multiple variables
OMNI\$DOWNLOAD	Requests the downloading of a domain
OMNI\$UPLOAD	Requests the uploading of a domain
OMNI\$START	Start execution of a program invocation
OMNI\$STOP	Stops execution of a program invocation
OMNI\$RESET	Resets execution of a program invocation
OMNI\$KILL	Kills a program invocation
OMNI\$GET_VALUE	For client support, reads a variable on a VMD. For server support, allows the application to perform variable write service for a remote MMS client
OMNI\$PUT_VALUE	For client support, writes to a variable on a VMD. For server support, allows the application to perform variable read service for a remote MMS client

A single procedure – OMNI\$GET_INDICATIONS – that the application calls to receive unsolicited indications from a remote MMS peer. A set of procedures to request file services.

2.2. User Include Files

VSImni provides the following user include files:

- OMNI\$CODES. *lang*

All completion and error codes. Found in SYS\$LIBRARY. The *lang* extension specifies a VMS-support language.

- OMNI\$CODES.EXE

Shareable VAX message file for non-C binding. Found in SYS\$MESSAGE.

- OMNI\$DEFS. *lang*

All VSImni defined types and constants.

OMNI\$DEFS contains definitions for six categories of data type:

- Types most often used in VSImni API procedures – for example, types that are used by VSImni for diagnostics, bookkeeping, and asynchronous procedures.
- Types that correspond to a particular type or parameter in MMS. For example, to obtain the value of the address attribute of a named variable, the application specifies the OMNI\$T_ADDRESS type. OMNI\$T_ADDRESS corresponds to the address parameter defined in MMS.
- Types that are a variation on a particular type in MMS. For example, to obtain the value of the vendor attribute on a VMD, the application specifies the OMNI\$T_VMD_VENDOR type. OMNI\$T_VMD_VENDOR is a variation of the visible string type used in MMS.
- OMNI\$R_VMD_DEF type and its subtypes. OMNI\$R_VMD_DEF is the type of the "negotiated VMD" parameter in the OMNI\$CONNECT procedure.

- Types used in VSImni definitions only.
- Enumerated types (constants). A comment included with each enumeration in OMNI\$DEFS explains the use of the enumeration.

2.3. Synchronous and Asynchronous Completion Notification

VSImni supports synchronous completion for all VSImni calls and asynchronous completion for all calls that result in network activity.

In synchronous mode, the following steps occur:

1. The application issues a VSImni call and waits for completion.
2. VSImni attempts to perform the requested operation and completes. On completion, VSImni returns a parameter checking or queuing status code by function value. If the application, has provided an I/O status block (IOSB), VSImni fills in the block with completion information.
3. The application continues execution.

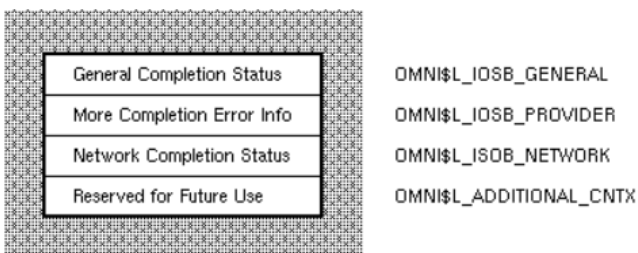
For procedures that involve network activity, the application can request asynchronous completion. In asynchronous mode, the following steps occur:

1. The application issues the call with the suffix `_A` and waits for completion.
2. The procedure returns a parameter checking queuing status code as a function value and completes.
3. If the request has been successfully issued, VSImni attempts to perform the specified operation.
4. When the attempt is completed, VSImni notifies the user by means of an asynchronous system trap (AST) or an event flag or both. (The AST and event flag number are specified in optional control structure. This structure is described in Section 2.5
5. If the application has provided an IOSB, VSImni fills in the block with completion information.

Note that the function value returned by VSImni procedures conveys parameter checking information and queuing status only. The returned value is not a completion code. To receive a completion code, the application must supply an optional I/O status block (IOSB). The IOSB is described in Section 2.4

2.4. I/O Status Block

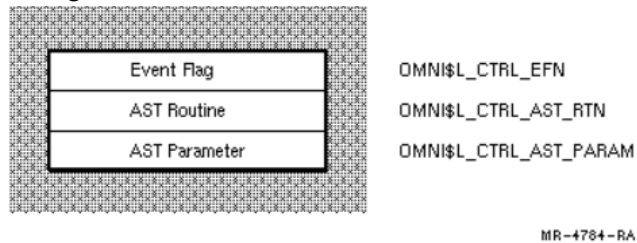
In order to receive completion status information from a VSImni procedure, the application must include the address of a 4-longword I/O Status block (IOSB) in the parameter list of the call. The figure below shows the format of the IOSB:



MR-4783-RA

2.5. Control Structure for Asynchronous Completion

To receive notification of an asynchronous completion, the application supplies a control structure that specifies an event flag or a VMS asynchronous system trap (AST) or both. This structure is described in the figure below:



2.6. Receiving Unsolicited Indications

In an MMS system, an application can receive unsolicited indications from the remote peer. Under VSImni, an application receives unsolicited indications by calling the OMNI\$GET_INDICATIONS procedure.

The call remains open until an indication arrives. When this occurs, the procedure returns a value informing the application that one of the following types of indications has been received:

- Read indication. VSImni has received a client request from the remote application to read a variable. Section 3.9.4.1 describes the way the VSImni application responds to a client read request.
- Write indication. VSImni has received a client request to write to a variable. Section 3.9.4.2 describes the way the VSImni application responds to a client write request.
- Unsolicited status. VSImni has received an unsolicited status report from a remote VMD. Section 3.6.2 describes the way the VSImni application obtains the status value.
- ACSE conclude indication. VSImni has received a conclude request from the association control service element (ACSE) and has concluded the association.
- Presentation conclude indication. VSImni has received a conclude request from presentation services. The VSImni application accepts or rejects the conclude request as described in Section 3.5.6
- Abort indication. VSImni has received an abort notice. The association is terminated.
- Information report indication. VSImni has received the current value of a variable on the remote VMD. The VSImni application obtains the value as described in Section 3.9

2.7. Obtaining Object Attributes

VSImni provides two procedures that an application program calls to obtain the attributes of objects on a remote MMS system:

- The OMNI\$GET_REMOTE_ATTRIBUTES procedure causes the VSImni MMS service provider to retrieve the current attribute values for a specified object and to place the values in a local buffer.
- The OMNI\$GET_ATTRIBUTE procedure reads a specified value from the buffer.

To obtain the current values of the attributes of an object on a remote MMS system, the application calls the GET_REMOTE_ATTRIBUTES procedure and specifies:

- The MMS object whose attribute values VSImoni is being asked to retrieve. The object can be a VMD, a domain, a program invocation, or a variable. The application specifies the object by passing the handle assigned to the object's ODF definition.
- The class of attributes whose values VSImoni is to retrieve. The application specifies the class by passing a class constant. VSImoni supports the attribute classes and constants shown in Table 2.4

Table 2.4. Attribute Classes and Constants

Constant	Result
OMNI\$K_ATTR_ALL	VSImoni retrieves all the attributes associated with the specified object.
OMNI\$K_CLS_DOM	VSImoni retrieves all the domain names for the specified VMD.
OMNI\$K_CLS_NAMED_VAR	VSImoni retrieves the attributes of a named variable.
OMNI\$K_CLS_PI	VSImoni retrieves all the program invocation names for the specified VMD.
OMNI\$K_CLS_VAR_LIST	VSImoni retrieves the attributes of a variable list.

- A location in which the procedure returns a context value to identify the buffer.

Once VSImoni has retrieved the values from the remote MMS system and placed them in the local buffer, the application calls the OMNI\$GET_ATTRIBUTE and specifies:

- The buffer to read from. The application specifies the buffer by passing the context returned by OMNI\$GET_REMOTE_ATTRIBUTES.
- An attribute to read. The application specifies the attribute by passing one of the constants shown in Table 4.5
- A location to receive the attribute. The application specifies the data type of the attribute value by passing one of the constants in Table 4.5

The constants in Table 4.5 cause VSImoni to return the first value in the attribute list of the specified type. To obtain the next value of the same type in the same list, the application calls the OMNI\$GET_ATTRIBUTE procedure and passes a NULL value in place of the constant. To read all the attributes of the same type, the application continues to issue OMNI\$GET_ATTRIBUTE calls with a NULL value until VSImoni returns the OMNI\$_ENDOFLIST code to indicate that it has reached the end of the list.

The OMNI\$GET_REMOTE_ATTRIBUTES procedure allocates memory to contain the attributes. The application is responsible for freeing this memory after it has finished reading attributes. To free memory allocated by the OMNI\$GET_REMOTE_ATTRIBUTES procedure, the application calls the OMNI\$END_LIST procedure and supplies the context value returned by OMNI\$GET_REMOTE_ATTRIBUTES.

In the following example:

1. The call to OMNI\$GET_REMOTE_ATTRIBUTES causes VSImoni to read all of the attributes for the remote VMD identified by RemoteVmdHandle and place the attributes in a local buffer.
2. The first call to OMNI\$GET_ATTRIBUTE causes VSImoni to return the vendor name attribute.

3. The second call to OMNI\$GET_ATTRIBUTE causes VSImni to read the first capability attribute in the list.
4. The OMNI\$GET_ATTRIBUTE loop causes VSImni to read the remaining capability attributes in the list.
5. The call to OMNI\$ENDLIST frees the buffer containing the attributes.

```
status = OMNI$GET_REMOTE_ATTRIBUTES(  
    NULL, /*Don't want invoke ID*/  
    RemoteVmdHandle, /*VMD identifier*/  
    OMNI$K_ATTR_ALL, /*Get all the attributes*/  
    &Context, /*Location to return context*/  
    NULL, /*Reserved for future use*/  
    &Iosb, /*I/O status block*/  
    NULL ); /*No asynchronous control structure*/  
  
/*Get vendor name*/  
Attribute=OMNI$K_ATTR_VENDOR  
status = OMNI$GET_ATTRIBUTE(  
    Context, /*Returned context value*/  
    &Attribute, /*Get vendor attribute*/  
    &VendorName /*Location to receive vendor name*/  
);  
printf( "Vendor=%s.\n", VendorName );  
/* get vmd list of capabilities*/  
Attribute=OMNI$K_ATTR_CAPABILITY  
status = OMNI$GET_ATTRIBUTE(  
    Context, /*Returned context value*/  
    &Attribute, /*Get 1st capability attribute*/  
    &Capability /*Location to receive attribute*/  
);  
while (( !error( status ) && ( status != OMNI$_ENDOFLIST)))  
{  
    printf( "Capability=%s.\n", Capability );  
    status = OMNI$GET_ATTRIBUTE(  
        Context, /*Returned context value*/  
        NULL, /*Get next capability attribute*/  
        &Capability /*Location to receive attribute*/  
    );  
}  
/*We're finished with the attribute list so free it*/  
OMNI$END_LIST(Context);
```

2.8. Invoke Identifiers

VSImni assigns a unique identifier called the invoke ID to each outbound MMS request PDU.

The optional *invoke_id* parameter on VSImni calls that request MMS services enables the application to obtain the invoke ID that VSImni has assigned to the request. The parameter specifies a location to receive the ID.

2.9. Quotas and Privileges

A value of at least 2000 for the ENQLM is required for all VSImni applications.

An application that calls the OMNI\$LISTEN procedure requires both SYSLOCK and PRMMBX privileges. If you do not have these privileges, VSImni returns a LISTEN_FAIL message in the OMNI \$L_IOSB_GENERAL field of the IOSB.

2.10. VSImni and MMS Companion Standards

MMS companion standards define extensions to the MMS specification required for specific application areas. These extensions can take the form of new objects and object attributes and new services to act on these objects. For example, the MMS specification defines all the attributes of a VMD. A companion standard for robot applications would include a definition for an application-specific VMD to model a robot. This definition would include all the attributes of the MMS VMD plus new attributes required by the robotic application area.

VSImni provides support for software products that implement MMS companion standards developed for specific manufacturing application areas.

If you are programming for a product that implements an MMS companion standard, see the product documentation for information about the ways in which your product affects the meaning of the parameters of VSImni procedure calls.

Chapter 3. Programming with VSIomni

This chapter describes how an application uses the VSIomni programming interface to perform the following operations:

- Initialize VSIomni (see Section 3.1)
- Load VMD object definitions and obtain a VMD definition handle (see Section 3.2)
- Create, modify, and retrieve definitions using the VSIomni run-time facility. (see Section 3.3)
- Obtain handles for object definitions associated with a VMD (see Section 3.4)
- Establish associations with remote applications and request other MMS environment and general management services (see Section 3.5)
- Request VMD support services (see Section 3.6)
- Request domain services and receive client requests for domain services (see Section 3.7)
- Request program invocation services (see Section 3.8)
- Request variable access services and receive client requests to read and write local variables (see Section 3.9)
- Request file management services (see Section 3.10)

3.1. Initializing VSIomni

Before an application program can begin issuing VSIomni calls, VSIomni must set certain operating parameters.

To initialize VSIomni the application calls the OMNI\$INITIALIZE procedure and supplies optional vendor, model, and revision information. VSIomni uses this information to reply to an Identify-Request from a remote MMS server.

For example, the following C code initializes VSIomni with vendor, model, and revision names.

```
status = OMNI$INITIALIZE ("Vendor", "Model", "Revision");
```

3.2. Loading VMD Object Definitions and Obtaining a VMD Handle

In a VSIomni system, every object in the MMS environment is represented by an object definition created by the system manager using ODF. Before an application can establish an association with a VMD object and operate on its related objects, the application must first load the object definitions into the user process space.

To do this, the application calls the OMNI\$LOAD_ DEFINITIONS procedure and specifies the name of the remote VMD on which it wishes to operate.

Upon completion, VSIomni delivers to the application the handle of the VMD definition. The VMD handle is a unique identifier. The application can use it to establish one association with the VMD.

For example, the following example loads the definitions created by ODF for "InitiatorVmd" and provides a location to receive the longword handle for the VMD definition.

```
status = OMNI$LOAD_DEFINITIONS ( "InitiatorVmd", &InitiatorVmdHandle);
```

When the procedure completes, all objects defined for the VMD are located in the user process space. The longword `InitiatorVmdHandle` contains the VMD handle. To specify the local definition of `InitiatorVmd` in subsequent VSImni calls, the application passes VSImni the VMD handle.

3.3. Creating Definitions at Run-time

VSImni's run-time object definition service allows the creation of definitions at run-time. It is similar to the OMNI Definition Facility (ODF) in the following ways:

- All run-time and ODF default values are identical
- Both run-time and ODF definitions must be committed before they are valid.
- For most definition classes, the values of some attributes must be specified before the definition can be committed.

Definitions created at run-time can be used in the same way as definitions that are created in ODF and loaded at run-time. The run-time object definition facility can be used in conjunction with, or in place of ODF. If it is used in place of ODF, then the `OMNI$LOAD_DEFINITIONS` call can be bypassed in the VSImni application.

3.3.1. Run-time Routines

The VSImni run-time object definition facility consists of the API routines `OMNI$DEFINE`, `OMNI$MODIFY_DEFINITION` and `OMNI$GET_DEFINITION`. The `OMNI$DEFINE` routine creates a definition in its simplest form. In other words, the call

```
Status = OMNI$DEFINE (OMNI$K_CLS_DOM, &DomHandle);
```

is equivalent to the following partial ODF command:

```
ODF> define domain
```

The first parameter to the `OMNI$DEFINE` call specifies the class of definition to create. The second parameter supplies an address in which to return the handle of the newly created definition. The handle will be used in a later portion of the discussion.

In both ODF and the run-time object definition facility, some attribute values of the Domain definition must be modified before VSImni can use the definition.

In the above example, a Name must be supplied to create a valid Domain definition. In ODF, the command can be completed successfully as follows:

```
ODF> define domain ExampleVmd:ExampleDom;
```

`ExampleVmd` does not have to be defined before `ExampleDom`. However, `ExampleVmd` must be defined in order to issue a commit or exit command. The definition is committed with the following command:

```
ODF> commit;
```

In ODF, the commit command will commit to the database all definitions created since the last commit command. It is assumed for this example that invalid definitions are not created.

In the run-time object definition facility, a Name must also be supplied for the Domain definition. The values of definition attributes are supplied by using the `OMNI$MODIFY_DEFINITION` routine. The following C program extract provides the name of the Domain Handle.

```
OMNI$L_ENUMERATION_CONST Attr = OMNI$K_ATTR_NAME; OMNI$L_CONTEXT Context =  
0;  
Status = OMNI$MODIFY_DEFINITION (DomHandle, &Attr, &Context, "Domain1");
```

The first parameter to the OMNI\$MODIFY_DEFINITION call is the handle that was returned in the previous call to OMNI\$DEFINE. The second parameter specifies the attribute whose value is to be modified. The third parameter is the address of a context. The Context is used for bookkeeping by VSIomni. The final parameter specifies the value of the specified attribute. If the specified attribute is OMNI\$K_ATTR_NAME, then VSIomni expects the attribute value to be of the data type OMNI\$T_MMS_ID. Tables that list the expected data types for possible attributes are given in the Usage Notes for OMNI\$MODIFY_DEFINITION.

The difference between the ODF command and the OMNI\$MODIFY_DEFINITION call is that a scope was not specified for the Domain. The OMNI\$MODIFY_DEFINITION routine can be used to modify only one attribute at a time. A scope must be provided for the run-time Domain definition in order for VSIomni to recognize it.

In the run-time object definition facility, modifying the value of the scope attribute is equivalent to committing that definition in ODF. The value of the scope attribute must be the handle of a previously created definition of the appropriate class. It is not necessary that the scope handle be committed before its subordinate definitions.

The following call to OMNI\$MODIFY_DEFINITION "commits" the run-time domain definition:

```
Attr = OMNI$K_ATTR_SCOPE;  
Status = OMNI$MODIFY_DEFINITION (DomHandle, &Attr, &Context, &VmdHandle);
```

It is assumed that the user has not modified the value of the context variable. The final parameter is the address of a previously created VMD. Although the VMD does not have to be committed before the domain has been committed, the user must ensure that the VMD is committed before either definition is used to request non-local operations.

Finally, the OMNI\$GET_DEFINITION routine can be used to retrieve the values of the attributes of a particular definition. The definition does not have to be committed to use OMNI\$GET_DEFINITION.

The following C program extract retrieves the value of the Name attribute of the Domain definition just created.

```
Attr = OMNI$K_ATTR_NAME;  
OMNI$T_MMS_ID Name;  
Status = OMNI$GET_DEFINITION (DomHandle, &Attr, &Context, &Name);
```

3.3.2. Supported Definition Classes

The VSIomni run-time object definition facility can be used to create definitions of the following classes:

- Vmd
- Domain
- Program Invocation
- Named Variable
- Unnamed Variable
- Message

- MMS Named Type
- Application Named Type
- MMS Type Specification
- Application Type Specification
- MMS Structure Component
- Application Structure Component

Creation of Vmd, Domain, Program Invocation, Named Variable, Unnamed Variable and Message definitions are straightforward. The remainder of the definition classes are concerned mainly with the type descriptions of variables. They are described in the following sections.

3.3.3. Variable Type Descriptions

The information VSIomni uses for type descriptions has two components: the MMS Type Definition and the Application Type Definition. The MMS Type Definition describes how data is represented on the remote device. MMS Type Definition information is communicated via the MMS protocol when the variable is read or written or its attributes are retrieved. The Application Type Definition describes how the data is stored by the application. Application Type Definition information is specific to the local programming environment and cannot be communicated via the MMS protocol.

For example, the type description of a Named Variable on the remote device may be a VisibleString. Locally, the application may wish to represent the data as a Null Terminated String. Similarly, the type description of the Named Variable on the remote device may be an Array of varying length OctetString. Locally, the application may choose to store the data as an Array of Word Counted Strings.

Every variable definition must specify a default Application Type Description Definition. In turn, each Application Type Definition refers to an MMS Type Description Definition.

A table that lists valid Application Types for MMS Types supported by VSIomni can be found in Appendix D.

3.3.3.1. Alternate Access

Simple applications will generally access a variable's data by using the default Application Type. Other applications may need to access the data by using an additional or alternate Application Type Description. Still other applications may not need to access all of the data in a particular variable. The term "Alternate Access" refers to the accessing of a variable of a simple type (simple types include all types except structures and arrays) by use of at least two different Application Types, or the partial accessing of a variable of a complex type (structure or array).

For example, an application may choose to read a Named Variable whose MMS type is a varying length visible string as both a null terminated string and a word-counted string. If it is assumed that the default Application Type Description Definition is a Null-Terminated string, then the application performs alternate access when it reads the Named Variable as a word-counted string.

Alternately, a Named Variable may be defined such that its MMS Type Description Definition is a very large array. The application may only need to access a certain portion of the array. In this case, the application may define an Application Type Definition that specifies a certain range of the MMS Type Definition. The application performs alternate access when it reads or writes the named variable by using the described Application Type Definition.

3.3.3.2. Named Types and Type Specifications

MMS allows a variable's type description to be Explicit or Named. An Explicit type description is like the C statemnt:

```
int VariableA;
```

The type description and the allocation of the variable go together. A Named type description is similar to the use of a C typedef:

```
typedef int IntegerType;  
IntegerType VariableA;
```

In this case, the type description is an entity distinct from the variable. A number of variables can be of the type IntegerType.

The run-time object definition facility supports both styles of type description.

