VSI OpenVMS

VSI OSAP Network Manager's and Programmer's Guide

Document Number: DO-DOSNMG-01A

Publication Date: April 2018

This manual describes how to configure, program and monitor the VSI OSAP environment. VSI OSAP stands for VSI OMNI services for SINEC-AP.

Revision Update Information: This is a new guide.

Operating system and Version: VSI OpenVMS Integrity Version 8.4-1H1

Operating system and Version: VSI OpenVMS Alpha Version 8.4-2L1

Software Version: VSI OSAP for OpenVMS Version 4.1

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

VSI OpenVMS VSI OSAP Network Manager's and Programmer's Guide:

Legal Notice

The VSI OpenVMS documentation set is available on DVD.

Preface

The *VSI OSAP Network Manager's and Programmer's Guide* describes how to configure, program and monitor the VSI OSAP environment. VSI OSAP is a product implementing the Siemens SINEC Automation Protocol (SINEC-AP).

SINEC-AP is a Siemens proprietary protocol which specifies the syntax and semantics for communication between applications running on computers and those running on dedicated factory floor devices such as programmable logical controllers, numerical control machines and robots.

1. Intended Audience

This manual is intended for:

- People who plan to use VSI OSAP in a shop floor plant.
- System managers responsible for maintenance of the OpenVMS system and installation of VSI OSAP
- Network managers who configure, maintain, and manage the VSI OSAP environment.
- Programmers who design and write manufacturing applications layered on VSI OSAP.

2. Prerequisites

You must have a good understanding of the Siemens SINEC Automation Protocol (AP)and the Manufacturing Message Specifications (MMS).

For information on SINEC-AP and related Siemens products, read Appendix A, *Overview of SINEC and Related Siemens Products*.

For information on MMS see *ISO 9506, Manufacturing Message Specification – Service Definition*.

3. Associated Documents

Other manuals in the VSI OSAP documentation set are:

- *VSI OSAP/AP and OSAP/H1 Installation Guide*

Further information can be found in the following manuals:

- *VSI OMNI Application Programmer's Guide*
- *VSI OMNI API Omni Directory Services Guide*

The information contained in these manuals is also applicable to VSI OSAP, with the following constraints:

- MMS-compatible AP and H1 objects and services are a subset of the corresponding objects and services of the MMS model.

  For this reason, the *VSI OSAP Network Manager's and Programmer's Guide* highlights which objects can be defined, and which services can be requested in the VSI OSAP environment, using the VSI OMNI configuration and programming facilities.

- There are objects and services which are specific to AP and H1 (that is, to VSI OSAP). These are fully documented in:

- *VSI OSAP Network Manager's and Programmer's Guide*
- *VSI OMNI Guide to Using OmniView*
- *VSI OMNI Network Manager's Guide*

4. Typographical Conventions

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

Convention	**Meaning**
x	A lowercase italic *x* indicates the generic use of a letter. For example, *xxx* indicates any combination of three alphabetic characters.
n	A lowercase italic *n* indicates the generic use of a number. For example, 19 *nn* indicates a 4-digit number in which the last 2 digits are unknown.
[ ]	In format descriptions, brackets indicate optional elements. You can choose none, one, or all of the options.
( )	In format descriptions, parenthesis delimit the parameter or argument list.
""	Quotation marks enclose system messages that are specified in text.
…	In format descriptions and illustrations, horizontal ellipsis point indicate one of the following: • An item that is repeated • An omission, such as additional optional arguments • Additional parameters, values, or other information that you can enter
italic type	Italic type emphasizes important information, indicates variables, and indicates complete titles of manuals.
boldface type	Boldface type in examples indicates user input. Boldface type in text indicates the first instance of terms defined in the text.
nn nnn.nnn nn	A space character separates groups of 3 digits in numerals with 5 or more digits. For example, *10 000* equals *ten thousand*.
n.nn	A period in numerals signals the decimal point indicator. For example, *1.75* equals *one and three-quarters*.
⋮	Vertical ellipsis dots indicate the omission of information from an example, an illustration, or a command format. The information is omitted because it is not important to the topic being discussed.
UPPERCASE	Words in uppercase indicate a command, the name of a file, the name of a file protection code, an abbreviation for a system privilege, the name of a field, or the value of an attribute where attributes are chosen from a list.
lowercase	In format descriptions, words in lowercase indicate parameters or arguments to be specified by the user.