Explicit type descriptions can be built by creating MMS and Application Type Specification definitions. The VSImni MMS Type Specification definition is an analog to the TypeSpecification parameter in MMS. MMS Type Specification definitions are referred to by other definitions. However, as in MMS, MMS Type Specification definitions may not be shared. In other words, they may only be referred to by one other MMS definition. They may be referred to by an unlimited number of Application Type Specification definitions.

Application Type Specification definitions describe the local format of the data. Every Application Type Specification definition must refer to an MMS Type Specification definition. Application Type Specification Definitions may only be referred to by one other definition.

Application and MMS Type Specification Definitions do not have to be committed. Rather, they are referenced by other definitions. VSImni enforces the limits on references to MMS and Application Type Specifications.

Named Type descriptions can be built by creating MMS and Application Named Type definitions. The VSImni MMS Named Type definition is an analog to the Named Type object in MMS. As in MMS, MMS Named Type definitions may be shared among Named Variables, Type Specifications, and other MMS Named Types.

Application Named Type definitions describe the local format of the data. Every Application Named Type definition must refer to an MMS Named Type Definition.

3.3.3.3. Creating Simple and Complex Variable Definitions

The following sections give code examples on how to build simple and complex variable definitions using the Run-time Object Definition Facility. For additional detail, see the Usage Notes for OMNI\$DEFINE, OMNI\$MODIFY_DEFINITION and OMNI\$GET_DEFINITION in Chapter 4

3.3.3.3.1. Variable_Int32: Explicit Type

```
OMNI$L_HANDLE MMSInt32, /* MMS Type Spec Handle */  
                AppInt32, /* Application Type Spec Handle */  
                Var_Int32; /* Named Variable Handle */  
  
int Status;  
OMNI$L_ENUMERATION_CONST Attribute;  
OMNI$L_ENUMERATION_CONST MMSIntType = OMNI$K_TKM_INTEGER;  
OMNI$L_ENUMERATION_CONST AppIntType = OMNI$K_TKA_INTEGER;  
int IntegerSize = 32;  
OMNI$L_CONTEXT Context = 0;
```

```

/* create an MMS Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_MMS_TYPE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute, &Context,
                                &MMSIntType);

/* The Integer has a size of 32 bits */
Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute, &Context,
                                &IntegerSize);

/* create the Application Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_APP_TYPE;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute, &Context,
                                &AppIntType);

/* The Integer has a size of 32 bits */
Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute, &Context,
                                &IntegerSize);

/* This Type Specification refers to the MMS Integer Type */
/* Specification */
Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute, &Context,
                                &MMSInt32);

/* Create a Named Variable Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_NAMED_VAR, &Var_Int32);
/* Give this Named Variable a Name */
Attribute = OMNI$K_ATTR_NAME;
Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute, &Context,
                                "Var_Int32");

/* Specify the Application Type Description */
Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute, &Context,
                                &AppInt32);

/* Modify the Scope ("Commit") */
/* Assume that the Vmd has already been created. */
Attribute = OMNI$K_ATTR_SCOPE;
Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute, &Context,
                                &VmdHandle);

```

3.3.3.3.2. Variable_Int32: Named Type

```

OMNI$L_HANDLE MMSInt32, /* MMS Type Spec Handle */
               AppInt32, /* Application Type Spec Handle
*/
               IntMNT /* MMS Named Type Handle */
               IntANT /* Application Named Type Handle */
               Var_Int32; /* Named Variable Handle */

int Status;
OMNI$L_ENUMERATION_CONST Attribute;
OMNI$L_ENUMERATION_CONST MMSIntType = OMNI$K_TKM_INTEGER;
OMNI$L_ENUMERATION_CONST AppIntType = OMNI$K_TKA_INTEGER;
int IntegerSize = 32;
OMNI$L_CONTEXT Context = 0;
/* create an MMS Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSInt32);
/* This Type Specification is an Integer */

```

```

        Attribute = OMNI$K_ATTR_MMS_TYPE;
        Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                                    &MMSIntType);

        /* The Integer has a size of 32 bits */
        Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
        Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                                    &IntegerSize);

        /* Create the MMS Named Type */
        Status = OMNI$DEFINE (OMNI$K_CLS_MMS_NAMED_TYPE, &IntMNT);
        /* Give this MMS Named Type a Name */
        Attribute = OMNI$K_ATTR_NAME;
        Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                                    "IntMNT");

        /* Specify the Type Description for the MMS Named Type */
        Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
        Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                                    &MMSInt32);

        /* Commit the MMS Named Type */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                                    &VmdHandle);

        /
        *****/
        /* create the Application Type Specification Definition */
        Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppInt32);
        /* This Type Specification is an Integer */
        Attribute = OMNI$K_ATTR_APP_TYPE;
        Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                                    &AppIntType);

        /* The Integer has a size of 32 bits */
        Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
        Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                                    &IntegerSize);

        /* This Type Specification refers to the MMS Integer Type */
        /* Specification */
        Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;
        Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                                    &MMSInt32);

        /* Create the Application Named Type */
        Status = OMNI$DEFINE (OMNI$K_CLS_APP_NAMED_TYPE, &IntANT);
        /* Give this Application Named Type a Name */
        Attribute = OMNI$K_ATTR_NAME;
        Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                                    "IntANT");

        /* Specify the Type Description for the Application Named Type
*/
        Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
        Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                                    &AppInt32);

        /* Commit the Application Named Type */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                                    &VmdHandle);

```

```

/* Create a Named Variable Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_NAMED_VAR, &Var_Int32);
/* Give this Named Variable a Name */
Attribute = OMNI$K_ATTR_NAME;
Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute,
&Context,
                                "Var_Int32");

/* Specify the Application Type Description */
Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute,
&Context,
                                &IntANT);

/* Modify the Scope ("Commit") */
/* Assume that the Vmd has already been created. */
Attribute = OMNI$K_ATTR_SCOPE;
Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute,
&Context,
                                &VmdHandle);

```

3.3.3.3. Struct_ABC: Explicit and Named Type

```

OMNI$L_HANDLE MMSInt32, /* MMS Type Spec Handle */
AppInt32, /* Application Type Spec Handle
*/

IntMNT /* MMS Named Type Handle */
IntANT /* Application Named Type Handle */
MMSStruct /* MMS Structure Type */
MMSCompA /* MMS Structure Component A */
MMSCompB /* MMS Structure Component B */
MMSCompC /* MMS Structure Component C */
AppStruct /* Application Structure Type */
AppCompA /* Application Structure Component
A */

AppCompB /* Application Structure Component
B */

AppCompC /* Application Structure Component
C */

Struct_ABC; /* Named Variable Handle */

int Status;
OMNI$L_ENUMERATION_CONST Attribute;
OMNI$L_ENUMERATION_CONST MMSIntType = OMNI$K_TKM_INTEGER;
OMNI$L_ENUMERATION_CONST MMSStructType = OMNI$K_TKM_STRUCTURE;
OMNI$L_ENUMERATION_CONST AppIntType = OMNI$K_TKA_INTEGER;
OMNI$L_ENUMERATION_CONST AppStructType = OMNI$K_TKA_STRUCTURE;
int IntegerSize = 32;
OMNI$L_CONTEXT Context = 0;
/* create an MMS Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_MMS_TYPE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                &MMSIntType);

/* The Integer has a size of 32 bits */
Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                &IntegerSize);

```

```

/* Create the MMS Named Type */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_NAMED_TYPE, &IntMNT);
/* Give this MMS Named Type a Name */
Attribute = OMNI$K_ATTR_NAME;
Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                "IntMNT");

/* Specify the Type Description for the MMS Named Type */
Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                &MMSInt32);

/* Commit the MMS Named Type */
Attribute = OMNI$K_ATTR_SCOPE;
Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                &VmdHandle);
/
*****/
/* create the Application Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_APP_TYPE;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                &AppIntType);

/* The Integer has a size of 32 bits */
Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                &IntegerSize);

/* This Type Specification refers to the MMS Integer Type */
/* Specification */
Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                &MMSInt32);

/* Create the Application Named Type */
Status = OMNI$DEFINE (OMNI$K_CLS_APP_NAMED_TYPE, &IntANT);
/* Give this Application Named Type a Name */
Attribute = OMNI$K_ATTR_NAME;
Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                "IntANT");

/* Specify the Type Description for the Application Named Type
*/
Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                &AppInt32);

/* Commit the Application Named Type */
Attribute = OMNI$K_ATTR_SCOPE;
Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                &VmdHandle);
/
*****/
/* Create the MMS Structure type */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSStruct);
/* This type specification is a structure */
Attribute = OMNI$K_ATTR_MMS_TYPE;
Status = OMNI$MODIFY_DEFINITION (MMSStruct, &Attribute,
&Context,
                                &MMStructType);

```

```

        /* Create the components on this structure */
        /* Create Component A */
        Status = OMNI$DEFINE (OMNI$K_CLS_MMS_STRUCT_COMP, MMSCompA);
        /* Give this component a name */
        Attribute = OMNI$K_ATTR_NAME;
        Status = OMNI$MODIFY_DEFINITION (MMSCompA, &Attribute,
&Context, "A");
        /* Give this component a Type Description */
        Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
        Status = OMNI$MODIFY_DEFINITION (MMSCompA, &Attribute,
&Context,
                                                &IntMNT);

        /* Commit this Component */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (MMSCompA, &Attribute,
&Context,
                                                &MMSStruct);

        /* Create Component B */
        Status = OMNI$DEFINE (OMNI$K_CLS_MMS_STRUCT_COMP, MMSCompB);
        /* Give this component a name */
        Attribute = OMNI$K_ATTR_NAME;
        Status = OMNI$MODIFY_DEFINITION (MMSCompB, &Attribute,
&Context, "B");
        /* Give this component a Type Description */
        Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
        Status = OMNI$MODIFY_DEFINITION (MMSCompB, &Attribute,
&Context,
                                                &IntMNT);

        /* Commit this Component */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (MMSCompB, &Attribute,
&Context,
                                                &MMSStruct);

        /* Create Component C */
        Status = OMNI$DEFINE (OMNI$K_CLS_MMS_STRUCT_COMP, MMSCompA);
        /* Give this component a name */
        Attribute = OMNI$K_ATTR_NAME;
        Status = OMNI$MODIFY_DEFINITION (MMSCompC, &Attribute,
&Context, "C");
        /* Give this component a Type Description */
        Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
        Status = OMNI$MODIFY_DEFINITION (MMSCompC, &Attribute,
&Context,
                                                &IntMNT);

        /* Commit this Component */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (MMSCompC, &Attribute,
&Context,
                                                &MMSStruct);

        /
        *****/
        /* Create the Application Structure type */
        Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppStruct);
        /* This type specification is a structure */
        Attribute = OMNI$K_ATTR_APP_TYPE;
        Status = OMNI$MODIFY_DEFINITION (AppStruct, &Attribute,
&Context,

```

```
&AppStructType);
    /* This type specification refers to the MMS Structure */
    Attribute = OMNISK_ATTR_MMS_TYPE_SPEC;
    Status = OMNISK_MODIFY_DEFINITION (AppStruct, &Attribute,
&Context,
                                                                    &MMSStruct);

    /* Create the components on this structure */
    /* Create Component A */
    Status = OMNISK_DEFINE (OMNISK_CLS_APP_STRUCT_COMP, AppCompA);
    /* Give this component a name */
    Attribute = OMNISK_ATTR_NAME;
    Status = OMNISK_MODIFY_DEFINITION (AppCompA, &Attribute,
&Context, "A");
    /* Give this component a Type Description */
    Attribute = OMNISK_ATTR_APP_TYPE_DESC;
    Status = OMNISK_MODIFY_DEFINITION (AppCompA, &Attribute,
&Context,
                                                                    &IntANT);

    /* This component refers to MMS Component A */
    Attribute = OMNISK_ATTR_MMS_STRUCT_COMP;
    Status = OMNISK_MODIFY_DEFINITION (AppCompA, &Attribute,
&Context,
                                                                    &MMSCompA);

    /* Commit this Component */
    Attribute = OMNISK_ATTR_SCOPE;
    Status = OMNISK_MODIFY_DEFINITION (AppCompA, &Attribute,
&Context,
                                                                    &AppStruct);

    /* Create Component B */
    Status = OMNISK_DEFINE (OMNISK_CLS_APP_STRUCT_COMP, AppCompB);
    /* Give this component a name */
    Attribute = OMNISK_ATTR_NAME;
    Status = OMNISK_MODIFY_DEFINITION (AppCompB, &Attribute,
&Context, "B");
    /* Give this component a Type Description */
    Attribute = OMNISK_ATTR_APP_TYPE_DESC;
    Status = OMNISK_MODIFY_DEFINITION (AppCompB, &Attribute,
&Context,
                                                                    &IntMNT);

    /* This component refers to MMS Component B */
    Attribute = OMNISK_ATTR_MMS_STRUCT_COMP;
    Status = OMNISK_MODIFY_DEFINITION (AppCompB, &Attribute,
&Context,
                                                                    &AppCompB);

    /* Commit this Component */
    Attribute = OMNISK_ATTR_SCOPE;
    Status = OMNISK_MODIFY_DEFINITION (AppCompB, &Attribute,
&Context,
                                                                    &AppStruct);

    /* Create Component C */
    Status = OMNISK_DEFINE (OMNISK_CLS_APP_STRUCT_COMP, AppCompC);
    /* Give this component a name */
    Attribute = OMNISK_ATTR_NAME;
    Status = OMNISK_MODIFY_DEFINITION (AppCompC, &Attribute,
&Context, "C");
    /* Give this component a Type Description */
    Attribute = OMNISK_ATTR_APP_TYPE_DESC;
```

```

        Status = OMNI$MODIFY_DEFINITION (AppCompC, &Attribute,
&Context,
                                &IntMNT);
        /* This component refers to MMS Component C */
        Attribute = OMNI$K_ATTR_MMS_STRUCT_COMP;
        Status = OMNI$MODIFY_DEFINITION (AppCompC, &Attribute,
&Context,
                                &AppCompC);
        /* Commit this Component */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (AppCompC, &Attribute,
&Context,
                                &AppStruct);
        /* Create a Named Variable Definition */
        Status = OMNI$DEFINE (OMNI$K_CLS_NAMED_VAR, &Struct_ABC);
        /* Give this Named Variable a Name */
        Attribute = OMNI$K_ATTR_NAME;
        Status = OMNI$MODIFY_DEFINITION (Struct_ABC, &Attribute,
&Context,
                                "Struct_ABC");
        /* Specify the Application Type Description */
        Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
        Status = OMNI$MODIFY_DEFINITION (Struct_ABC, &Attribute,
&Context,
                                &AppStruct);
        /* Modify the Scope ("Commit") */
        /* Assume that the Vmd has already been created. */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute,
&Context,
                                &VmdHandle);

```

3.3.3.3.4. Struct_ABC: Alternate Access

```

OMNI$L_HANDLE MMSInt32, /* MMS Type Spec Handle */
                                AppInt32, /* Application Type Spec Handle
*/
                                IntMNT /* MMS Named Type Handle */
                                IntANT /* Application Named Type Handle */
                                MMSStruct /* MMS Structure Type */
                                MMSCompA /* MMS Structure Component A */
                                MMSCompB /* MMS Structure Component B */
                                MMSCompC /* MMS Structure Component C */
                                AppStruct /* Application Structure Type */
                                AppCompA /* Application Structure Component
A */
                                AppCompC /* Application Structure Component
C */
                                Struct_ABC; /* Named Variable Handle */

int Status;
OMNI$L_ENUMERATION_CONST Attribute;
OMNI$L_ENUMERATION_CONST MMSIntType = OMNI$K_TKM_INTEGER;
OMNI$L_ENUMERATION_CONST MMSStructType = OMNI$K_TKM_STRUCTURE;
OMNI$L_ENUMERATION_CONST AppIntType = OMNI$K_TKA_INTEGER;
OMNI$L_ENUMERATION_CONST AppStructType = OMNI$K_TKA_STRUCTURE;
int IntegerSize = 32;
OMNI$L_CONTEXT Context = 0;
/* create an MMS Type Specification Definition */

```

```

Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_MMS_TYPE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                &MMSIntType);

/* The Integer has a size of 32 bits */
Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                &IntegerSize);

/* Create the MMS Named Type */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_NAMED_TYPE, &IntMNT);
/* Give this MMS Named Type a Name */
Attribute = OMNI$K_ATTR_NAME;
Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                "IntMNT");

/* Specify the Type Description for the MMS Named Type */
Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
                                &MMSInt32);

/* Commit the MMS Named Type */
Attribute = OMNI$K_ATTR_SCOPE;
Status = OMNI$MODIFY_DEFINITION (IntMNT, &Attribute, &Context,
&VmdHandle);
/
*****
/* create the Application Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_APP_TYPE;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                &AppIntType);

/* The Integer has a size of 32 bits */
Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                &IntegerSize);

/* This Type Specification refers to the MMS Integer Type */
/* Specification */
Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;
Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,
                                &MMSInt32);

/* Create the Application Named Type */
Status = OMNI$DEFINE (OMNI$K_CLS_APP_NAMED_TYPE, &IntANT);
/* Give this Application Named Type a Name */
Attribute = OMNI$K_ATTR_NAME;
Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                "IntANT");

/* Specify the Type Description for the Application Named Type
*/
Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                &AppInt32);

/* Commit the Application Named Type */
Attribute = OMNI$K_ATTR_SCOPE;

```

```

        Status = OMNI$MODIFY_DEFINITION (IntANT, &Attribute, &Context,
                                         &VmdHandle);
    /
    *****/
    /* Create the MMS Structure type */
    Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSStruct);
    /* This type specification is a structure */
    Attribute = OMNI$K_ATTR_MMS_TYPE;
    Status = OMNI$MODIFY_DEFINITION (MMSStruct, &Attribute,
    &Context,
                                         &MMStructType);

    /* Create the components on this structure */
    /* Create Component A */
    Status = OMNI$DEFINE (OMNI$K_CLS_MMS_STRUCT_COMP, MMSCompA);
    /* Give this component a name */
    Attribute = OMNI$K_ATTR_NAME;
    Status = OMNI$MODIFY_DEFINITION (MMSCompA, &Attribute,
    &Context, "A");
    /* Give this component a Type Description */
    Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
    Status = OMNI$MODIFY_DEFINITION (MMSCompA, &Attribute,
    &Context,
                                         &IntMNT);

    /* Commit this Component */
    Attribute = OMNI$K_ATTR_SCOPE;
    Status = OMNI$MODIFY_DEFINITION (MMSCompA, &Attribute,
    &Context,
                                         &MMSStruct);

    /* Create Component B */
    Status = OMNI$DEFINE (OMNI$K_CLS_MMS_STRUCT_COMP, MMSCompB);
    /* Give this component a name */
    Attribute = OMNI$K_ATTR_NAME;
    Status = OMNI$MODIFY_DEFINITION (MMSCompB, &Attribute,
    &Context, "B");
    /* Give this component a Type Description */
    Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
    Status = OMNI$MODIFY_DEFINITION (MMSCompB, &Attribute,
    &Context,
                                         &IntMNT);

    /* Commit this Component */
    Attribute = OMNI$K_ATTR_SCOPE;
    Status = OMNI$MODIFY_DEFINITION (MMSCompB, &Attribute,
    &Context,
                                         &MMSStruct);

    /* Create Component C */
    Status = OMNI$DEFINE (OMNI$K_CLS_MMS_STRUCT_COMP, MMSCompA);
    /* Give this component a name */
    Attribute = OMNI$K_ATTR_NAME;
    Status = OMNI$MODIFY_DEFINITION (MMSCompC, &Attribute,
    &Context, "C");
    /* Give this component a Type Description */
    Attribute = OMNI$K_ATTR_MMS_TYPE_DESC;
    Status = OMNI$MODIFY_DEFINITION (MMSCompC, &Attribute,
    &Context,
                                         &IntMNT);

    /* Commit this Component */
    Attribute = OMNI$K_ATTR_SCOPE;

```

```

        Status = OMNI$MODIFY_DEFINITION (MMSCompC, &Attribute,
&Context,
                                                    &MMSStruct);
    /
    *****/
    /* Create the Application Structure type */
    Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppStruct);
    /* This type specification is a structure */
    Attribute = OMNI$K_ATTR_APP_TYPE;
    Status = OMNI$MODIFY_DEFINITION (AppStruct, &Attribute,
&Context,

&AppStructType);
    /* This type specification refers to the MMS Structure */
    Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;
    Status = OMNI$MODIFY_DEFINITION (AppStruct, &Attribute,
&Context,
                                                    &MMSStruct);

    /* Create the components on this structure */
    /* Create Component A */
    Status = OMNI$DEFINE (OMNI$K_CLS_APP_STRUCT_COMP, AppCompA);
    /* Give this component a name */
    Attribute = OMNI$K_ATTR_NAME;
    Status = OMNI$MODIFY_DEFINITION (AppCompA, &Attribute,
&Context, "A");
    /* Give this component a Type Description */
    Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
    Status = OMNI$MODIFY_DEFINITION (AppCompA, &Attribute,
&Context,
                                                    &IntANT);

    /* This component refers to MMS Component A */
    Attribute = OMNI$K_ATTR_MMS_STRUCT_COMP;
    Status = OMNI$MODIFY_DEFINITION (AppCompA, &Attribute,
&Context,
                                                    &MMSCompA);

    /* Commit this Component */
    Attribute = OMNI$K_ATTR_SCOPE;
    Status = OMNI$MODIFY_DEFINITION (AppCompA, &Attribute,
&Context,
                                                    &AppStruct);

    /* Create Component C */
    Status = OMNI$DEFINE (OMNI$K_CLS_APP_STRUCT_COMP, AppCompC);
    /* Give this component a name */
    Attribute = OMNI$K_ATTR_NAME;
    Status = OMNI$MODIFY_DEFINITION (AppCompC, &Attribute,
&Context, "C");
    /* Give this component a Type Description */
    Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
    Status = OMNI$MODIFY_DEFINITION (AppCompC, &Attribute,
&Context,
                                                    &IntMNT);

    /* This component refers to MMS Component C */
    Attribute = OMNI$K_ATTR_MMS_STRUCT_COMP;
    Status = OMNI$MODIFY_DEFINITION (AppCompC, &Attribute,
&Context,
                                                    &AppCompC);

    /* Commit this Component */
    Attribute = OMNI$K_ATTR_SCOPE;

```

```

        Status = OMNI$MODIFY_DEFINITION (AppCompC, &Attribute,
&Context,
                                                &AppStruct);

        /* Create a Named Variable Definition */
        Status = OMNI$DEFINE (OMNI$K_CLS_NAMED_VAR, &Struct_ABC);
        /* Give this Named Variable a Name */
        Attribute = OMNI$K_ATTR_NAME;
        Status = OMNI$MODIFY_DEFINITION (Struct_ABC, &Attribute,
&Context,
                                                "Struct_ABC");

        /* Specify the Application Type Description */
        Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
        Status = OMNI$MODIFY_DEFINITION (Struct_ABC, &Attribute,
&Context,
                                                &AppStruct);

        /* Modify the Scope ("Commit") */
        /* Assume that the Vmd has already been created. */
        Attribute = OMNI$K_ATTR_SCOPE;
        Status = OMNI$MODIFY_DEFINITION (Var_Int32, &Attribute,
&Context,
                                                &VmdHandle);

```

3.3.3.3.5. Array_Int: Explicit

```

OMNI$L_HANDLE MMSInt32, /* MMS Int Type Spec Handle */
                                AppInt32, /* Application Int Type Spec
Handle */
                                MMSArray /* MMS Array Type Spec Handle */
                                AppArray /* Application Array Type Spec
Handle */
                                Array_Int; /* Named Variable Handle */

int Status;
OMNI$L_ENUMERATION_CONST Attribute;
OMNI$L_ENUMERATION_CONST MMSIntType = OMNI$K_TKM_INTEGER;
OMNI$L_ENUMERATION_CONST AppIntType = OMNI$K_TKA_INTEGER;
OMNI$L_ENUMERATION_CONST MMSArrayType = OMNI$K_TKM_ARRAY;
OMNI$L_ENUMERATION_CONST AppArrayType = OMNI$K_TKA_ARRAY;
int IntegerSize = 32;
int ArrayElements = 10;
OMNI$L_CONTEXT Context = 0;
/* create an MMS Integer Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_MMS_TYPE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                                &MMSIntType);

        /* The Integer has a size of 32 bits */
        Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
        Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
&IntegerSize);

        /* Create the MMS Array */
        Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSArray);
        /* The newly created Type Specification is an Array */
        Attribute = OMNI$K_ATTR_MMS_TYPE;

```

```

        Status = OMNI$MODIFY_DEFINITION (MMSArray, &Attribute,
&Context,

&MMSArrayType);
        /* This array has ten elements */
        Attribute = OMNI$K_ATTR_ARRAY_ELEM_NUM;
        Status = OMNI$MODIFY_DEFINITION (MMSArray, &Attribute,
&Context,

&ArrayElements);
        /* The Array elements are of type Integer */
        Attribute = OMNI$K_ATTR_ARRAY_ELEM_MMS_TYPE;
        Status = OMNI$MODIFY_DEFINITION (MMSArray, &Attribute,
&Context,

&MMSInt32);
        /
*****
        /* Create the Application Type */
        /* create the Application Type Specification Definition */
        Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppInt32);
        /* This Type Specification is an Integer */
        Attribute = OMNI$K_ATTR_APP_TYPE;
        Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,

&AppIntType);

        /* The Integer has a size of 32 bits */
        Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
        Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,

&IntegerSize);
        /* This Type Specification refers to the MMS Integer Type */
        /* Specification */
        Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;
        Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,

&MMSInt32);

        /* Create the Application Array */
        Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppArray);
        /* The newly created Type Specification is an Array */
        Attribute = OMNI$K_ATTR_APP_TYPE;
        Status = OMNI$MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&AppArrayType);
        /* This array has ten elements */
        Attribute = OMNI$K_ATTR_ARRAY_ELEM_NUM;
        Status = OMNI$MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&ArrayElements);
        /* The Array elements are of type Integer */
        Attribute = OMNI$K_ATTR_ARRAY_ELEM_APP_TYPE;
        Status = OMNI$MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&AppInt32);

        /* The Application Array refers to the MMS Array */
        Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;

```

```

        Status = OMNI$MODIFY_DEFINITION (AppArray, &Attribute,
&Context,
                                                &MMSArray);
    /
    *****/
    /* Create a Named Variable Definition */
    Status = OMNI$DEFINE (OMNI$K_CLS_NAMED_VAR, &Array_Int);
    /* Give this Named Variable a Name */
    Attribute = OMNI$K_ATTR_NAME;
    Status = OMNI$MODIFY_DEFINITION (Array_Int, &Attribute,
&Context,
                                                "Array_Int");

    /* Specify the Application Type Description */
    Attribute = OMNI$K_ATTR_APP_TYPE_DESC;
    Status = OMNI$MODIFY_DEFINITION (Array_Int, &Attribute,
&Context,
                                                &AppArray);

    /* Modify the Scope ("Commit") */
    /* Assume that the Vmd has already been created. */
    Attribute = OMNI$K_ATTR_SCOPE;
    Status = OMNI$MODIFY_DEFINITION (AppArray, &Attribute,
&Context,
                                                &VmdHandle);

```

3.3.3.3.6. Array_Int: Alternate Access

```

OMNI$L_HANDLE MMSInt32, /* MMS Int Type Spec Handle */
AppInt32, /* Application Int Type Spec
Handle */
MMSArray /* MMS Array Type Spec Handle */
AppArray /* Application Array Type Spec
Handle */
Array_Int; /* Named Variable Handle */

int Status;
OMNI$L_ENUMERATION_CONST Attribute;
OMNI$L_ENUMERATION_CONST MMSIntType = OMNI$K_TKM_INTEGER;
OMNI$L_ENUMERATION_CONST AppIntType = OMNI$K_TKA_INTEGER;
OMNI$L_ENUMERATION_CONST MMSArrayType = OMNI$K_TKM_ARRAY;
OMNI$L_ENUMERATION_CONST AppArrayType = OMNI$K_TKA_ARRAY;
int IntegerSize = 32;
int MMSArrayElements = 10; /* Number of elements in array on
device */
int AppArrayElements = 4; /* Number of elemntes in application
array*/
int AppArrayLow = 3; /* Beginning of subrange of MMS array */
OMNI$L_CONTEXT Context = 0;
/* create an MMS Integer Type Specification Definition */
Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSInt32);
/* This Type Specification is an Integer */
Attribute = OMNI$K_ATTR_MMS_TYPE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,
                                                &MMSIntType);

/* The Integer has a size of 32 bits */
Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
Status = OMNI$MODIFY_DEFINITION (MMSInt32, &Attribute,
&Context,

```

```
&IntegerSize);
    /* Create the MMS Array */
    Status = OMNI$DEFINE (OMNI$K_CLS_MMS_TYPE_SPEC, &MMSArray);
    /* The newly created Type Specification is an Array */
    Attribute = OMNI$K_ATTR_MMS_TYPE;
    Status = OMNI$MODIFY_DEFINITION (MMSArray, &Attribute,
&Context,

&MMSArrayType);
    /* This array has ten elements */
    Attribute = OMNI$K_ATTR_ARRAY_ELEM_NUM;
    Status = OMNI$MODIFY_DEFINITION (MMSArray, &Attribute,
&Context,

&MMSArrayElements);
    /* The Array elements are of type Integer */
    Attribute = OMNI$K_ATTR_ARRAY_ELEM_MMS_TYPE;
    Status = OMNI$MODIFY_DEFINITION (MMSArray, &Attribute,
&Context,

&MMSInt32);
    /
*****/
    /* Create the Application Type */
    /* create the Application Type Specification Definition */
    Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppInt32);
    /* This Type Specification is an Integer */
    Attribute = OMNI$K_ATTR_APP_TYPE;
    Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,

&AppIntType);

    /* The Integer has a size of 32 bits */
    Attribute = OMNI$K_ATTR_SIMPLE_SIZE;
    Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,

&IntegerSize);
    /* This Type Specification refers to the MMS Integer Type */
    /* Specification */
    Attribute = OMNI$K_ATTR_MMS_TYPE_SPEC;
    Status = OMNI$MODIFY_DEFINITION (AppInt32, &Attribute,
&Context,

&MMSInt32);

    /* Create the Application Array */
    Status = OMNI$DEFINE (OMNI$K_CLS_APP_TYPE_SPEC, &AppArray);
    /* The newly created Type Specification is an Array */
    Attribute = OMNI$K_ATTR_APP_TYPE;
    Status = OMNI$MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&AppArrayType);
    /* Access the MMS Elements beginning with index 3 */
    Attribute = OMNI$K_ATTR_ARRAY_LOW;
    Status = OMNI$MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&AppArrayLow);
    /* This array has four elements */
```

```

        Attribute = OMNISK_ATTR_ARRAY_ELEM_NUM;
        Status = OMNISK_MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&AppArrayElements);
        /* The Array elements are of type Integer */
        Attribute = OMNISK_ATTR_ARRAY_ELEM_APP_TYPE;
        Status = OMNISK_MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&AppInt32);

        /* The Application Array refers to the MMS Array */
        Attribute = OMNISK_ATTR_MMS_TYPE_SPEC;
        Status = OMNISK_MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&MMSArray);

/
*****/
        /* Create a Named Variable Definition */
        Status = OMNISK_DEFINE (OMNISK_CLS_NAMED_VAR, &Array_Int);
        /* Give this Named Variable a Name */
        Attribute = OMNISK_ATTR_NAME;
        Status = OMNISK_MODIFY_DEFINITION (Array_Int, &Attribute,
&Context,

"Array_Int");

        /* Specify the Application Type Description */
        Attribute = OMNISK_ATTR_APP_TYPE_DESC;
        Status = OMNISK_MODIFY_DEFINITION (Array_Int, &Attribute,
&Context,

&AppArray);

        /* Modify the Scope ("Commit") */
        /* Assume that the Vmd has already been created. */
        Attribute = OMNISK_ATTR_SCOPE;
        Status = OMNISK_MODIFY_DEFINITION (AppArray, &Attribute,
&Context,

&VmdHandle);

```

3.3.3.4. Reading and Writing Complex Variables with Full and Partial Access

The API routines OMNISK_GET_VALUE and OMNISK_PUT_VALUE are used to read and write variables. These routines use the application type of the variable to determine how the variable is stored locally. The application type of the variable may specify an alternate access type. The Method Handle parameter can be used to read or write a variable with Alternate Access. The MethodHandle parameter is the address of a committed Application Named Type definition. If a Method Handle is supplied, then VSImni treats the Method Handle as the default application type.

3.4. Obtaining Definition Handles

Once VSImni has loaded the definitions for a VMD and received the VMD handle, the application can obtain the handles of all definitions of objects belonging to the VMD. The application can obtain the handle of a specific definition or a list of handles for a group of definitions.

3.4.1. Obtaining a Handle by Name

To obtain a specific handle, the application calls the OMNI\$GET_HANDLE_BY_NAME procedure, specifies the scope of the find operation, the class of object, and name of the definition, and provides a location to receive the handle. (The scope is typically a domain or VMD. VSImni looks for the definition of the object within the specified domain or VMD.)

For example, the following C code issues a request for the handle of a domain. ``DomainName" identifies the definition. RemoteVmdHandle indicates the VMD to which the domain belongs. The call specifies the class of the object and provides a location to receive the handle for the object definition.

```
status = OMNI$GET_HANDLE_BY_NAME ( RemoteVmdHandle, OMNI$K_CLS_DOM,
                                     "DomainName",
                                     &DomainHandle );
```

Upon completion, the provided location contains the handle of the definition.

3.4.2. Obtaining a List of Handles

To obtain the list of handles for a list of corresponding definitions, the application calls the OMNI\$GET_HANDLE_LIST procedure.

Each call to OMNI\$GET_HANDLE_LIST returns a definition handle of the specified class for the specified scope. When used as part of a loop, OMNI\$GET_HANDLE_LIST can be used to retrieve all of the handles of a specified class for the specified scope.

The following C code example issues a request for the first Named Variable handle on Example_VMD:

```
OMNI$L_CONTEXT Context = 0;
OMNI$L_HANDLE VariableHandle;
OMNI$L_ENUMERATION_CONST Class, *ClassPtr;
    Class = OMNI$K_CLS_NAMED_VAR;
    ClassPtr = &Class;
status = OMNI$GET_HANDLE_LIST (Example_VMD_Handle,
                               ClassPtr,
                               &Context,
                               &VariableHandle);
```

The first argument is of type OMNI\$L_HANDLE, and specifies the scope of the definitions whose handles are to be retrieved. To specify that Example_VMD is the scope, use its handle as the value of the first argument. The OMNI\$L_HANDLE type is defined in OMNI\$DEFS.

The second argument is the address of a variable of type OMNI\$L_ENUMERATION_CONST. The value of the variable should be set to the class of handle to retrieve. The constants that are used to specify the class are also found in OMNI\$DEFS.

The third argument is the address of a variable of type OMNI\$L_CONTEXT. This type is defined in OMNI\$DEFS. On the first call to OMNI\$GET_HANDLE_LIST, the value of the Context should be zero. The value of the Context should not be modified on succeeding calls to OMNI\$GET_HANDLE_LIST.

The final argument is a pointer to a variable of type OMNI\$L_HANDLE. This parameter provides a location in which to receive the definition handle.

The following fragment of C code issues a request for the next Named Variable handle on Example_VMD:

```
status = OMNI$GET_HANDLE_LIST (Example_VMD_Handle, NULL,  
                               &Context, &VariableHandle);
```

Notice that the second parameter has a value of NULL. A NULL value for the class parameter indicates that VSImoni should retrieve the next definition handle of the same type. This code fragment can be called as part of a loop to retrieve the rest of the Named Variable handles on Example_VMD. When the last handle has been retrieved, OMNI\$_ENDOFLIST is returned as the status value. Subsequent calls to OMNI\$GET_HANDLE_LIST will return null handles.

If the value of the second argument is not null, then the first handle in the list will be returned, and will reset the internal bookkeeping so that the second handle will be returned on the next call to OMNI\$GET_HANDLE_LIST.

After the final call to OMNI\$GET_HANDLE_LIST, OMNI\$END_LIST should be called to free storage allocated by OMNI for bookkeeping purposes.

The following fragment of C Code retrieves all of the Named Variable Handles on Example_VMD:

```
OMNI$L_ENUMERATION_CONST CLASS = OMNI$K_CLS_NAMED_VAR;  
OMNI$L_ENUMERATION_CONST *ClassPtr = &Class;  
OMNI$L_CONTEXT Context = 0;  
status = OMNI$GET_HANDLE_LIST (Example_VMD_Handle,  
                               ClassPtr,  
                               &Context,  
                               &VariableHandle);  
  
if (status == OMNI$_ENDOFLIST)  
{  
    printf ("No More Handles\n");  
    exit (0);  
}  
else if (status != OMNI$SUCCESS)  
{  
    printf ("Error obtaining handle\n");  
    exit (0);  
}  
else  
{  
    ClassPtr = 0;  
    while ( (status = OMNI$GET_HANDLE_LIST (Example_VMD_Handle,  
ClassPtr,  
&Context,  
&VariableHandle)  
            != OMNI$_ENDOFLIST)  
    {  
        /* processing*/  
        :  
        :  
    }  
    OMNI$END_LIST (Context);  
}
```

3.5. Environment and General Management Services

Environment and general management services allow an MMS client application to perform the following operations:

- Initiate an association with a remote VMD.
- Conclude an association.
- Abort an association.
- Receive a request for an association with a local VMD.
- Accept, negotiate or reject an association request.
- Receive a request to conclude an association.
- Accept or reject a request to conclude an association.

3.5.1. Initiating an Association

Before an application can exchange MMS messages with a remote peer, the applications must first establish a logical connection, or association.

To initiate an association with a remote VMD, the application calls the OMNI\$CONNECT procedure and supplies a VMD handle to specify the remote VMD.

Optionally, the application can supply:

- The handle of a local VMD – the calling VMD. The application can receive client requests to operate on this VMD.
- A location to receive negotiated service parameters from the remote VMD.
- A flag that tells VSlomni to deliver any conclude request that arrives from the remote VMD during the association or to automatically accept a conclude request without delivering it to the application.
- The address of an I/O status block.
- A control structure for completion.

For example, the following C code initiates an association. `RecipientVmdHandle` identifies the target of the association request. `InitiatorVmdHandle` specifies a VMD that the initiating application will make accessible during the association. The application can receive client requests to operate on this VMD. The application is not proposing parameters for negotiation (this is currently not supported) but is providing a location to receive proposed parameters from the remote peer.

```
status = OMNI$CONNECT(  
    RecipientVmdHandle, /*Remote Vmd*/  
    NULL, /*Reserved*/  
    InitiatorVMDHandle /*Local Vmd*/  
    NegotiatedVMDStruct /*Received parameters*/
```

```
FALSE, /*Conclude flag*/  
&Iosb, /*I/O status block*/  
NULL); /*Control Structure*/
```

3.5.2. Concluding an Association

A conclude request indicates that the application has completed all the requests it had planned and will issue no further requests. VSIomni performs an orderly relinquishing of the MMS environment in cooperation with the remote MMS service.

To issue a request to conclude an association, the application calls the OMNI\$CONCLUDE procedure and supplies a VMD handle to indicate the association, an I/O status block, and an optional control structure for an event flag number and an AST.

For example, the following C code requests the conclusion of an association with the VMD identified by RecipientVmdHandle:

```
status = OMNI$CONCLUDE(  
    RecipientVmdHandle, /*Remote Vmd*/  
    &Iosb, /*I/O status block*/  
    Null); /*Control structure*/
```

The remote VMD can either accept or reject the conclude request.

The conclude request can complete successfully only if there are no outstanding requests on the association. If a request remains open, VSIomni returns an error message.

Note

In communications with certain MMS service providers, VSIomni may receive a reject PDU that does not contain an invoke ID to specify the request being rejected. In such cases, VSIomni logs the event, but the request remains outstanding. Because of the outstanding request, the application will be unable to conclude the association. To terminate an association with an outstanding request, the application must call OMNI\$ABORT.

3.5.3. Aborting an Association

An abort request indicates that the application wishes to discontinue communications on the association immediately and without negotiation. An abort will destroy outstanding requests and responses issued by either party in the association.

To issue an abort, the application calls the OMNI\$ABORT procedure and supplies a VMD handle to indicate the association, an I/O status block, and an optional control structure.

3.5.4. Receiving an Association Request

An application can receive a request from a remote VMD to establish an association with a local VMD. To receive this request, the application calls the OMNI\$LISTEN procedure and provides the following mandatory parameters:

- The handle of a local VMD. The application will receive requests for an association with this VMD.
- A location to receive the handle of the remote VMD initiating the association.

- One of the following flags:
 - 0. Indicates that VSIomni should reject an association request from a VMD whose ODF definition is not currently loaded (or from a VMD that does not identify itself as a VMD).
 - Non 0. Indicates that VSIomni should accept an association request from an undefined VMD. VSIomni creates a dummy (empty) definition and passes the handle to the application.

Optionally, the application can also supply a structure to receive connection values proposed for negotiation by the remote system (for example, the nesting level), an IOSB, and a control structure for asynchronous completion.

In the following C example:

- RecipientVmdHandle indicates the local VMD for which the application is receiving the association request.
- &RemoteVmdHandle is a location to receive the handle of the VMD initiating the association request.
- The flag indicates that the application is not willing to receive an association request from an initiating VMD whose ODF definition is not currently loaded.

In this example, the application is not providing a structure to receive proposed connection values.

```
status = OMNI$LISTEN(  
    RecipientVmdHandle, /*Local Vmd*/  
    FALSE, /*Translate Flag*/  
    &RemoteVmdHandle, /*Initiating Remote Vmd*/  
    NULL, /*Don't Receive Proposed Values*/  
    &Iosb, /*I/O Status Block*/  
    NULL ); /*No Control Structure*/
```

3.5.5. Responding to an Association Request

Once the association request has been received, the application can accept or reject the request.

To accept an association request, the application calls the OMNI\$ACCEPT_CONNECT procedure and supplies:

- The definition handle for the remote VMD requesting the association.
- A flag that indicates whether VSIomni can automatically accept a conclude request for the association or must pass the conclude indication to the application.

Optionally, the application can provide an IOSB to receive completion status and a control structure for asynchronous completion.

To reject an association request, the applications calls the OMNI\$REJECT_CONNECT procedure and supplies the definition handle for the remote VMD requesting the association.

3.5.6. Receiving a Conclude Request

At any point during the life of an association, the remote peer can issue a request to conclude the association.

To receive a conclude request, the application calls the OMNI\$GET_INDICATIONS procedure as described in Section 2.6

3.5.7. Responding to a Conclude Request

An application that receives a request to conclude an association can accept or reject the request.

To accept a conclude request, the application calls the OMNI\$ACCEPT_CONCLUDE procedure and specifies the definition handle of the remote VMD requesting the conclude.

To reject a conclude request, the application calls the OMNI\$REJECT_CONCLUDE procedure and specifies the definition handle of the remote VMD requesting the conclude.

3.6. VMD Support Services

MMS VMD support services allow an MMS client application to perform the following operations on a VMD object:

- Obtain the status of a VMD.
- Receive an unsolicited message about the status of a VMD.
- Obtain lists of defined objects.
- Identify vendor-specific attributes of the MMS application at the remote peer system.
- Obtain lists of the VMD's capabilities.

3.6.1. Obtaining VMD Status

The VMD status service allows a client application to determine the general condition or status of a VMD.

To obtain the status of a VMD, the application calls the OMNI\$GET_REMOTE_ATTRIBUTES procedure as described in Section 2.7 specifying the VMD and including the OMNI\$K_ATTR_ALL class constant in the parameter list.

Once VSIomni has retrieved the attributes and placed them in a local buffer, the application calls the OMNI\$GET_ATTRIBUTE procedure and specifies one of the attribute constants shown in Table 3.1

Table 3.1. Attribute Constants

Constant	Action
OMNI\$K_ATTR_LOGICAL_STATUS	VSIomni returns the logical status of the VMD.
OMNI\$K_ATTR_PHYSICAL_STATUS	VSIomni returns the physical status of the VMD.

3.6.2. Receiving Unsolicited Status

The unsolicited status service allows a VMD to spontaneously report its status to a client application.

To receive unsolicited status, the application calls the OMNI\$GET_INDICATIONS procedure as described in Section 2.6. If a status indication arrives, OMNI\$GET_INDICATIONS returns a status

code and the status context and completes. To obtain the status, the application calls the OMNI\$GET_ATTRIBUTE procedure and supplies the context and a status constant.

3.6.3. Obtaining a List of Objects

The MMS get name list service allows a client to obtain a list or part of a list of objects defined at the VMD.

To obtain an object name list, the application calls the OMNI\$GET_REMOTE_ATTRIBUTES procedure as described in Section 2.7 and supplies one of the constants shown in Table 3.2

Table 3.2. Constants and Action

Constant	Action
OMNI\$K_CLS_PI	VSIomni returns the list of program invocations for the specified VMD.
OMNI\$K_CLS_DOM	VSIomni returns the list of domains for the specified VMD.
OMNI\$K_CLS_VAR_LIST	VSIomni returns a variable list for the specified VMD.
OMNI\$K_CLS_NAMED_VAR	VSIomni returns a list of named variables for the specified VMD.

Once VSIomni has placed the list in a local buffer, the application calls OMNI\$GET_ATTRIBUTE with the OMNI\$K_ATTR_NAME constant to read the list.

3.6.4. Requesting VMD Identification

The MMS identify service allows a client application to obtain identifying information from a responding VMD.

To request identification, the application calls the OMNI\$GET_REMOTE_ATTRIBUTES procedure as described in Section 2.7 specifying the VMD and including the OMNI\$K_ATTR_ALL class constant.

Once VSIomni has placed the VMD attributes in a local buffer, the application calls OMNI\$GET_ATTRIBUTE and specifies one of the constants shown in Table 3.3

Table 3.3. VMD Constants

Constant	Action
OMNI\$K_ATTR_VENDOR	VSIomni returns the vendor name.
OMNI\$K_ATTR_MODEL	VSIomni returns the model number.
OMNI\$K_ATTR_REVISION	VSIomni returns the version number.

3.7. Domain Services

A domain represents a subset of the capabilities of a VMD. This subset is used for a specific purpose. A domain can be a sequence of instructions for a device or data, or both instructions and data.

Under VSIomni, a client application can request services to perform the following operations on domains:

- Download a domain to a VMD.

- Upload the contents of a domain from a VMD.
- Delete a domain.
- Obtain domain attributes.

In addition, the application can receive client requests to download and upload local domains.

3.7.1. Downloading a Domain

MMS download services allow a client application to instruct an MMS server to create the specified domain and to load the domain.

To download a domain, the application calls the OMNI\$DOWNLOAD procedure and passes the handle of the domain definition. The application can optionally pass the name of a file containing the domain to be read, the address of an IOSB, and a control structure for asynchronous completion. If the call does not include a file specification, VSIomni uses the domain contents file specified in the ODF definition of the domain.

VSIomni supports only one download to a domain at a time. VSIomni will reject an attempt to download a domain that already exists on the VMD.

If, on completion of the download service, the remote application discards the domain, VSIomni notifies the user in the IOSB.

The OMNI\$DOWNLOAD service translates into an MMS initiate download sequence service, a number of download segment services, and a terminate download sequence service.

3.7.2. Uploading a Domain

MMS upload services allows a client application to instruct the MMS server to upload the specified domain.

To upload a domain, the application calls the OMNI\$UPLOAD procedure and passes the handle of the domain definition. The application can optionally pass the name of a file to receive the domain to be written, the address of an IOSB, and a control structure for asynchronous completion. If the call does not include a specification for a file to receive the domain, VSIomni uses the file name associated with the ODF domain definition.

In an upload operation, VSIomni creates two files: a file containing the list of capabilities for the domain and a file containing the domain contents. For the capabilities file, VSIomni uses the file name in the ODF domain definition. The file name is a mandatory field in the ODF definition.

The OMNI\$UPLOAD procedure translates into an initiate upload sequence, a number of upload segments, and a terminate upload sequence.

3.7.3. Deleting a Domain

The delete domain service allows a client application to request that an MMS server delete the specified domain.

To request the deletion of a domain, the application calls the OMNI\$DELETE procedure and specifies the handle of the domain definition.

Deletions cannot occur if objects are currently referencing the domain.

3.7.4. Obtaining the Attributes of a Domain

The MMS get domain attributes service allows a client application to request that the MMS server return all the attributes associated with the specified domain.

To obtain the attributes of a domain, the application calls the OMNI\$GET_REMOTE_ATTRIBUTES procedure as described in Section 2.7 and provides a parameter list that includes.

- The name of the domain
- The class constant OMNI\$K_ATTR_ALL to obtain all of the attributes of the specified domain

VSIomni reads the attributes from the remote VMD and places them in a local buffer.

To read an attribute from the buffer, the application calls the OMNI\$GET_ATTRIBUTE procedure and includes one of the constants in the parameter list shown in Table 3.4

Table 3.4. Domain Constants

Constant	Meaning
OMNI\$K_ATTR_DELETABLE	Requests current value of MMS deletable attribute
OMNI\$K_ATTR_SHARABLE	Requests current value of MMS sharable attribute
OMNI\$K_ATTR_UPLD_IN_PROGRESS	Requests current value of MMS upload in progress attribute
OMNI\$K_ATTR_CAPABILITY	Requests list of domain capabilities
OMNI\$K_ATTR_REF_PI_NAMES	Requests a list of PIs linked to the domain
OMNI\$K_ATTR_STATE	Requests the current state of the domain

The application also includes a constant specifying the data type of the attribute value. VSIomni uses the constant to build a structure to contain the attribute value.

3.7.5. Receiving Download and Upload Client Requests

VSIomni can receive a client request from a remote application to download or upload the contents of a domain on the local system. VSIomni fulfills the request automatically. The application is not notified unless an error occurs, in which case VSIomni logs the error to the logging device.

3.8. Program Invocation Services

In an MMS system, a program invocation consists of a set of domains together with control information for its execution.

Program invocations may be dynamic in nature – coming into existence and being removed from the system either by the client or by local action on the server – or they may be predefined within the VMD.

MMS program invocation services allow a client application to perform the following operations on program invocation objects:

- Create a program invocation.
- Delete a program invocation.

- Start, stop, resume, reset, and kill the execution of a program invocation.
- Obtain the attributes of a program invocation.

3.8.1. Creating a Program Invocation

The MMS create program invocation service allows an MMS client application to assemble domains into a specific program invocation at the MMS server. The client specifies a list of domains that are to be included in the program invocation.

To request the program invocation service the application calls the OMNI\$CREATE procedure and passes the handle of a loaded PI definition.

3.8.2. Deleting a Program Invocation

The MMS delete program invocation service allows a client application to cause the deletion of an existing program invocation on a remote MMS server.

To request the delete program invocation service, the application calls the OMNI\$DELETE procedure and passes a PI definition.

3.8.3. Starting a Program Invocation

The MMS start service allows a client application to change the state of a program invocation from IDLE to RUNNING.

To start a program invocation on a remote VMD, the application calls the OMNI\$START procedure and passes the handle of the local definition of the program.

3.8.4. Stopping a Program Invocation

The MMS stop service allows a client application to change the state of a program invocation from RUNNING to STOPPED.

To stop a program invocation on a remote VMD, the application calls the OMNI\$STOP procedure and passes the handle of the PI definition.

3.8.5. Resuming a Program Invocation

The MMS resume service allows an MMS client to change the state of a program invocation from STOPPED to RUNNING.

To resume execution of a program invocation on a remote VMD, the application calls the OMNI\$RESUME procedure and passes the handle of the local definition of the program.

3.8.6. Resetting a Program Invocation

The MMS reset service allows a client application to change the state of a program invocation from STOPPED to IDLE or UNRUNNABLE, depending on the reusable attribute.

To reset a program invocation, the application calls the OMNI\$RESET procedure and specifies the handle of the definition of the program.

3.8.7. Killing a Program Invocation

The MMS kill service allows a client application to place a program invocation in the UNRUNNABLE state.

To kill a program invocation, the application calls the OMNI\$KILL procedure and specifies the handle of the program definition.

3.8.8. Getting Program Invocation Attributes

To obtain the attributes of a PI the application calls the OMNI\$GET_REMOTE_ATTRIBUTES procedure as describes in Section 2.7 and provides a parameter list that includes:

- The name of the PI.
- The class constant OMNI\$K_ATTR_ALL to obtain all of the attributes of the specified PI.

VSImni reads the attributes from the remote VMD and places them in a local buffer.

To read an attribute from the buffer, the application calls the OMNI\$GET_ATTRIBUTE procedure and includes one of the constants in the parameter list shown in Table 3.5

Table 3.5. PI Constants

Constant	Meaning
OMNI\$K_ATTR_REUSABLE	Requests current value of MMS reusable attribute
OMNI\$K_ATTR_MONITOR	Requests current value of MMS monitor attribute
OMNI\$K_ATTR_REF_DOM_NAMES	Requests a list of domains associated with the PI
OMNI\$K_ATTR_STATE	Requests the current state of the PI
OMNI\$K_ATTR_DELETABLE	Requests the current value of the deletable attribute
OMNI\$K_EXEC_ARG	Requests an execution argument for the PI

The application also includes a constant specifying the data type of the attribute value. VSImni uses the constant to build a structure to contain the attribute value.

3.9. Variable Access Services

MMS variable access services allow a client application to obtain access to typed variables defined at the VMD.

In an MMS system, access to variables is obtained through variable objects. A variable object describes the attributes of one typed variable in the VMD and provides an access path to the variable.

MMS defines five classes of variable objects:

- Unnamed variable objects. Under VSImni, an application can read and write the values described by unnamed variable objects and obtain attributes and information reports (see Section 3.9.1)
- Named variable objects. Under VSImni, an application can create and delete named variable objects, read and write values, and obtain attributes and information reports (see Section 3.9.2)

- Scattered access objects. VSIomni does not currently provide support for scattered access.
- Named variable list objects. Under VSIomni, an application can create and delete named variable list objects, read the list, and obtain attributes of the list (see Section 3.9.3)

VSIomni also allows an application to receive client service requests to read and write local variables. Section 3.9.4 describes the way VSIomni delivers the requests and how the application reads, writes, and transmits the values.

3.9.1. Operating on Unnamed Variable Objects

An unnamed variable object describes a variable existing at a known and fixed address. The existence of an unnamed variable object is inherent in the VMD. The client can neither create nor delete an unnamed variable object.

To read the value of a variable described by an unnamed variable object, the application calls the OMNI\$GET_VALUE procedure and specifies the handle of a definition for the unnamed variable object.

To replace the current value of a variable described by an unnamed variable object, the application calls the OMNI\$PUT_VALUE procedure and supplies the handle of the definition for the unnamed variable.

To receive an unsolicited inforeport indication, the application calls the GET\$INDICATIONS procedure. To obtain the current value of the variable contained in the inforeport, the application calls the GET\$VALUE procedure and passes the context information returned by GET\$INDICATIONS.

To obtain the type description of a variable described by an unnamed variable object, the application calls the OMNI\$GET_REMOTE_ATTRIBUTES and OMNI\$GET_ATTRIBUTE procedures.

3.9.2. Operating on Named Variable Objects

A named variable object describes an application-defined variable at the VMD.

To read the value of a variable described by a named variable object, the application calls the OMNI\$GET_VALUE procedure and specifies the handle of a definition for the named variable object.

To replace the current value of a variable described by a named variable object, the application calls the OMNI\$PUT_VALUE procedure and specifies the handle of the variable definition.

To receive an unsolicited inforeport indication, the application calls the OMNI\$GET_INDICATIONS procedure. To obtain the current value of the variable contained in the inforeport, the application calls the OMNI\$GET_VALUE procedure and passes the context information returned by OMNI\$GET_INDICATIONS.

To create a named variable object, the application calls the OMNI\$CREATE procedure.

To obtain the attributes of a variable described by a named variable object, the application calls the OMNI\$GET_REMOTE_ATTRIBUTES and OMNI\$GET_ATTRIBUTE procedures.

To delete a named variable object, the application calls the OMNI\$_DELETE procedure and supplies the handle of the definition.

3.9.3. Operating on Named Variable List Objects

The named variable list object allows an MMS client to assign a name to a list of variable objects describing independent MMS variables. The client uses this name to specify access to all the variables

described in the list. The list can include named variable objects and unnamed variable objects. Access to variables using a named variable list object is analogous to independent accesses using the referenced variable objects in the list.

To create a variable list object, the application calls the OMNI\$CREATE procedure and specifies the handle of the definition of the list.

To obtain the list of variable attributes, the application calls the OMNI\$GET_REMOTE_ATTRIBUTES and OMNI\$GET_ATTRIBUTE procedures.

3.9.4. Receiving Client Requests for Variable Access Services

During the life of an association, an application can receive client requests to read or write a variable on a local VMD.

Note

Unnamed variables are not locally supported.

3.9.4.1. Receiving and Fulfilling a Read Request

To receive a client read request for a specific local VMD, the application calls the OMNI\$GET_INDICATION procedure and supplies the name of the VMD. When a request arrives to read a value on the specified VMD, VSImni delivers the following information:

- The handle of the variable object definition
- The indication type – in this case, a read indication
- Association context information

The application uses the variable object definition handle to determine which variable to read. The application locates and obtains the value. (How this is done lies outside the scope of VSImni.)

To send the value to the remote application, the application calls the OMNI\$PUT_VALUE and supplies the value and context information returned on the OMNI\$GET_INDICATION call. The context information tells VSImni where to send the value.

3.9.4.2. Receiving and Fulfilling a Write Request

To receive a client write request for a specific local VMD, the application calls the OMNI\$GET_INDICATION procedure and supplies the name of the VMD. When a request arrives to modify a value on the specified VMD, VSImni delivers the following information:

- The handle of the variable object definition
- The indication type – in this case, a write indication
- Association context information

To obtain the updated value from the remote application, the application calls the OMNI\$GET_VALUE and supplies the context information returned on the OMNI\$GET_INDICATION call. The context information tells VSImni where to get the value.

The application uses the variable object definition handle to determine which variable on the VMD to modify. The application locates and modifies the value. (How this is done lies outside the scope of VSImni.)

3.10. File Services

VSImni provides the following set of procedures that an application can call to operate on files on a remote MMS system:

- OMNI\$FDELETE. Deletes a file from a remote MMS system.
- OMNI\$FDIR. Obtains a list of file specifications from a remote MMS system and places the list in a local buffer. To read a file specification, the application calls the OMNI\$GET_ATTRIBUTE procedure with the OMNI\$K_ATTR_NAME constant.
- OMNI\$FGET. Copies a file from a remote MMS system to the local system.
- OMNI\$FPUT. Copies a file from the local system to a remote MMS system.
- OMNI\$FRENAME. Renames a file on a remote MMS system.

Chapter 4. VSIomni Procedure Calls

This chapter describes the format and arguments of VSIomni procedure calls.

Each description of a call consists of the following sections:

- Procedure call format (see the section called “Procedure Call Format Section”)
- Returns (see the section called “Returns Section”)
- Argument definitions (see the section called “Argument Definitions Section”)
- Usage notes (see the section called “Usage Notes Section”)

Procedure Call Format Section

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 and their meaning are shown in Table 4.1

Table 4.1. Elements

Element	Meaning
<i>status</i>	A location to receive a longword condition value that the procedure returns to the caller.
<i>procedure</i>	A VSIomni procedure.
<i>_A</i>	A suffix to specify asynchronous operation of the requested service.
<i>arg1</i> , <i>[arg2]</i> . . . , <i>[argn]</i>	A list of required and optional arguments.
<i>[. . .]</i>	Square brackets, used to indicate that the enclosed element is optional. In the general format example, <i>arg2</i> and <i>argn</i> 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 VSIomni.

Returns Section

The returns section describes the type of information that the called procedure returns to the caller. All VSIomni procedures return a longword condition value to R0.

The run-time environment established for programs written in high-level languages allows the status information in R0 to be moved automatically to the *status* variable specified in the call.

Argument Definitions Section

The argument definitions section provides detailed information about each argument listed in the procedure call. A complete definition includes the following entries:

- Data type (see the section called “Data Type Entry”)
- Argument access (see the section called “Argument Access Entry”)
- Passing mechanism (see the section called “Passing Mechanism Entry”)

Data Type Entry

The data type entry is a code indicating the VMS data type of the data that is specified by the argument.

VSIomni defines the data type for most arguments used in VSIomni procedure calls.

For a list of VAX standard data types and data type codes, see the *VSI OpenVMS System Services Reference Manual*.

Argument Access Entry

The argument access entry indicates the way that the called procedure accesses the data specified by the argument. VSIomni procedures require three types of access to the arguments in VSIomni calls: read-only access, write-only access, and modify (or read/write) access.

Read-Only Access

The called procedure reads data from the location specified by the argument.

Data on which the procedure operates, or data needed by the procedure to perform its operation, must be read by the called procedure. Such data is also called input data. When an argument specifies input data, the access entry is read only.

The term **only** indicates that the called procedure does not both read and write (that is, modify) the input data. Thus, input data is preserved when the called procedure completes execution.

Write-Only Access

The called procedure writes data into the location specified by the argument.

Data that the called procedure returns to the calling application must be written into a location where the calling program can access it. Such data is also called output data. When an argument specifies output data, the access entry is write only.

The term **only** indicates that the called procedure does not read the contents of the location either before or after it writes into the location.

Modify Access

The called procedure reads data from the location specified by the argument and then overwrites the location.

When an argument specifies data that is both read and written by the called procedure, the access entry is **modify**. In this case, the called procedure reads the input data, uses it, and then overwrites the input data with the results (the output data) of the operation. Thus, when the called procedure completes execution, the input data specified by the argument is lost.

Passing Mechanism Entry

The passing mechanism entry indicates whether the longword argument is data, the address of data, or a data descriptor.

If the argument is a location containing data that the called procedure reads or if it is a location to contain data that the procedure writes, that data is said to be passed by value. Note that since an argument is only one longword in length, only data that can be contained in one longword can be passed by value.

If the argument is the address of a location containing data that the called procedure reads or if it is the address of a location to contain data that the procedure writes, that data is said to be passed by reference.

If the argument is the address of a descriptor for a buffer containing data that the called procedure reads or if it is the address of a descriptor for a buffer to contain data that the procedure writes, that data is said to be passed by descriptor. A descriptor consists of two or more longwords (depending on the type of descriptor used), that describe the location, length, and type of the data that is being passed.

Allocating Text String Buffers For OMNI\$ Calls

Some of the parameters in OMNI\$ calls require text strings be passed into or out of various subroutines.

The VSImni include file, OMNI\$DEFS contains definitions for each type of string data that is passed into or out of the OMNI\$ calls. The length of each string type is provided with two constant definitions of the form:

```
OMNI$K_XXXXXXXX_SIZEOMNI$K_XXXXXXXX_MAX
```

where XXXXXXXX is the name of the string data, for example MMS_ID.

The string MMS_ID has the following two constant definitions from OMNI\$DEFS:

```
OMNI$K_MMS_ID_SIZE  
OMNI$K_MMS_ID_MAX
```

The constant OMNI\$K_MMS_ID_SIZE specifies the size of the buffer that should be allocated for MMS_ID, and the constant OMNI\$K_MMS_ID_MAX specifies the usable portion of the buffer allocated for MMS_ID.

When allocating buffer space and using text strings for the various OMNI\$ calls always use the OMNI \$K_XXXXXXX_SIZE constant of a string to allocate memory. The OMNI\$K_XXXXXXX_MAX portion of that allocated memory is the usable data.

Usage Notes Section

Some call descriptions include optional usage notes. These notes describe restrictions, exceptions, and other function-specific matters.

OMNI\$ABORT

OMNI\$ABORT — Immediately terminates an association with a remote VMD.

Format

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$ABORT procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

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

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

All pending requests will return with status OMNI\$ABORT.

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]
```

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$ACCEPT_CONCLUDE procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The longword identifier of a loaded VMD definition.

The *vmd_handle* is returned by the OMNI\$LOAD_DEFINITIONS procedure.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

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],

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$ACCEPT_CONNECT procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The longword identifier of a loaded VMD definition.

The *vmd_handle* is returned by the OMNI\$LOAD_DEFINITIONS procedure.

reserved

Reserved for future use.

conclude_flag

type: longword

access: read only

mechanism: see Appendix B

One of the values shown in Table 4.2

Table 4.2. Value and Meaning

Value	Meaning
non 0	VSImni delivers all conclude indications to the calling application for processing.
0	VSImni automatically accepts conclude requests.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSImni I/O status block. For a description of the codes that VSImni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$CANCEL

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

Format

`status=OMNI$CANCEL[_A]invoke_id, [omni_iosb], [ctrl_struct]`

Returns

VMS Usage: cond_value

type: longword(unsigned)

access: write only

mechanism: by value in R0

The OMNI\$CANCEL returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

Invoke_id is an identifier assigned by VSIomni. The parameter is used for asynchronous calls only.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

Omni_iosb is the VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A. The *omni_iosb* parameter is the address of the status block.

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$CONCLUDE

OMNI\$CONCLUDE — Brings an association with a remote VMD to an orderly conclusion.

Format

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$CONCLUDE procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The longword identifier of a loaded VMD definition.

The *vmd_handle* is returned by the OMNI\$LOAD_DEFINITIONS.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

1. It is valid to conclude an association only when all requests have been satisfied.
2. If there are pending operations on the association, VSIomni will not accept an OMNI\$CONCLUDE request.

OMNI\$CONNECT

OMNI\$CONNECT — Initiates an association with a remote VMD.

Format

```
status=OMNI$CONNECT[_A] vmd_handle,reserved, [calling_vmd_handle], [negotiated_vmd_handle]
```

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only mechanism: by value in R0

The OMNI\$CONNECT procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The longword identifier of a loaded VMD definition.

The *vmd_handle* is returned by the OMNI\$LOAD_DEFINITIONS procedure.

reserved

Reserved for future use.

calling_vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The handle of a VMD that the application wishes 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 VSIomni to service network objects - such as variables - for the user.

negotiated_vmd_struct

type: OMNI\$R_VMD_DEF

access: write only

mechanism: see Appendix B

A VMD data structure to receive negotiated service parameters.

conclude_flag

type: longword

access: read only

mechanism: see Appendix B

One of the values shown in Table 4.3 to specify the way OMNI handles conclude indications.

Table 4.3. Conclude Values

Value	Meaning
non 0	VSIomni delivers all conclude indications to the calling application for processing.
0	VSIomni automatically accepts conclude requests.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

The calling VMD specifies network objects.

OMNI\$CREATE

OMNI\$CREATE — Creates an object on a VMD.

Format

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$CREATE procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSIomni. This parameter is used for asynchronous calls only.

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

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

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$DEFINE

OMNI\$DEFINE — Creates a definition of the specified class and fills a user buffer with the value of the definition handle. In all cases, the value of the definition's scope attribute must be modified (using OMNI\$MODIFY_DEFINITION) before it is recognized by VSIomni.

Format

status=OMNI\$DEFINE(class, handleaddress)

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

Arguments

class

type: OMNI\$L_ENUMERATION_CONST

access: read only

mechanism: see Appendix B Class identifies the class of definition to create.

The value of the class parameter is one of the values shown in Table 4.4

Table 4.4. 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

handleaddress

type: OMNI\$L_HANDLE

access: write only

mechanism: see Appendix B

HandleAddress specifies the return address of the definition handle.

OMNI\$DELETE

OMNI\$DELETE — Deletes a specific object on a VMD. **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.**

Format

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

Returns

VMS Usage: `cond_value`

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$DELETE procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSIomni. This parameter is used for asynchronous calls only.

object_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded variable definition

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

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.

OMNI\$DOWNLOAD

OMNI\$DOWNLOAD — Initiates the download- ing of a domain to a remote VMD.

Format

status=OMNI\$DOWNLOAD[_A] [invoke_id],domain_handle, [domain_file], [modifier_objec

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$DOWNLOAD procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImni. This parameter is used for asynchronous calls only.

domain_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded domain definition.

domain_file

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

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.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSImni I/O status block. For a description of the codes that VSImni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

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

OMNI\$END_LIST

OMNI\$END_LIST — Terminates the use of a list context.

Format

status=OMNI\$END_LISTcontext

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$END_LIST procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

context

type: OMNI\$R_CONTEXT

access: read only

mechanism: see Appendix B

A location for use by VSImni. The *context* is the same value that was returned by OMNI\$_GET_REMOTE_ATTRIBUTES.

Usage Notes

OMNI\$GET_REMOTE_ATTRIBUTES allocates memory. OMNI\$END_LIST frees that same memory. Failure to call OMNI\$END_LIST will result in an increase of memory usage.

OMNI\$FDELETE

OMNI\$FDELETE — Deletes a file from a remote system.

Format

status=OMNI\$FDELETE[_A] [invoke_id], vmd_handle, remote_file_name, [modifier_

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$FDELETE procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImni. The parameter is used for asynchronous calls only.

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Longword identifier of a previously loaded VMD definition.

The VMD handle is returned by the OMNI\$LOAD_DEFINITIONS procedure.

remote_file_name

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

The file specification for the remote file to delete.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSImni I/O status block. For a description of the codes that VSImni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

1. Wild cards will be delivered to the remote device as specified.
2. See PIC for remote device specification.

OMNI\$FDIR

OMNI\$FDIR — Obtains a list of file specifications from a remote directory.

Format

status=OMNI\$FDIR[_A] [invoke_id], vmd_handle, remote_directory, context, [modifier

Returns

VMS Usage: cond_value type:

longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$FDIR procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImoni. The parameter is used for asynchronous calls only.

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Longword identifier of a previously loaded VMD definition.

The VMD handle is returned by the OMNI\$LOAD_DEFINITIONS procedure.

remote_directory

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

A character string (in native format) representing the remote directory.

context

type: OMNI\$R_CONTEXT

access: write only

mechanism: see Appendix B

A location for use by VSIomni.

The *context* parameter is the address of the context location.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

1. OMNI\$FDIR obtains a directory. To obtain the file specifications in the directory, call the OMNI\$GET_ATTRIBUTE procedure for each filespec.
2. OMNI\$FDIR allocates space for a buffer to contain the directory. To release the buffer, call OMNI\$END_LIST.

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$FGET procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImoni. The parameter is used for asynchronous calls only.

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Longword identifier of a loaded VMD definition. The VMD handle is returned by the OMNI \$LOAD_DEFINITIONS procedure.

remote_file_name

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

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

local_file_name

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

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

size

type: OMNI\$L_SIZE_OF_FILE

access: write only

mechanism: see Appendix B

Size of the file on the remote device.

last_modified

type: OMNI\$L_LAST_MOD_DATE

access: write only

mechanism: see Appendix B

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

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSImni I/O status block. For a description of the codes that VSImni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

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

OMNI\$FPUT

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

Format

status=OMNI\$FPUT[_A] [invoke_id], vmd_handle, remote_file_name, local_file_name

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$FPUT procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSIomni. The parameter is used for asynchronous calls only.

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Longword identifier of a loaded VMD definition. The VMD handle is returned by the OMNI \$LOAD_DEFINITIONS procedure.

remote_file_name

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

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

local_file_name

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

The file specification for the local file to copy.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

The local file must be an FTAM-3 documentation type.

OMNI\$FRENAME

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

Format

status=OMNI\$FRENAME[_A] [invoke_id], vmd_handle, original_remote_file_name, r

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$FRENAME procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImni. The parameter is used for asynchronous calls only.

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Longword identifier of a loaded VMD definition. The VMD handle is returned by the OMNI \$LOAD_DEFINITIONS procedure.

original_remote_file_name

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

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

new_remote_file_name

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

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

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

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_struct

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$GET_ATTRIBUTE procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

context

type: OMNI\$R_CONTEXT

access: read only

mechanism: see Appendix B

A context value returned by the OMNI\$GET_REMOTE_ATTRIBUTES or OMNI\$FDIR procedure.

object_attribute

type: longword (unsigned)

access: read only

mechanism: see Appendix B

One of the constants listed in Table 4.5 to specify the attribute to read. See usage note 2.

Table 4.5. Attribute Constants and Associated Data Types

Constants	Data Type
General Constants	
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_SHARABLE	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_STATE	OMNI \$L_ENUMERATION_CONSTANT
OMNI\$K_ATTR_CAPABILITY	OMNI\$T_CAPABILITY
OMNI\$K_ATTR_ADDITIONAL_DETAIL	OMNI\$L_HANDLE
VMD Constants	
OMNI\$K_ATTR_VENDOR	OMNI\$T_VENDOR
OMNI\$K_ATTR_MODEL	OMNI\$T_MODEL
OMNI\$K_ATTR_REVISION	OMNI\$ _REVISION
OMNI\$K_ATTR_SYNTAX_LIST	OMNI\$ _MMS_ID
OMNI\$K_ATTR_LOGICAL_STATUS	Longword
OMNI\$K_ATTR_PHYSICAL_STATUS	Longword
OMNI\$K_ATTR_LOCAL_DETAIL	OMNI \$B_STATUS_LOCAL_DETAIL
Domain Constants	
OMNI\$K_ATTR_UPLD_IN_PROGRESS	OMNI\$B_UPLD_IN_PROG
OMNI\$K_ATTR_REF_PI_NAMES	OMNI\$T_MMS_ID
PI Constants	
OMNI\$K_ATTR_REUSABLE	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_MONITOR	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_EXEC_ARG	OMNI\$T_EXEC_ARG_STR
OMNI\$K_ATTR_REF_DOM_NAMES	OMNI\$T_MMS_ID
Variable Constants	
OMNI\$K_ATTR_MMS_TYPE_SPEC	OMNI \$R_MMS_TYPE_SPEC_MMS
OMNI\$K_ATTR_ADDRESS	OMNI\$R_ADDRESS

receive_struct

type: depends on attribute specified

access: write only

mechanism: see Appendix B

One of the data type codes in Table 4.6 to indicate the data type of the attribute. VSlomni uses the code to construct a location to contain the returned attribute value.

Table 4.6. VMD Attributes and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_APPL_SIMPLE_NAME	OMNI\$T_APPL_SIMPLE_NAME_WC
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_MAX_SEGMENT	Long Integer
OMNI\$K_ATTR_MAX_SRV_CALLED	Word
OMNI\$K_ATTR_MAX_SRV_CALLING	Word
OMNI\$K_ATTR_MODEL	OMNI\$T_MODEL
OMNI\$K_ATTR_NESTING	Byte
OMNI\$K_ATTR_PARAM_SUPPORTED	OMNI\$B_PARAM_CBB
OMNI\$K_ATTR_REVISION	OMNI\$T_REVISION
OMNI\$K_ATTR_SRV_SUPPORTED	OMNI\$B_SERVICES_SUPPORTED
OMNI\$K_ATTR_VENDOR	OMNI\$T_VENDOR
OMNI\$K_ATTR_VERSION	Word
OMNI\$K_CLS_DOM	OMNI\$T_MMS_ID
OMNI\$K_CLS_PI	OMNI\$T_MMS_ID
OMNI\$K_CLS_NAMED_VAR	OMNI\$T_MMS_ID
OMNI\$K_CLS_UNNAMED_VAR	OMNI\$T_MMS_ID
OMNI\$K_CLS_MMS_NAMED_TYPE	OMNI\$T_MMS_ID
OMNI\$K_CLS_APP_NAMED_TYPE	OMNI\$T_MMS_ID
OMNI\$K_CLS_MESSAGE	OMNI\$T_MMS_ID

Usage Notes

1. The receive structure is based on the type of attribute.
2. 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. OMNI\$GET_DEFINITION retrieves the values of both single-valued and multi-valued attributes. In the case of multi-valued 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\$_ENDOF LIST is returned as the status value. The OMNI\$END_LIST routine should be called after a list of values has been retrieved.

Format

status=OMNI\$GET_DEFINITION (defhandle,attraddress, contextaddress, value)

Returns

VMS Usage: cond_value

type: longword

access: write only

mechanism: by value in R0

Arguments

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

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.

contextaddress

type: OMNI\$L_CONTEXT

access: modify

mechanism: by reference

Contextaddress is the address of a variable. This parameter is used only if you retrieve the value of a multi-valued attribute.

The value of contextaddress should be initialized to NULL. In general, the value of the context should be modified when as many values as required have been retrieved from a multi-valued attribute. In this case, OMNI\$END_LIST should be called to free any unneeded space allocated by VSImni. The value of the context should be then reset to zero.

value

type: see table

access: write only

mechanism: by reference

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

The attributes supported for each definition class and the expected type of the buffer to receive the value are listed in Table 4.7 through Table 4.16

Table 4.7 shows Domain data information.

Table 4.7. Domain Attributes and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_CAPABILITY_FILE	OMNI\$T_FILENAME
OMNI\$K_ATTR_CAPABILITY	OMNI\$T_CAPABILITY
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION_WC
OMNI\$K_ATTR_DOM_CONTENT_FILE	OMNI\$T_FILENAME
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_SHARABLE	OMNI\$B_BOOLEAN
OMNI\$K_CLS_NAMED_VAR	OMNI\$T_MMS_ID
OMNI\$K_CLS_UNNAMED_VAR	OMNI\$T_MMS_ID
OMNI\$K_CLS_MMS_NAMED_TYPE	OMNI\$T_MMS_ID

Attribute	Expected Data Type
OMNI\$K_CLS_APP_NAMED_TYPE	OMNI\$T_MMS_ID
OMNI\$K_ATTR_USER_PARAM	Longword

Table 4.8 shows PI data information.

Table 4.8. PI Attributes and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_REF_DOM_NAMES	OMNI\$T_MMS_ID
OMNI\$K_ATTR_EXEC_ARG_STR	OMNI\$T_EXEC_ARG_STR
OMNI\$K_ATTR_REUSABLE	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_MONITOR	OMNI\$L_ENUMERATION_CONST
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_USER_PARAM	Longword

Table 4.9 shows Named Variable data information.

Table 4.9. Named Variable Attributes and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$R_MMS_TYPE_SPEC_DEF
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$R_APP_TYPE_SPEC_DEF
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_USER_PARAM	Longword
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN

Table 4.10 shows Unnamed Variable data information.

Table 4.10. Unnamed Variable Attributes and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$R_MMS_TYPE_SPEC_DEF
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$R_APP_TYPE_SPEC_DEF
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_USER_PARAM	Longword
OMNI\$K_ATTR-SUPPLY_TYPE_SPEC	OMNI\$B_BOOLEAN

Attribute	Expected Data Type
OMNI\$K_ATTR_ADDRESS_TYPE	OMNI\$L_ENUMERATION_CONST
OMNI\$K_ATTR_ADDRESS_STRING	OMNI\$T_ADDRESS_STR
OMNI\$K_ATTR_ADDRESS_NUMBER	Longword

Table 4.11 shows MMS Named Type data information.

Table 4.11. MMS Named Type Attributes and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$R_MMS_TYPE_SPEC
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_USER_PARAM	Longword
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN

Table 4.12 shows MMS Type Specification data information.

Table 4.12. MMS Type Specification Attributes and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$R_MMS_TYPE_SPEC
OMNI\$K_ATTR_USER_PARAM	Longword
OMNI\$K_ATTR_STRUCT_COMPS	OMNI\$L_HANDLE
OMNI\$K_ATTR_ARRAY_ELEM_MMS_TYPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE

Usage Notes

1. The value of the scope attribute will be the handle of the only MMS definition or the last Application definition to refer to the MMS Type Specification definition. Table 4.13 shows Application Named Type data information.

Table 4.13. Application Named Type and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$R_APP_TYPE_SPEC
OMNI\$K_ATTR_MMS_NAMED_TYPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_USER_PARAM	Longword
OMNI\$K_ATTR_ALT_ACCESS	OMNI\$B_BOOLEAN

Table 4.14 shows Application Type Specification data information.

Table 4.14. Application Type Specification and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$R_APP_TYPE_SPEC
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_USER_PARAM	Longword
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$R_MMS_TYPE_SPEC
OMNI\$K_ATTR_STRUCT_COMPS	OMNI\$L_HANDLE
OMNI\$K_ATTR_ARRAY_ELEM_APP_TYPE	OMNI\$L_HANDLE

- The value of the scope attribute is the handle of the definition that refers to the Application Type Specification definition.

Table 4.15 shows MMS Structure Component data information.

Table 4.15. MMS Structure Component and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$R_MMS_TYPE_SPEC
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE
OMNI\$K_ATTR_USER_PARAM	Longword

Table 4.16 shows Application Structure Component data information.

Table 4.16. Application Structure Component and Expected Data Types

Attribute	Expected Data Type
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION
OMNI\$K_ATTR_MMS_TYPE_COMP	OMNI\$L_HANDLE
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$R_APP_TYPE_SPEC
OMNI\$K_ATTR_USER_PARAM	Longword
OMNI\$K_ATTR_BYTE_OFFSET	Longword
OMNI\$K_ATTR_ALTERNATE_ACCESS	OMNI\$B_BOOLEAN
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$GET_HANDLE_BY_NAME procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

scope

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

A handle indicating the scope of the search. The *scope* parameter is typically the handle of a VMD or a domain. VSImni 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).

class

type: OMNI\$L_ENUMERATION_CONST

access: read only

mechanism: see Appendix B

One of the constants shown in Table 4.17 to indicate the class of definition to search for.

Table 4.17. Class Constants

Constant	Meaning
OMNI\$K_CLS_DOM	Domain
OMNI\$K_CLS_PI	Program invocation
OMNI\$K_CLS_VAR	First variable type found
OMNI\$K_CLS_NAMED_VAR	Named variable

Constant	Meaning
OMNI\$K_CLS_UNNAMED_VAR	Unnamed variable
OMNI\$K_CLS_VAR_LIST	Variable list
OMNI\$K_CLS_MMS_NAMED_TYPE	MMS Named Type
OMNI\$K_CLS_APP_NAMED_TYPE	Application Named Type

def_name

type: character-coded text string

access: read only

mechanism: see Appendix B

The name of the definition to search for.

handle

type: OMNI\$L_HANDLE

access: write only

mechanism: see Appendix B

A location to receive the definition handle.

OMNI\$GET_HANDLE_LIST

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

Format

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$GET_HANDLE_LIST procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

scope

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

A handle indicating the scope of the search. The *scope* parameter is typically the handle of a domain or VMD, obtained from an earlier call of OMNI\$LOAD_DEFINITIONS, OMNI\$GET_HANDLE_BY_NAME, or OMNI\$GET_HANDLE_LIST.

class

type: OMNI\$L_ENUMERATION_CONST

access: read only

mechanism: by reference

One of the constants shown in Table 4.18 (defined in OMNI\$DEFS) to indicate the class of the definition whose handle is to be retrieved.

Table 4.18. Definition Class Constants

Constant	Definition Class
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_MMS_NAMED_TYPE	MMS Named Type
OMNI\$K_CLS_APP_NAMED_TYPE	Application Named Type
OMNI\$K_CLS_MSG	Message

context

type: OMNI\$L_CONTEXT

access: modify

mechanism: see Appendix B

A variable for use by VSImni.

The value of this variable is modified by VSImni. It should be initialized to NULL before the first call and should not be modified by the caller until the value of the return status is OMNI\$ENDOFLIST or until OMNI\$END_LIST has been called.

handle

type: OMNI\$L_HANDLE

access: write only

mechanism: see Appendix B

A variable of the type OMNI\$L_HANDLE to receive the definition handle.

Usage Notes

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 may 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,

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$GET_INDICATIONS procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: by value

The longword identifier of a loaded VMD definition, indicating the remote VMD for which indications are to be received. The *vmd_handle* is returned by the OMNI\$LOAD_DEFINITIONS procedure. The *vmd_handle* parameter is the handle.

def_handle

type: OMNI\$L_HANDLE

access: write only

mechanism: see Appendix B

An object definition handle returned by VSIomni. This is the handle of the object referred to by the indication (for example, a read indication would be accompanied by the handle of the variable to read). If the indication does not refer to an object (for example, a conclude indication), this parameter is null on completion.

context

type: OMNI\$R_CONTEXT

access: write only

mechanism: see Appendix B

Context information returned by VSIomni to be used in subsequent calls.

indication_type

type: OMNI\$L_ENUMERATION_CONST

access: write only

mechanism: see Appendix B

One of the following constants listed in Table 4.19 to indicate the type of indication received.

Table 4.19. OMNI\$GET_INDICATIONS Function Values

Value	Meaning
OMNI\$K_IND_ACSE_CONCL	Incoming conclude has completed.
OMNI\$K_IND_PREP_CONCL	Incoming conclude has been received. User must accept or reject the conclude.
OMNI\$K_IND_ABORT	Incoming abort has been received.
OMNI\$K_IND_READ	Incoming read request has been received.
OMNI\$K_IND_WRITE	Incoming write request has been received.
OMNI\$K_IND_UNSOL_STATUS	Incoming status report has been received.
OMNI\$K_IND_INFO_REPORT	Incoming information report has been received.

reserved

A place holder.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

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, [mod

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$GET_REMOTE_ATTRIBUTES procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSIomni. This parameter is used for asynchronous calls only.

def_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded definition of the object whose attribute values the application wishes to obtain.

class

type: OMNI\$L_ENUMERATION_CONST

access: read only

mechanism: see Appendix B

One of the values in Table 4.20 to specify the class of attributes to retrieve.

Table 4.20. Attributes Retrieved

Constant	Meaning
OMNI\$K_ATTR_ALL	The value of all attributes for the specified object.
OMNI\$K_CLS_DOM	The list of domains for the specified VMD.
OMNI\$K_CLS_PI	The list of program invocations for the specified VMD.
OMNI\$K_CLS_NAMED_VAR	The list of named variables for the specified VMD.
OMNI\$K_CLS_VAR_LIST	A variable list for the specified VMD.

context

type: OMNI\$R_CONTEXT

access: write only

mechanism: see Appendix B

A context value returned by VSIomni. The application uses this value in calls to the OMNI \$GET_ATTRIBUTE procedure to specify the buffer containing the retrieved attributes.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$GET_VALUE

OMNI\$GET_VALUE — 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_s

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$GET_VALUE procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImni. This parameter is used for asynchronous calls only.

object_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

In a client call, the identifier of a loaded variable definition.

In a server call, the context value returned by the GET\$INDICATIONS procedure call that delivered the write indication.

method_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Method_handle modifies the default presentation of a variable.

receiving_structure

type: depends on value

access: write only

mechanism: see Appendix B

A data structure to receive the value of the object.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$GROUP_VARIABLES

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

Format

status=OMNI\$GROUP_VARIABLES(contextaddress, object_handle, method_handle, pdata,mo

Returns

VMS Usage: cond_value

type: longword(unsigned)

access: write only

mechanism: by value in R0

The OMNI\$GROUP_VARIABLES procedure returns the Bad_Parm code.

Description

To use this procedure, do the following:

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 iosb 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.

Arguments

contextaddress

type: OMNI\$L_CONTEXT

access: modify

mechanism: by reference

Context is the address of a variable. This parameter is used by VSImni only if you modify a multi-value attribute.

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

object_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

In a client call, the identifier of a loaded variable definition.

In a server call, the context value returned by the GET\$INDICATIONS procedure call that delivered the write indication.

method_handle

type: OMNI\$HANDLE

access: read only

mechanism: see Appendix B

Method_handle modifies the default presentation of a variable.

pdata

type: depends on value

access: read only

mechanism: see Appendix B

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.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

Usage Notes

In the the section called “Example” segment code, variables A, B and C are grouped together using IosbA, IosbB and IosbC for the variable iosb's. The general transaction iosb used is TransIosb.

If transiosb returns with a general error then the get value request did not complete properly. If the request completed properly, iosbA, iosbB and iosbC indicates whether the read of each of those variables completed successfully. If variable "A" was read properly, the data returned will be in pdataA. If variable "B" was read properly, the data returned is pdataB; and if variable ``C" was read properly, the data returned is pdataC.

Example

Grouping Variables

```
OMNI$R_CONTEXT context = 0;
OMNI$R_IOSB IosbA, IosbB, IosbC, TransIosb;
/* group variable 'A' */
status = OMNI$GROUP_VARIABLES(
    &context,
    objecthandleA,
    altAccesshandleA,
```

```
        pdataA,
        modifierhandleA,
        &iosbA );
/* group variable 'B' */
status = OMNI$GROUP_VARIABLES(
        &context,
        objecthandleB,
        altAccesshandleB,
        pdataB,
        modifierhandleB,
        &iosbB );
/* group variable 'C' */
status = OMNI$GROUP_VARIABLES(
        &context,
        objecthandleC,
        altAccesshandleC,
        pdataC,
        modifierhandleC,
        &iosbC );
/* issue the get value request for 'A', 'B' and 'C' */
status = OMNI$GET_VALUE(
        &invokeid,
        context,
        NULL,
        NULL,
        NULL,
        &transiosb,
        &ctrlblock );
status = OMNI$END_LIST(
        &context
);
```

OMNI\$INITIALIZE

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

Format

status=OMNI\$INITIALIZE [vendor], [model], [revision]

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$INITIALIZE[A] procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vendor

type: character-coded text string

access: read only

mechanism: by reference

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

model

type: character-coded text string

access: read only

mechanism: see Appendix B

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

Revision

type: character-coded text string

access: read only

mechanism: see Appendix B

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

OMNI\$KILL

OMNI\$KILL — Kills a program on the VMD.

Format

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$KILL procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

Invoke_id is an identifier assigned by VSIomni. The parameter is used for asynchronous calls only.

pi_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded program definition.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

OMNI\$KILL terminates a program invocation by causing it to transition to the unrunnable state.

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, [inc

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$LISTEN procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

called_vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

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.)

translate_flag

type: longword

access: read only

mechanism: see Appendix B

One of the values shown in Table 4.21 to specify the way VSIomni handles indications received from a remote VMD.

Table 4.21. Indications Received

Value	Meaning
0	VSIomni 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, VSIomni returns the initiation indication, creates a dummy VMD definition, and passes the handle of the dummy definition to the user.

calling_vmd_handle

type: OMNI\$L_HANDLE

access: write only

mechanism: see Appendix B

Handle of a VMD specified by the calling application (the remote peer). During the association, the remote peer will accept client requests to operate on this VMD.

incoming_vmd_struct

type: OMNI\$R_VMD_DEF

access: write only

mechanism: see Appendix B

Service parameters proposed by the calling (remote) application.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$LOAD_DEFINITIONS

OMNI\$LOAD_DEFINITIONS — Procedure 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
```

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$LOAD_DEFINITIONS procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_name

type: character-coded text string

access: read only mechanism: see Appendix B

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

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

A location to receive the handle of the VMD definition.

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)

Returns

VMS Usage: cond_value

type: longword

access: write only

mechanism: by value in R0

Description

OMNI\$MODIFY_DEFINITION modifies the values of both single-valued and multi-valued attributes. In the case of multi-valued 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 VSIomni.

Arguments

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

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.

contextaddress

type: OMNI\$L_CONTEXT

access: modify

mechanism: by reference

Contextaddress is the address of a variable. This parameter is used only if you retrieve the value of a multi-valued attribute.

The value of contextaddress should be initialized to NULL. In general, the value of the context should be modified when as many values as required have been retrieved from a multi-valued attribute. In this case, OMNI\$END_LIST should be called to free any unneeded space allocated by VSImni. The value of the context should be then reset to zero.

value

type: depends on attribute

access: read only

mechanism: by reference

Value is the address of a buffer that contains the new attribute value.

Usage Notes

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.

The attributes supported for each definition class, the expected type of attribute value, and the default value are listed in the following tables.

1. Table 4.22 shows VMD values.

Table 4.22. VMD Attributes, Expected Data Types, and Defaults

VMD Attributes	Expected Data Type	Defaults
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_VMD_APPL_SIMPLE_NAME	OMNI\$T_AE_TITLE	Name of VMD
OMNI\$K_ATTR_VMD_MAX_SEGMENT	Longword	512
OMNI\$K_ATTR_VMD_MAX_SRV_CALLING	Longword	5
OMNI\$K_ATTR_VMD_MAX_SRV_CALLED	Longword	5
OMNI\$K_ATTR_VMD_NESTING	Word	10
OMNI\$K_ATTR_VMD_PARAM_SUPPORTED	OMNI\$B_PARAM_CBB	See below
OMNI\$K_ATTR_VMD_SRV_SUPPORTED	OMNI\$B_SVC_CBB	See below
OMNI\$K_ATTR_VMD_VERSION	OMNI\$L_ENUMERATION_CONST	
OMNI\$K_ATTR_VENDOR	OMNI\$T_MMS_ID	""
OMNI\$K_ATTR_MODEL	OMNI\$T_MMS_ID	""
OMNI\$K_ATTR_REVISION	OMNI\$T_MMS_ID	""
OMNI\$K_ATTR_SCOPE	NULL	NONE
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	""
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute of the VMD definition, the value of the Name attribute must already be specified.
- The value of the Version attribute must be one (1) (MMS_IS) or zero (0) (MMS_DIS).
- The scope of a VMD Definition should be specified as NULL.
- The following are defaults for OMNI\$K_ATTR_VMD_PARAM_SUPPORTED and OMNI\$K_ATTR_SRV_SUPPORTED. Default for OMNI\$K_ATTR_VMD_PARAM_SUPPORTED:

```

STR1 (Arrays) TRUE
STR2 (Structures) TRUE
VNAM (Named Variables) TRUE
VALT (Alternate Access) TRUE
VADR (Unnamed Variables) TRUE
VSCA (Scattered Access) FALSE
TPY (Third Party) FALSE
VLIS (Named Variable List) FALSE

```

```
REAL (Real) FALSE
AKEC (Acknowledgement Event Condtions) FALSE
CEI (Evaluation Interval) FALSE
Default for OMNI$K_ATTR_SRV_SUPPORTED
/* vmd support services */
Status FALSE
GetNameList TRUE
Identify TRUE
Rename FALSE
/* variable access services */
Read TRUE
Write TRUE
GetVariableAccessAttributes TRUE
DefineNamedVariable FALSE
DefineScatteredAccess FALSE
GetScatteredAccessAttributes FALSE
DeleteVariableAccess FALSE
DefineNamedVariableList FALSE
GetNamedVariableListAttributes FALSE
DeleteNamedVariableList FALSE
DefineNamedType FALSE
GetNamedTypeAttributes FALSE
DeleteNamedType FALSE
/* operator communication services */
Input FALSE
Output FALSE
/* semaphore management services */
TakeControl FALSE
RelinquishControl FALSE
DefineSemaphore FALSE
DeleteSemaphore FALSE
ReportSemaphoreStatus FALSE
ReportPoolSemaphoreStatus FALSE
ReportSemaphoreEntryStatus FALSE
/* domain management services */
InitiateDownloadSequence TRUE
DownloadSegment TRUE
TerminateDownloadSequence TRUE
InitiateUploadSequence TRUE
UploadSegment TRUE
TerminateUploadSequence TRUE
RequestDomainDownload TRUE
RequestDomainUpload TRUE
LoadDomainContent TRUE
StoreDomainContent TRUE
DeleteDomain TRUE
GetDomainAttributes TRUE
/* program invocation management services */
CreateProgramInvocation FALSE
DeleteProgramInvocation FALSE
Start FALSE
Stop FALSE
Resume FALSE
Reset FALSE
Kill FALSE
GetProgramInvocationAttributes FALSE
/* file management services */
ObtainFile TRUE
```

```
/* event management services */
DefineEventCondition FALSE
DeleteEventCondition FALSE
GetEventConditionAttributes FALSE
ReportEventConditionStatus FALSE
AlterEventConditionmonitoring FALSE
TriggerEvent FALSE
DefineEventAction FALSE
DeleteEventAction FALSE
GetEventActionAttributes FALSE
ReportEventActionStatus FALSE
DefineEventEnrollment FALSE
DeleteEventEnrollment FALSE
AlterEventEnrollment FALSE
ReportEventEnrollmentStatus FALSE
GetEventEnrollmentAttributes FALSE
AcknowledgeEventNotification FALSE
GetAlarmSummary FALSE
GetAlarmEnrollmentSummary FALSE
/* journal management services */
ReadJournal FALSE
WriteJournal FALSE
InitializeJournal FALSE
ReportJournalStatus FALSE
CreateJournal FALSE
DeleteJournal FALSE
/* vmd support services */
GetCapabilityList FALSE
/* file management services */
FileOpen TRUE
FileRead TRUE
FileClose TRUE
FileRename TRUE
FileDelete TRUE
FileDirectory TRUE
/* unconfirmed services */
UnsolicitedStatus FALSE
InformationReport FALSE
EventNotification FALSE
AttachToEventCondition FALSE
AttachToSemaphore FALSE
/* additional services */
Conclude TRUE
Cancel FALSE
```

2. Table 4.23 shows Domain values.

Table 4.23. Domain Attributes, Expected Data Type, and Defaults

Domain Constants	Data Type	Defaults
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_CAPABILITY_FILE	OMNI\$T_FILENAME	OMNI\$DOMAINS: [VMD]domain.cap where VMD is the name of the parent VMD and

Domain Constants	Data Type	Defaults
		domain is the domain name
OMNI\$K_ATTR_DOM_CONTENT_FILE	OMNI\$T_FILENAME	""
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN	True
OMNI\$K_ATTR_SHARABLE	OMNI\$B_BOOLEAN	False
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	""
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute, the Name of the Domain definition must already be specified.
- The run-time object definition facility does not restrict the sharing of a domain that has been specified as not sharable.

The value of the scope attribute must be a VMD handle.

3. Table 4.24 shows PI values.

Table 4.24. PI Attributes, Expected Data Types, and Defaults

PI Constants	Data Types	Defaults
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_REF_DOM_NAMES	OMNI\$L_HANDLE	""
OMNI\$K_ATTR_EXEC_ARG	OMNI\$R_EXEC_ARG_STR	""
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN	True
OMNI\$K_ATTR_REUSABLE	OMNI\$B_BOOLEAN	False
OMNI\$K_ATTR_MONITOR	OMNI\$L_ENUMERATION_CONST	OMNI\$K_MNT_NOT_PRESENT
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	None
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute, the Name of the PI and at least one Referenced Domain must already be specified.
- The value of the scope attribute must be a VMD handle.
- The values of the Referenced Domains attribute must be the handles of committed domains.

- The Domains referenced by the Program Invocation must all have the same VMD as a parent. Additionally, the Program Invocation must have the same parent as the Domains it references.
- A Program Invocation definition may not reference more than 20 Domains.

4. Table 4.25 shows Named Variable values.

Table 4.25. Named Variable Attributes, Expected Data Types, and Defaults

Named Variable	Data Types	Defaults
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN	True
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	""
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute, the values of the Name and the Application Type Description attributes must already be specified.
- The value of the scope attribute of a named variable definition must be the handle of a domain or a VMD
- If the value of the Application Type Description Attribute is an Application Named Type handle, then the Application Named Type definition must be committed. Additionally, the Named Variable and the Application Named Type definitions must reside on the same VMD definition.
- If the value of the Application Type Description attribute is an Application Type Specification handle, then the Application Type Specification may not already be referenced by any other definition.

5. Table 4.26 shows Unnamed Variable values.

Table 4.26. Unnamed Variable Attributes, Expected Data Types, and Defaults

Unnamed Variables	Data Types	Defaults
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_ADDRESS_TYPE	OMNI \$L_ENUMERATION_CONST	None
OMNI\$K_ATTR_ADDRESS_STRING	OMNI\$R_ADDRESS_STR	None
OMNI\$K_ATTR_ADDRESS_NUMBER	Longword	None
OMNI\$K_ATTR_SUPPLY_TYPE_SPEC	OMNI\$B_BOOLEAN	True
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DESCRIPTION	MNI\$T_DESCRIPTION	""
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute, the values of the Name, Address Type and either Address String or Address Number, and Application Type Description attributes must be already specified.
- The value of the scope attribute of an Unnamed Variable definition must be the handle of a VMD.
- If the value of the Application Type Description Attribute is an Application Named Type handle, then the Application Named Type must be committed. Additionally, the Named Variable and the Application Named Type definitions must reside on the same VMD definition.
- If the value of the Application Type Description attribute is an Application Type Specification handle, then the Application Type Specification may not already be referenced by any other definition.
- The value of the Address Type attribute must be modified before the value of either the Address String or Address Number attribute.
- VSIomni allows the user to modify the value of the Address Type attribute even after the value of the Address String or Address Number attribute has been modified, VSIomni does not compare the value of the Address Type attribute with the type of the address.

6. Table 4.27 shows MMS Named Type values.

Table 4.27. MMS Named Type Attributes, Expected Data Types, and Defaults

MMS Named Types	Data Types	Defaults
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DELETABLE	OMNI\$B_BOOLEAN	True
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	""
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute, the values of the Name and MMS Type Description Attributes must already be specified.
- The value of the scope attribute of an MMS Named Type definition must be the handle of a domain or a VMD definition.
- If the value of the MMS Type Description Attribute is an MMS Named Type handle, then the referred to MMS Named Type must already be committed. Additionally, both MMS Named Type definitions must ultimately reside on the same VMD definition.
- If the value of the MMS Type Description Attribute is an MMS Type Specification handle, then the MMS Type Specification may not already be referenced by any other definition. When the MMS Named Type definition is ready to be committed, its parent VMD must be the same as the

parent VMDs of any MMS Named Type definitions referred to by the MMS Type Specification definition.

7. Table 4.28 shows Application Named Type values.

Table 4.28. Application Named Type Attributes, Expected Data Types, and Defaults

Application Named Types	Data Types	Defaults
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_MMS_NAMED_TYPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION_WC	""
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute, the values of the Name, MMS Named Type, and Application Type Description Attributes must already be specified.
- The value of the scope attribute of an Application Named Type Definition must be the handle of a domain or a VMD definition.
- If the value of the Application Type Description Attribute is an Application Named Type Handle, then the Application Named Type definition must already be committed. Additionally, the Application Named Type Definitions must ultimately reside on the same VMD definition.
- If the value of the Application Type Description Attribute is an Application Type Specification Handle, then the Application Type Specification definition cannot already be referenced by any other definition. Additionally, the Application Named Type definition must reside on the same vmd as any Application Named Type definition referred to by the Application Type Specification definition.
- The value of the MMS Named Type Attribute must be the handle of a committed MMS Named Type Definition. The MMS Named Type Definition and the Application Named Type Definition must ultimately reside on the same VMD Definition.

8. Table 4.29 shows MMS Type Specification values.

Table 4.29. MMS Type Specification Attributes, Expected Data Types, and Defaults

MMS Type	Data Type	Default
OMNI\$K_ATTR_MMS_TYPE	MNI\$L_ENUMERATION_CONST	None
OMNI\$K_ATTR_ARRAY_PACKED	OMNI\$B_BOOLEAN	False
OMNI\$K_ATTR_ARRAY_ELEM_NUM	Longword	0
OMNI\$K_ATTR_ARRAY_ELEM_MMS_TYPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_STRUCT_PACKED	OMNI\$B_BOOLEAN	False
OMNI\$K_ATTR_SIMPLE_TYPE_SIZE	Longword	0
OMNI\$K_ATTR_SIMPLE_TYPE_VARYING	OMNI\$B_BOOLEAN	False

MMS Type	Data Type	Default
OMNI\$K_ATTR_DATE_INCLUDED	OMNI\$B_BOOLEAN	False
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	None
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- The value of the MMS Type Attribute must be modified before other attributes may be modified. The MMS Type Attribute may be modified once.
- An MMS Type Specification is valid when the following is true:
 - The value of the MMS Type Attribute has been specified
 - If the value of the MMS Type Attribute is OMNI\$K_TKM_ARRAY then the value of OMNI\$K_ATTR_ARRAY_ELEM_NUM attribute has been specified,
 or
 - If the value of the MMS Type Attribute is OMNI\$K_TKM_STRUCTURE then at least one structure component has been specified for the type specification.
- The MMS Type Specification must be valid before it can be referenced by another definition. Additionally, an MMS Type Specification may only be referenced once by either another MMS Type Specification definition or by an MMS Named Type Definition.
- The value of the MMS Type Attribute is a constant from the OMNI\$K_TKM_ * enumeration in OMNI\$DEFS. The following constants are acceptable values:
 - OMNI\$K_TKM_ARRAY
 - OMNI\$K_TKM_STRUCTURE
 - OMNI\$K_TKM_BOOLEAN
 - OMNI\$K_TKM_BIT_STR
 - OMNI\$K_TKM_INTEGER
 - OMNI\$K_TKM_UNSIGNED
 - OMNI\$K_TKM_FLOATING_POINT
 - OMNI\$K_TKM_OCTET_STR
 - OMNI\$K_TKM_VISIBLE_STR
 - OMNI\$K_TKM_GENERALIZED_TIME
 - OMNI\$K_TKM_BINARY_TIME
 - OMNI\$K_TKM_BCD
 - OMNI\$K_TKM_OBJECT_ID

9. Table 4.30 shows Application Type Specification values.

Table 4.30. Application Type Specification Attributes, Expected Data Types, and Defaults

Application Type	Data Type	Default
OMNISK_ATTR_APP_TYPE	OMNISL_ENUMERATION_CONST	None
OMNISK_ATTR_MMS_TYPE_SPEC	OMNISL_HANDLE	None
OMNISK_ATTR_ARRAY_ELEM_NUM	Longword	None
OMNISK_ATTR_ARRAY_ELEM_APP_TYPE	OMNISL_HANDLE	None
OMNISK_ATTR_ARRAY_LOW	Longword	None
OMNISK_ATTR_SIMPLE_TYPE_SIZE	Longword	None
OMNISK_ATTR_DESCRIPTION	OMNIST_DESCRIPTION	None
OMNISK_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- The value of the Application Type Attribute must be modified before other attributes may be modified. The Application Type Attribute may be modified once.
- An Application Type Specification definition is valid when:
 - The value of the Application Type Attribute has been specified
 - If the value of the Application Type Attribute is OMNISK_TKA_ARRAY then the values of the OMNISK_ATTR_ARRAY_ELEM_APP_TYPE attribute and the OMNISK_ATTR_ARRAY_ELEM_NUM attribute have been specified,

or

 - If the value of the Application Type Attribute is OMNISK_TKA_STRUCTURE then at least one structure component has been specified for the type specification definition
 - The value of the MMS Type Specification Attribute has been specified
 - The Application Type Specification is not already referenced by another definition . The Application Type Specification definition must be valid before it can be referenced by another definition. Additionally, an Application Type Specification may only be referenced by one definition.
- The value of the App Type Attribute is a constant from the OMNISK_TKA_ * enumeration. The following constants are acceptable values:
 - OMNISK_TKA_ARRAY
 - OMNISK_TKA_STRUCTURE
 - OMNISK_TKA_BOOLEAN
 - OMNISK_TKA_BIT_STR

- OMNI\$K_TKA_INTEGER
- OMNI\$K_TKA_UNSIGNED
- OMNI\$K_TKA_F_FLOAT
- OMNI\$K_TERMINATED_STR
- OMNI\$K_WORD_COUNTED_STR
- OMNI\$K_SCALAR_STR
- OMNI\$K_OMNI_TIME
- OMNI\$K_VMS_ABSOLUTE_TIME
- OMNI\$K_TKA_BOOLEAN_ARRAY

10. Table 4.31 shows MMS Structure Component values.

Table 4.31. MMS Structure Component Attributes, Expected Data Types, and Defaults

MMS Structure Component	Data Type	Default
OMNI\$K_ATTR_NAME	OMNI\$T_MMS_ID	None
OMNI\$K_ATTR_MMS_TYPE_DESCRIPTION	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	None
OMNI\$K_ATTR_USER_PARAM	Longword	None
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None

Requirements and restrictions are:

- In order to modify the value of the scope attribute, the values of the Name and MMS Type Description attributes must already be specified.
- The value of the scope attribute of an MMS Structure Component Definition is the handle of an MMS Type Specification Definition whose MMS Type Attribute has a value of OMNI\$K_TKM_STRUCTURE.
- If the value of the MMS Type Description Attribute is an MMS Named Type handle, then the MMS Named Type definition must be committed. Additionally, the MMS Named Type definition and the MMS structure component must ultimately reside on the same vmd definition.
- If the value of the MMS Type Description attribute is an MMS Type Specification Handle, then the MMS Type Specification may not already be referenced by any other definition.

11. Table 4.32 shows App Structure Component values.

Table 4.32. App Structure Component Attributes, Values, and Defaults

App Structure Component	Value	Default
OMNI\$K_ATTR_NAME	OMNI\$T_APP_ID	None

App Structure Component	Value	Default
OMNI\$K_ATTR_DESCRIPTION	OMNI\$T_DESCRIPTION	None
OMNI\$K_ATTR_APP_TYPE_DESCRIPTION	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_MMS_STRUCT_COMP	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_SCOPE	OMNI\$L_HANDLE	None
OMNI\$K_ATTR_USER_PARAM	Longword	None

Requirements and restrictions are as follows:

- In order to modify the value of the scope attribute, the values of the Name, App Type Description, and MMS Type Specification Attributes must already be specified.
- The value of the scope attribute of an Application Structure Component Definition is the handle of an Application Type Specification Definition whose Application Type Attribute has a value of OMNI\$K_TKA_STRUCTURE.
- If the value of the Application Type Description Attribute is an Application Named Type Handle, then the Application Named Type definition must be committed. Additionally, the Application Named Type Definition and the Application Structure Component definition must reside on the same VMD definition.
- If the value of the Application Type Description Attribute is an Application Type Specification Handle, then the Application Type Specification definition cannot already be referenced by any other definition

OMNI\$PUT_VALUE

OMNI\$PUT_VALUE — 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_`

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$PUT_VALUE procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

`invoke_id`

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSIomni. This parameter is used for asynchronous calls only.

object_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

In a client call, the identifier of a loaded variable definition.

In a server call, the context value returned by the GET\$INDICATIONS procedure call that delivered the write indication.

method_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Method_handle modifies the default presentation of a variable.

value_structure

type: depends on value

access: read only

mechanism: see Appendix B

A data structure containing the value of the object.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$REJECT

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

Format

status=OMNI\$REJECT[_A] context,[reason], [omni_iosb], [ctrl_struct]

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$REJECT procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

Arguments

context

type: OMNI\$R_CONTEXT

access: read only

mechanism: see Appendix B

Information returned by VSIomni.

reason

type: condition value

access: read only

mechanism: see Appendix B

The reason the indication is being rejected.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$REJECT_CONCLUDE

OMNI\$REJECT_CONCLUDE — Rejects an association conclude request from a remote application.

Format

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$REJECT_CONCLUDE procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

vmd_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

A location to receive the handle of the VMD definition.

reason

type: condition value

access: read only

mechanism: see Appendix B

The reason the indication is being rejected.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

Call the OMNI\$REJECT_CONCLUDE procedure in response to an indication returned by OMNI\$GET_INDICATIONS.

OMNI\$RESET

OMNI\$RESET — Resumes execution of a stopped program on the VMD.

Format

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$RESTART procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImni. This parameter is used for asynchronous calls only.

pi_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded program definition.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSImni I/O status block. For a description of the codes that VSImni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

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 will transition to the idle state; otherwise, it will transition to the unrunnable state.

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_ob]

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$RESUME procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSIomni. This parameter is used for asynchronous calls only.

pi_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded program definition.

execution_argument

type: OMNI\$T_EXEC_ARG_STR

access: read only

mechanism: see Appendix B

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

Note

Externally encoded execution arguments are not supported for V1.0.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

OMNI\$START

OMNI\$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_obje

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$START procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImni. This parameter is used for asynchronous calls only.

pi_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded program definition.

execution_argument

type: OMNI\$T_EXEC_ARG_STR

access: read only

mechanism: see Appendix B

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

Note

Externally encoded execution arguments are not supported for V1.0.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

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_struct]

Returns

VMS Usage: `cond_value`

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$STOP procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSIomni. This parameter is used for asynchronous calls only.

pi_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded program definition.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSIomni I/O status block. For a description of the codes that VSIomni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

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

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$OMNI_TO_VMS_TIME procedure returns parameter-checking codes and completion codes to R0. For a list of these codes, see Appendix A

Arguments

OMNI_Time

type: OMNI\$L_LAST_MODIFIED

access: read only

mechanism: by reference

Time returned by OMNI\$FGET.

VMS_Time

type: OMNI\$L_LAST_MODIFIED

access: read only

mechanism: by reference

Location of the VMS quadword.

OMNI\$UPLOAD

OMNI\$UPLOAD — Performs the uploading of a domain from a remote VMD.

Format

status=OMNI\$UPLOAD[_A] [invoke_id], domain_handle, [domain_file], [modifier_c]

Returns

VMS Usage: cond_value

type: longword (unsigned)

access: write only

mechanism: by value in R0

The OMNI\$UPLOAD procedure returns parameter- checking codes and completion codes to R0. For a list of these codes, see Appendix A

The *status* variable is a longword to contain this value.

Arguments

invoke_id

type: longword

access: write only

mechanism: see Appendix B

An identifier assigned by VSImni. This parameter is used for asynchronous calls only.

domain_handle

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

The identifier of a loaded domain definition.

domain_file

type: OMNI\$T_FILE_NAME

access: read only

mechanism: see Appendix B

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.

modifier_object

type: OMNI\$L_HANDLE

access: read only

mechanism: see Appendix B

Reserved for future use.

omni_iosb

type: OMNI\$R_IOSB

access: write only

mechanism: by reference

The VSImni I/O status block. For a description of the codes that VSImni returns to the IOSB, see Appendix A

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

ctrl_struct

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_struct* parameter is the address of the control structure.

Usage Notes

1. 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.
2. If an error occurs during the upload process, the files will be deleted.

Chapter 5. VSiomni MMSI

VSiomni MMSI is an implementation of the programming interface to MMS services defined in the MAP 3.0 MMS Application Interface Specification.

The MMS Application Interface Functions implemented for MMSI are fully described in the following documents:

- MAP_Attachment 1 to MAP_Appendix 7, Application Interface Model and Specification Requirements
- MAP_Attachment 2 to MAP_Appendix 7, Application Interface Support Functions
- MAP_Attachment 6 to MAP_Appendix 7, MMS Application Interface Specification
- 1989 MAP Errata #1

Attachment 2 describes the functions `mm_dfdbc`, `mm_didcb`, the Errata describes `mm_dfmemory` and Attachment 6 describes the other functions implemented for MMSI.

5.1. Overview of the MMSI Implementation

Only the following context-sensitive functions have been implemented for the first release of MMSI:

- `mm_identify` (Identify)
- `mm_status` (Status)
- `mm_gdattribute` (Get Domain Attribute)
- `mm_gpinvocation` (Get Program Invocation)

Only the following context-free functions have been implemented for the first release of MMSI:

- `mm_didcb` (Dynamic Initialize DCB)
- `mm_dfdbc` (Dynamic Free DCB)
- `mm_leextract` (List Element Extract)
- `mm_dfmemory` (Free Dynamic Memory)
- `mm_fhandle` (Free Handle)

All function call interfaces are implemented as documented in the MMS Application Interface Specification of MAP except for the following deviations and exclusions:

- Context sensitive functions:

Asynchronous requests are not supported; therefore, the only valid value for the input parameter *return_event_name* is SYNCHRONOUS.

Since none of the connection management functions are supported in this release, the input parameter *connection_id* is to contain a valid VSiomni VMD handle value of a VMD for which VMD definitions have been initialized and loaded through the VSiomni functions OMNI\$INITIALIZE and OMNI\$LOAD_DEFINITIONS and a connection has been established through the VSiomni function OMNI\$CONNECT.

- `mm_didcb` and `mm_dfcb`

Only the following data control block types are supported:

- `MMiGpinvocation`
 - `MMoGpinvocation`
 - `MMiGdattribute`
 - `MMoGdattribute`
 - `MMiIdentify`
 - `MMoIdentify`
 - `MMiStatus`
 - `MMoStatus`
- `mm_status`

The input parameter `extended_derivation` is ignored.

5.2. Requirements for Using MMSI

VSImni MMSI has the following requirements:

- VSImni must be installed.
- OMNI Definition Facility (ODF) must be used to create local definitions of all MMS objects that will be accessed.

5.3. Compiling and Linking User Programs with MMSI

OMNI\$MMSI_DEFS.H contains C-Bindings from the MMS Application Interface Specification that are needed to correctly compile C language modules which call MMSI functions.

Note that this file contains only those C-Bindings which are relevant to the functions implemented for MMSI and excludes error and return code values.

To use this file, the statement `"#include <omni$mmsi_defs.h>"` is put in the user source code modules.

OMNI\$MMSI_CODES.H contains constant definition statements for all return codes and error codes defined in the MMS Application Interface Specification.

To use this file, the statement `"#include <omni$mmsi_codes.h>"` is put in the user source code modules.

The VMS MESSAGE facility can be used to obtain string descriptions associated with these return codes.

Applications containing MMSI function calls must link against the shareable image `SY$SHARE:OMNI$VAXCSHR.EXE`.

5.4. Example Code

The following is an example of MMSI code.

```
/*
**++
** FACILITY: VSImni/MMSI
**
** MODULE DESCRIPTION:
**
** This module contains coding examples of the following MMSI functions:
**
** mm_identify - Identify
** mm_dfdcb - Dynamic Free Data Control Block
**
**
** CREATION DATE: 27-APR-1990
**
**__
*/
/*
**
** INCLUDE FILES
**
**
*/
#include descrip
#include <omni$codes.h>
#include <omni$defs.h>
#include <omni$mmsi_defs.h>
/*
**
** MACRO DEFINITIONS
**
**
*/
#define TRUE 1
#define NULL 0
#define ODD(x) ( x & 1)
#define SetDesc(x,y) { x.dsc$b_dtype=DSC$K_DTYPE_T; \
    x.dsc$b_class=DSC$K_CLASS_S;\
    x.dsc$w_length=sizeof(y); x.dsc$a_pointer= &y;}
static void DisplayResult(status, err_blk)
    int status;
    Mm_Error_type *err_blk;
{
    int msglen=0, flags= 0xF;
    char msgbuf[256];
    struct dsc$descriptor_s str_desc;
    if (!(status)) printf(" SUCCESS \n");
    else
    {
        SetDesc( str_desc, msgbuf)
        lib$sys_getmsg( &status, &msglen, &str_desc, &flags);
        msgbuf[msglen]='\0';
        printf(" %s \n", msgbuf);
        if (err_blk != NULL)
        {
            if (err_blk->result.vendor_code)
```

```

        {
            lib$sys_getmsg( &err_blk->result.vendor_code, &msglen, &str_desc,
&flags);
            msgbuf[msglen]='\0';
            printf(" Vendor code: %s \n", msgbuf);
        }
        if (err_blk->mask == MM_ET_ERROR_CODE)
            printf(" Error code : %d \n", err_blk->error_code);
    }
}

static void ShowIdentifyDCB(inout_dcb)
    Mm_Identify_out_dcb *inout_dcb;
{
    char address_NULL[] = {" address = NULL\n"};
    char format_str[] = {" : \"%s\"\n"};
    printf("\n IDENTIFY attributes returned\n");
    printf(" vendor name");
    if (inout_dcb->vendor_name != NULL) printf( format_str,
inout_dcb->vendor_name);
    else printf( address_NULL);
    printf(" model name ");
    if (inout_dcb->model_name != NULL) printf( format_str,
inout_dcb->model_name);
    else printf( address_NULL);
    printf(" revision ");
    if (inout_dcb->revision != NULL) printf( format_str, inout_dcb-
>revision);
    else printf( address_NULL);
}

main()
{
    Return_code ret_code;
    Api_rc result;
    Mm_Identify_in_dcb input_dcb= { MM_MMS};
    Mm_Identify_out_dcb *inout_dcb=NULL;
    OMNI$L_HANDLE VmdHandle;
    OMNI$R_IOSB iosb;
    OMNI$T_MMS_ID VmdName='\0';
    int status;
    printf("\nEnter VMD name : ");
    scanf( "%s", &VmdName);
    /*
    ** initialize VSiomni
    */
    printf("\n\nInitializing OMNI ... ");
    DisplayResult(status= OMNI$INITIALIZE( "Digital", "Read-Test", "V1.0"),
NULL);
    if ( status == OMNI$_SUCCESS)
    {
        /*
        ** load VMD definitions
        */
        printf( "\nLoading OMNI definitions ... ");
        DisplayResult(status= OMNI$LOAD_DEFINITIONS ( VmdName, &VmdHandle),
NULL);
        if ( status == OMNI$_SUCCESS)
        {

```

```
/*
** connect to VMD
*/
printf( "\nEstablishing connection with VMD ... ");
DisplayResult(status= OMNI$CONNECT ( VmdHandle,0,0,0,0,&iosb,0),
NULL);
if ( status == OMNI$_SUCCESS)
    if ( iosb.OMNI$L_IOSB_GENERAL != OMNI$_SUCCESS)
        DisplayResult( iosb.OMNI$L_IOSB_GENERAL, NULL);
    else
    {
/*
** get VMD identification
*/
printf( "\nCalling mm_identify ... ");
status= mm_identify( VmdHandle, SYNCHRONOUS, &input_dcb,
&inout_dcb);
DisplayResult( status, &inout_dcb->error_block);
if (status == SUCCESS) ShowIdentifyDCB( inout_dcb);
/*
** free dynamically allocated dcb
*/
printf ( "\nDeallocating in/out DCB ... ");
DisplayResult( ret_code= mm_dfdbc( inout_dcb, &result), NULL);
}
}

/*
** terminate VMD connection
*/
printf( "\nConcluding connection with VMD ... ");
DisplayResult( status= OMNI$CONCLUDE( VmdHandle, &iosb, 0 ), NULL);
if ( status == OMNI$_SUCCESS)
if ( iosb.OMNI$L_IOSB_GENERAL != OMNI$_SUCCESS)
    DisplayResult( iosb.OMNI$L_IOSB_GENERAL, NULL);
}
}
```


Appendix A. VSIomni Messages

This appendix describes VSIomni status and completion codes.

A.1 Success Codes

Table below describes VSIomni success codes.

Table A.1. VSIomni Codes (Severity: Success)

Code	Meaning
SQLSUCCESS	SQL call succeeded
SUCCESS	Successful completion

A.2 Informational Codes

Table below describes VSIomni informational codes.

Table A.2. VSIomni Message Codes (Severity: Informational)

Code	Meaning
ABORT_IND	Association aborted by peer
ABORT_REQ	Association aborted by user
AS_ACCEPTED	Abstract syntax accepted
AS_REJECTED	Abstract syntax rejected
CONCLUDE	Association concluded
CONCLUDE_REJECTED	Conclude rejected
ENDOFLIST	No more elements-nothing was returned
EXIT FSM:	Action exit
INITIATE	Initiate completed
INITIATE_REJECT	Initiate rejected
ONEENTQUE	One entry in queue
PENDING FSM:	Action is to continue later

A.3 Warning Codes

Table below describes VSIomni warning codes.

Table A.3. VSIomni Message Codes (Severity: Warning)

Code	Meaning
QUEWASEMP	Queue was empty
SQLCURNOTOPE	SQL cursor is not open
SQLENDOFLIST	SQL cursor has no more elements
STRTRN	String was truncated
UDFAPTITLE	The VMD AP Title is not defined

Code	Meaning
UNRMAPPREF	Unresolved Mapping Reference
UNRTYPREF	Unresolved Type Reference
UNRVADREF	Unresolved Unnamed Variable Reference
UNRVLSREF	Unresolved Variable List Reference
UNRVNMREF	Unresolved Named Variable Reference

A.4 Error Codes

Table below describes VSImni error codes.

Table A.4. VSImni Message Codes (Severity: Error)

Code	Meaning
ABORT_ERR	Service Error Class-Vmd-State : Other
ABORT_IND_ERR	Abort Indication Error
ACSEAbortFail	ACSE Abort Failed
ACSEGetIndFail	ACSE Get Indication Failed
ACSEReleaseFail	ACSE Release Failed
ACSE_Abort	Get Indication Received an ACSE Abort Indication
ACSE_Release	Get Indication Received an ACSE Release Request
ACSE_Unknown	Get Indication Received an Unknown Request
ALRINI	VSImni is already initialized
APP_REFERENCE_INVD	Service Error Class-Applic Ref: Invalid
APP_UNREACHABLE	Service Error Class-Applic Ref: Unreachable
ASMisMtc	Abstract Syntax Mismatch
ATCREFATS	App Type Component refers to nonexistent App Type Specification
ATCREFMTC	App Type Component refers to nonexistent MMS Type Component
ATNODOM	App Named Type depends on a nonexistent domain
ATNOMT	App Named Type depends on nonexistent MMS Named Type
ATNOVMD	App Named Type depends on a nonexistent VMD
ATSREFMTS	App Type Specification refers to nonexistent MMS Type Specification
ATTRRIB	Attribute Set constant is out of range
BADATTR	Attribute Set constant is out of range or unreadable
BADCLASS	Class constant is out of range or unreadable
BADHANDLE	The handle parameter is invalid
BADNAME	The name parameter is not readable, or too large
BADPARAM	A parameter is not readable/writeable, or too large
BADSCOPE	The scope parameter is invalid
BADVMDNAME	The VMD Name is not readable, or too large
BStrAlign	Invalid Bit alignment in BitString

Code	Meaning
Bad_Parm	One of the specified parameters is invalid
CANCEL	Service Error Class-Service Preempt: Cancel
CANCEL_NOT_POSSIBLE	Service Error Class-Cancel: Cancel Not Possible
CAPABILITY_UNAVAIL	Service Error Class-Resource: Capability Unavailable
CAPABILITY_UNKNOWN	Service Error Class-Resource: Capability Unknown
CONCLUDE_CNF_FINISH_ERR	Conclude Confirm Finish Error
CONCLUDE_ERR	Conclude Error
CONCLUDE_ERR_IND_ERR	Conclude Error Indication Error
CONCLUDE_RSP_ERR	Conclude Response Error
CONCLUDE_RSP_FINISH_ERR	Conclude Response Finish Error
CONNECTION_LOST	Service Error Class-Applic Ref: Connection Lost
CONNECT_CNF_ERR	Connect Confirm Error
CONNECT_ERR	Connect Error
CONNECT_ERR_IND_ERR	Connect Error Indication Error
CONTENT_TYPE_INVD	Service Error Class-File: Content Type Invalid
CONTEXT_UNSPRTD	Service Error Class-Applic Ref: Context Unsupported
CONTINUATION_INVD	Service Error Class-Service: Continuation Invalid
CloseFile_CNF_Err	Close File Confirmation Service Error
CloseFile_IND_Err	Close File Indication Service Error
CloseFile_REQ_Err	Close File Request Service Error
CloseFile_RSP_Err	Close File Response Service Error
CnclErrFail	Cancel Error Service Failed
CnclListenFail	Cancel Listen Request Failed
CnclRqsFail	Cancel Request Service Failed
CnclRspFail	Cancel Response Service Failed
CncldErrFail	Conclude Error Service Failed
CncldRqsFail	Conclude Request Service Failed
CncldRspFail	Conclude Response Service Failed
CnfErrFail	Confirmed Error Failed
CnfRqsFail	Confirmed Request Failed
CnfRspFail	Confirmed Response Failed
CnsMisMtc	Constructor Mismatch
CreatePI_CNF_Err	Create Program Invocation Confirmation Service Error
CreatePI_REQ_Err	Create Program Invocation Request Service Error
DEADLOCK	Service Error Class-Service Preempt: Deadlock
DECODING_ERR	Decoding Error
DEFNOTFND	The requested definition was not found
DOMNOVMD	Domain depends on a VMD which does not exist

Code	Meaning
DOM_XFER_PROB	Service Error Class- Vmd-State : Domain Transfer Problem
DS_Error	Directory Service Error
DT_Gen_Error	Data Transfer Component Failure, please submit an SPR
DataOnNULL	Decoder Found Data with NULL Type
DbgErr	Debugging Error
DelDom_CNF_Err	Delete Domain Confirm Service Error
DelDom_REQ_Err	Delete Domain Request Service Error
DelPI_CNF_Err	Delete Program Invocation Confirmation Service Error
DelPI_REQ_Err	Delete Program Invocation Request Service Error
DeleteFile_CNF_Err	Delete File Confirmation Service Error
DeleteFile_IND_Err	Delete File Indication Service Error
DeleteFile_REQ_Err	Delete File Request Service Error
DeleteFile_RSP_Err	Delete File Response Service Error
DirectoryFile_CNF_Err	Directory File Confirmation Service Error
DirectoryFile_IND_Err	Directory File Indication Service Error
DirectoryFile_REQ_Err	Directory File Request Service Error
DirectoryFile_RSP_Err	Directory File Response Service Error
DomAbort_Err	Domain Abort Indication Processing Failed
DomDiscarded	Download Domain Discarded
DwlSeg_IND_Err	Download Segment Indication Service Error
DwlSeg_RSP_Err	Download Segment Response Service Error
END_LIST_ERR	Error Occured In End List Request
EleNotCns	Element not of Constructed Type
ExpBStr	Decoder Expected BitString
ExpBool	Decoder Expected Boolean not found
ExpChcSel	Decoder Expected Choice Selector not found
ExpCnst	Decoder Expected Constructor not found
ExpEOC	Decoder Expected End of Constructor not found
ExpExt	Decoder Expected External not found
ExpInt	Decoder Expected Integer Primitive
ExpSeqOf	Decoder Expected Sequence Of not found
ExpSeq	Decoder Expected Sequence
ExpStr	Decoder Expected String not found
FILENAME_AMBIGUOUS	Service Error Class- File: Ambiguous Name
FILENAME_SYNTAX_ERR	Service Error Class-File: Syntax Error
FILE_ACCESS_DENIED	Service Error Class-File: File Access Denied
FILE_BUSY	Service Error Class-File: Busy
FILE_NON_EXISTENT	Service Error Class-File: File Non-Existent

Code	Meaning
FREE_ASSOC_CB_ERR	Free Associate Control Block Error
FURTHER_COMM_RQD	Service Error Class-Conclude: Further Comm Rqd
File_Attr_Err	Error Attempting to Access File Attributes
File_Close_Err	Error Attempting to Close a File
File_Del_Err	Error Attempting to Delete a File
File_Open_Err	Error Attempting to Open a File
File_Prot_Err	File Protection Error
File_Read_Err	Error Attempting to Read a File
File_Rename_Err	Error Attempting to Rename the File
File_Write_Err	Error Attempting to Write to a File
GETVAL_CNF_ERR	Get Value Confirm Error
GETVAL_IND_ERR	Get Value Indication Error
GETVAL_REQ_ERR	Get Value Request Error
GETVAL_RSP_ERR	Get Value Response Error
GET_ATTRIBUTE_ERR	Error Occurred In Get Attribute Request
GET_ATTR_CNF_ERR	Get Attribute Confirm Error
GET_ATTR_REQ_ERR	Get Attribute Request Error
GET_CAPABILITY_CNF_ERR	Get Capability List Confirm Error
GET_CAPABILITY_REQ_ERR	Get Capability List Request Error
GET_INDICATIONS_ERR	Get Indications Error
GET_NAME_LIST_CNF_ERR	Get Name List Confirm Error
GET_NAME_LIST_REQ_ERR	Get Name List Request Error
GetDomAttr_CNF_Err	Get Domain Attributes Confirmation Service Error
GetDomAttr_REQ_Err	Get Domain Attributes Request Service Error
GetPIAttr_CNF_Err	Get Program Invocation Attributes Confirmation Service Error
GetPIAttr_REQ_Err	Get Program Invocation Attributes Request Service Error
IDENTIFY_CNF_ERR	Identify Confirm Error
IDENTIFY_IND_ERR	Identify Indication Error
IDENTIFY_REQ_ERR	Identify Request Error
IDENTIFY_RSP_ERR	Identify Response Error
INFO_REPORT_IND_ERR	Info Report Indication Error
INVALMOD	Invalid modifier supplied
INVCLASS	The requested class is not defined within this scope
INVD_ADDRESS	Service Error Class-Definition: Invalid Address
INVOKE_ID_UNKNOWN	Service Error Class-Cancel: Invoke ID Unknown
In_AST	This function may not be called from an AST
InitDwlSeq_CNF_Err	Initiate Download Sequence Confirmation Service Error
InitDwlSeq_REQ_Err	Initiate Download Sequence Request Service Error

Code	Meaning
InitErrFail	Initiate Error Service Failed
InitFail	Initiate Request Failed
InitRspFail	Initiate Response Service Failed
InitUpSeq_CNF_Er	Initiate Upload Sequence Confirmation Service Error
InitUpSeq_REQ_Err	Initiate Upload Sequence Request Service Error
Init_Rejected	Initiate Request Rejected by Remote Partner
InsDTIRP	Insufficient DT IRP Allocation
InvBitCnt	Invalid Bit Count in BitString
InvEOCLen	Decoder Found Invalid End of Constructor length
InvIdentifier	Invalid MMS Identifier Type
InvMemType	Invalid Memory Type
InvProtoVer	Invalid Protocol Version in IRP
InvRequest	Invalid Request Type Specified in IRP
InvTimeOfDay	Invalid Time of Day Type
Inv_File_PDU	Invalid File Service PDU Received
Inv_Init_Position	Invalid Initial Position
Inv_RMS_Format	Invalid RMS Record Format or Attributes
Inv_Time_Format	Invalid Time Format
InvalidFloat	Invalid Float
InvalidObjID	Invalid Object Identifier
Invd_ID	The specified AssocID is invalid
KillPI_CNF_Err	Kill Program Invocation Confirmation Service Error
KillPI_REQ_Err	Kill Program Invocation Request Service Error
LISTEN_ERR	Listen Error
LenTooLrg	Length Field Too Large
ListenFail	Listen Request Failed
Local_Listen_Err_Ind	Received a listen ind error which originated locally
MASS_STORAGE_UNAVAIL	Service Error Class-Resource: Mass Storage Unavailable
MAX_SEGMENT_INSUFF	Service Error Class-Initiate: Max Seg Insufficient
MAX_SVCS_OUT_DEST_INSUFF	Service error Class-Initiate: Max Svcs Out Called Insufficient
MAX_SVCS_OUT_SRC_INSUFF	Service Error Class-Initiate: Max Svcs Out Calling Insufficient
MEMORY_UNAVAIL	Service Error Class-Resource: Memory Unavailable
MTCREFMTS	MMS Type Component refers to nonexistent MMS Type Specification
MTNODOM	MMS Named Type depends on a domain which does not exist
MTNOVMD	MMS Named Type depends on VMD which does not exist
Miss_Handle	The specified handle does not match object type
MsgStruc	Invalid Message Structure

Code	Meaning
MsgTooCmplx	Encoded Message Too Complex
NESTING_LEVEL_INSUFF	Service Error Class- Initiate: Nesting Level Insufficient
NOCONN	There is no connection with the remote device
NOTINI	VSImni has not been initialized
NO_AE_TITLE	No Local Ae Title Defined
NVNODOM	Named Variable depends on a domain which does not exist
NVNOVMD	Named Variable depends on a VMD which does not exist
NVREFAT	Named Variable refers to a nonexistent Application Named Type
NoPriv	No Privilege for Attempted Operation
No_CSAS	The specified companion standard cannot be found
No_Domain	The specified Domain file does not exist, or cannot be opened
No_File_Ctxt	No File Context, Please Submit an SPR
No_Queue	The specified VMD name list is invalid
No_VMD	The specified VMD does not exist
NotImpYet	Function Not Yet Implemented
Not_Found	Cannot locate specified object
OBJECT_ADDRESS_ERR	Object Address Error
OBJECT_NAME_ERR	Error in object name
OBJ_ACCESS_DENIED	Service Error Class-Access:Denied
OBJ_ACCESS_UNSPRTD	Service Error Class-Access: Unsupported
OBJ_ATTRIB_INCON	Service Error Class-Definition: Object Attr Inconsistent
OBJ_CONSTRAINT_CONFLICT	Service Error Class-Service: Object Constraint Conflict
OBJ_EXISTS	Service Error Class-Definition: Object Exists
OBJ_INVDTD	Service Error Class-Access: Invalidated
OBJ_NON_EXISTENT	Service Error Class-Access: Non-Existent
OBJ_STATE_CONFLICT	Service Error Class-Service: Object State Conflict
OBJ_UNDEFINED	Service Error Class- Definition: Object Undefined
ODFDOMLIS	A domain content list can only have !SL entries
ODFDOMSCO	Domain name is not defined
ODFONEATS	Only one Application Type Specification is allowed
ODFONEMTS	Only one MMS Type Specification is allowed
ODFSTRLEN	String is too long
ODFVMDCRS	Cross-VMD reference (!AD to !AD) is not allowed/FAO=4
ODFVMDSCO	VMD name is not defined
OTHER_ACCESS	Service Error Class-Access: Other
OTHER_APP_REFERENCE	Service Error Class-Applic Ref: Other
OTHER_CANCEL	Service Error Class-Cancel: Other
OTHER_CONCLUDE	Service Error Class-Conclude: Other

Code	Meaning
OTHER_DEFINITION	Service Error Class-Definition: Other
OTHER_ERR_CLASS	Service Error Class-Other Error
OTHER_FILE	Service Error Class- File: Other
OTHER_INITIATE	Service Error Class- Initiate: Other
OTHER_RSRC	Service Error Class-Resource: Other
OTHER_STATE	Service Error Class-Vmd-State : Other
OTHER_SVC	Service Error Class-Service: Other
OTHER_SVC_PREEMPT	Service Error Class-Service Preempt: Other
OTHER_TIME_RESOLUTION	Service Error Class-Time Resolution: Other
ObtainFile_CNF_Err	Obtain File Confirmation Service Error
ObtainFile_IND_Err	Obtain File Indication Service Error
ObtainFile_REQ_Err	Obtain File Request Service Error
ObtainFile_RSP_Err	Obtain File Response Service Error
OpenFile_CNF_Err	Open File Confirmation Service Error
OpenFile_IND_Err	Open File Indication Service Error
OpenFile_REQ_Err	Open File Request Service Error
OpenFile_RSP_Err	Open File Response Service Error
PARAMETER_CBB_INSUFF	Service Error Class-Initiate: Param CBB Insufficient
PDU_SIZE	Service Error Class-Service: Pdu Size
PDataFail	Presentation Data Get Indication Failed
PINOVM	PI depends on a VMD which does not exist
PIREFDOM	PI refers to a domain which does not exist
POSITION_INVD	Service Error Class-File: Position Invalid
PRIMITIVES_OUT_OF_SEQ	Service Error Class-Service: Primitives out of sequence
PROCESSOR_RSRC_UNAVAIL	Service Error Class- Resource: Processor Unavailable
PUTVAL_CNF_ERR	Put Value Confirm Error
PUTVAL_IND_ERR	Put Value Indication Error
PUTVAL_REQ_ERR	Put Value Request Error
PUTVAL_RSP_ERR	Put Value Response Error
P_Abort	Get Indication Received an Presentation Abort Request
Peer_PDU_Rej	Peer PDU was Rejected. Reject Reason: !ZL Reject Code: !ZL>/ FAO_COUNT=2
PrmTooLng	Encoder/Decoder Primitive Too Long
QUEUE_NOT_EMPTY_ERR	Queue Not Empty
RQS_PENDING_ERR	Requests Pending Error
ReadFile_CNF_Err	Read File Confirmation Service Error
ReadFile_IND_Err	Read File Indication Service Error
ReadFile_REQ_Err	Read File Request Service Error

Code	Meaning
ReadFile_RSP_Err	Read File Response Service Error
RejectFail	Reject Service Failed
Remote_Listen_Err_Ind	Received a listen ind error which originated remotely
RenameFile_CNF_Err	Rename File Confirmation Service Error
RenameFile_IND_Err	Rename File Indication Service Error
RenameFile_REQ_Err	Rename File Request Service Error
RenameFile_RSP_Err	Rename File Response Service Error
ReqDwlSeq_IND_Err	Request Download Sequence Indication Service Error
ReqDwlSeq_RSP_Err	Request Download Sequence Response Service Error
ReqUplSeq_IND_Err	Request Upload Sequence Indication Service Error
ReqUplSeq_RSP_Err	Request Upload Sequence Response Service Error
ResetPI_CNF_Err	Reset Program Invocation Confirmation Service Error
ResetPI_REQ_Err	Reset Program Invocation Request Service Error
ResumePI_CNF_Err	Resume Program Invocation Confirmation Service Error
ResumePI_REQ_Err	Resume Program Invocation Request Service Error
SQLERROR	SQL signaled an error
STATE_MACHINE_ID_INVD	Service Error Class-Vmd-State : State Machine ID Invalid
STATUS_CNF_ERR	Status Confirm Error
STATUS_IND_ERR	Status Indication Error
STATUS_REQ_ERR	Status Request Error
SVC_CBB_INSUFF	Service Error Class-Initiate: Svc CBB Insufficient
StartPI_CNF_Err	Start Program Invocation Confirmation Service Error
StartPI_REQ_Err	Start Program Invocation Request Service Error
StopPI_CNF_Err	Stop Program Invocation Confirmation Service Error
StopPI_REQ_Err	Stop Program Invocation Request Service Error
Sy_Dynmem	No Dynamic memory available
TIMEOUT	Service Error Class-Service Preempt: Timeout
TYPE_INCON	Service Error Class-Definition: Type Inconsistent
TYPE_UNSPRTD	Service Error Class-Definition: Type Unsupported
TermDwlSeq_IND_Err	Terminate Download Sequence Indication Service Error
TermDwlSeq_RSP_Err	Terminate Download Sequence Response Service Error
TermUplSeq_CNF_Err	Terminate Upload Sequence Confirmation Service Error
TermUplSeq_REQ_Err	Terminate Upload Sequence Request Service Error
UDFCLS	Undefined Definition Class
UDFVMD	The specified VMD is not defined
UNKNOWNPDU	Unknown PDU type received
UNSPRTBL_TIME_RESOLUTION	Service Error Class-Time Resolution: Unsupportable
UVNODOM	Unnamed Variable depends on a domain which does not exist

Code	Meaning
UVNOVMD	Unnamed Variable depends on a VMD which does not exist
UVREFAT	Unnamed Variable refers to a nonexistent Application Named Type
UncnfRqsFail	Unconfirmed Request Failed
UnxEOM	Unexpected End of Memory
UplSeg_CNF_Err	Upload Segment Confirmation Service Error
UplSeg_REQ_Err	Upload Segment Request Service Error
User_PDU_Rej	PDU Rejected by Peer. Reject Reason: !ZL Reject Code: !ZL/ FAO_COUNT=2
VARIABLE_SPEC_ERR	Variable Specification Error
VERSION_INCOMPATIBL	E Service Error Class-Initiate: Version Incompatible
VLSNODOM	Variable List depends on a domain which does not exist
VLSNOVMD	Variable List depends on a VMD which does not exist
VLSREFAT	Variable List refers to nonexistent Application Named Type
VLSREFVAR	Variable List refers to a nonexistent Variable
VMD_OPERATIONAL_PROB	Service Error Class-Vmd-State : Operational Problem
VMD_STATE_CONFLICT	Service Error Class-Vmd-State : State Conflict

A.5 Severe Error Codes

Table below describes VSImni severe error codes.

Table A.5. VSImni Messages (Severity: Error)

Code	Meaning
ATCNOATS	Type Comp depends on nonexistent App Type Spec
ATSNOAT	App Named Type depends on nonexistent App Named Type
INVFUNC	Connection is not in a valid state for this OMNI function
INVOKEID	Returned Invoke ID does not match
INTPRGERR	Internal Programming Error – Send SPR
MTCNOMTS MMS	Type Comp depends on nonexistent MMS Type Spec
MISSING	Required parameter missing
MTCNOMTS MMS	Type Comp depends on nonexistent MMS Type Spec
MTSNOMT MMS	Type Spec depends on nonexistent MMS Named Type
NETNOSTATE	Provider error, peer requested an invalid function
NOCP OMNI	Control Process not found
NOMBX	Failure creating command mailbox

Code	Meaning
PIDNOPI	Orphaned PI Domain List Entry
SECINTFAI	Secondary Interlock Failure in Queue
UNSUPPORTED	This function is not supported
VLENOVLS	Var List Ele depends on a nonexistent Var List

Appendix B. Data Type Passing Mechanisms

Table B.1 contains the data type passing mechanisms.

Table B.1. Data Type Passing Mechanisms

Data Type	Usage	VAX Calling Standard Binding	C Binding	f77 Binding
Boolean	Read only	By reference	By value	By reference
Boolean	Modify	By reference	By reference	By reference
Character	Read only	By reference	By value	By reference
Character	Modify	By reference	By reference	By reference
Integer	Read only	By reference	By value	By reference
Integer	Modify	By reference	By reference	By reference
Single-precision	Read only	By reference	By value	By reference
Floating	Modify	By reference	By reference	By reference
Procedure	Read only	By reference	By value	By reference
Procedure	Modify	By reference	By reference	By reference
Strings	Read only	By descriptor	By reference ¹	By reference ²
Strings	Modify	By descriptor	By reference ¹	By reference ²
Structure	Read only	By reference	By reference	By reference
Structure	Modify	By reference	By reference	By reference
Array	Read only	By reference	By reference	By reference
Array	Modify	By reference	By reference	By reference
Opaque	Read only	By reference	By reference	By reference
Opaque	Modify	By reference	By reference	By reference

¹The C binding passes a string data type as the address of a null terminated string.

²The f77 binding passes a string data type as two arguments: the string by reference and an integer argument, specifying the length of the string, that is appended to the argument list and passed by value.

Appendix C. MMS Attributes and VSiomni Constants

This appendix provides a cross reference of MMS object attributes and corresponding VSiomni constants.

C.1 VMD

Table below contains attributes and constants that the object VMD supports.

Table C.1. VMD Attribute Support

MMS Object Attribute	VSiomni Constant
Vendor Name	OMNISK_ATTR_VENDOR
Model Name	OMNISK_ATTR_MODEL
Revision	OMNISK_ATTR_REVISION
Logical Status	OMNISK_ATTR_LOGICAL_STATUS
Local Detail	OMNISK_ATTR_LOCAL_DETAIL
List of Capabilities	OMNISK_CLS_CAPABILITY
List of Abstract Syntax	OMNISK_CLS_SYNTAX
List of Programs Invocations	OMNISK_CLS_PROGRAM_INVOCATION
List of Domains	OMNISK_CLS_DOMAIN
List of Event Actions	
List of Event Enrolments	
List of Event Conditions	
List of Named Variables	OMNISK_CLS_NAMED_VARIABLE
List of Scattered Access	
List of Named Variable List	OMNISK_CLS_VARIABLE_LIST
List of Named Types	
List of Semaphore	
List of Journal	
List of Operator Station	

C.2 DOMAIN

Table below contains attributes and constants that the object DOMAIN supports.

Table C.2. DOMAIN Attribute Support

MMS Object Attribute	VSiomni Constant
List of Capabilities	OMNISK_CLS_CAPABILITY
State	OMNISK_ATTR_STATE
MMS Deletable	OMNISK_ATTR_DELETABLE

MMS Object Attribute	VSImoni Constant
Sharable	OMNISK_ATTR_SHARABLE
Upload In Progress	OMNISK_ATTR_UPLD_IN_PROG
List of Programs Invocations	OMNISK_CLS_PROGRAM_INVOCATION
List of Domains	OMNISK_CLS_DOMAIN
List of Event Actions	
List of Event Enrollments	
List of Event Conditions	
List of Named Variables	OMNISK_CLS_NAMED_VARIABLE
List of Scattered Access	
List of Named Variable List	OMNISK_CLS_VARIABLE_LIST
List of Named Types	
List of Semaphore	

C.3 PROGRAM INVOCATION

Table below contains attributes and constants that the object PROGRAM INVOCATION supports.

Table C.3. PROGRAM INVOCATION Attribute Support

MMS Object Attribute	VSImoni Constant
State	OMNISK_ATTR_STATE
List of Domains	OMNISK_CLS_DOMAIN
MMS Deletable	OMNISK_ATTR_DELETABLE
Reusable	OMNISK_ATTR_REUSABLE
Monitor	OMNISK_ATTR_MONITOR
List of Event Actions	
List of Event Enrollments	
List of Event Conditions	
Execution Argument	OMNISK_ATTR_EXE_ARGUMENT

C.4 VARIABLE

Table below contains attributes and constants that the object VARIABLE supports.

Table C.4. VARIABLE Attribute Support

MMS Object Attribute	VSImoni Constant
Unnamed Variable Object	
MMS Deletable	OMNISK_ATTR_DELETABLE
Access Method	OMNISK_ATTR_ACCESS
Type Description	OMNISK_CLS_TYPE_SPEC
Named Variable List Object	

MMS Object Attribute	VSImoni Constant
MMS Deletable	OMNI\$K_ATTR_DELETABLE
List of Variable	OMNI\$K_CLS_VAR
Named Variable Object	
MMS Deletable	OMNI\$K_ATTR_DELETABLE
Access Method	OMNI\$K_ATTR_ACCESS
Type Description	OMNI\$K_CLS_TYPE_SPEC
Address	OMNI\$K_ATTR_ADDRESS_TYPE
Scattered Access	
MMS Deletable	
List of Component	
Kind of Reference	
Named Type	
MMS Deletable	
Type Description	

Appendix D. Supported Mappings

This appendix provides Table D.1 that indicates supported mappings between MMS and Application Types.

Table below needs to be checked!

Table D.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

MMS Type	Application Type
ARRAY [n] OF <MMS type x> ARRAY [s1..s2] OF	<application type y> where $s1 \leq n$, $s2 \leq n$, $s1 \leq s2$ and x and y are a supported mapping
STRUCTURE	STRUCTURE