Part I. Introduction

Chapter 1. Overview

VSI OSAP is a product that allows inter-connection of OpenVMS systems with factory automation devices that support the Siemens SINEC-AP and SINEC-H1 protocols.

1.1. What Does VSI OSAP Stand For?

VSI OSAP stands for VSI OMNI Services for SINEC-AP.

VSI OSAP is a software environment for the development of OpenVMS applications that access and handle SINEC-AP and SINEC-H1 devices, using the facilities of the VSI OMNI high-level application programming interface.

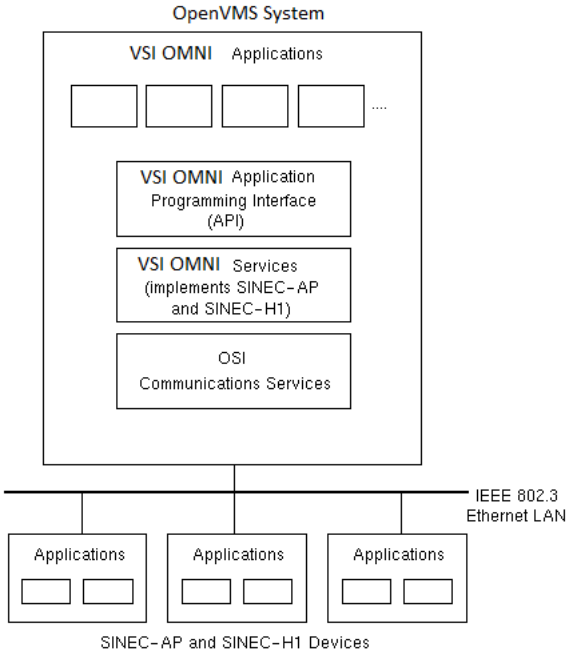
VSI OMNI is the implementation of the 9506 ISO/OSI Manufacturing Message Specifications (MMS).

1.2. VSI OSAP: What It Is and What It Does

VSI OSAP supplies application programmers with a set of high level facilities to develop OpenVMS applications that control and monitor a production plant. This can contain a wide range of SINEC-AP devices (for example, programmable logical controllers, numerical control machines, personal computers and minicomputers), and SINEC-H1 Programmable Logical Controllers (PLCs).

Figure 1.1, “VSI OSAP Scenario” shows an example of a shop floor configuration, including a VAX processor and various SINEC-AP and SINEC-H1 devices, which are interconnected via IEEE 802.3 Ethernet Local Area Networks. Although Siemens assumes SINEC-H1 to be the combination of an Application Protocol with the ISO/OSI Transport on IEEE 802.3, in this manual the term SINEC-H1 refers to the Application Protocol only.

Figure 1.1. VSI OSAP Scenario



VSI OSAP for OpenVMS allows transparent communications between **VSI OSAP applications** running on an OpenVMS system and applications running on any other vendor hardware that supports the SINEC-AP and SINEC-H1 protocol stacks.

Transparent communication is made possible by the **VSI OMNI Application Programming Interface (API)**. This supplies VSI OSAP applications with a set of high-level procedures to create **associations**, that is, logical connections with remote applications, and request or fulfill AP and H1 services.

AP and H1 service requests are passed from the API to the **VSI OSAP Services** component, which is the software layer that implements the AP and H1 protocols. It supplies all the services for the management of associations and the formatting and exchange of messages generated by the AP and H1 service requests. This component guarantees transparent handling of the application requests, sparing the programmer the task of dealing with aspects regarding protocol, plant topology and any communication problems.

The high speed and reliability of the IEEE 802.3 Ethernet Local Area Network guarantees efficient communication between OpenVMS systems and SINEC-AP and SINEC-H1 devices. This communication conforms to the architecture and rules of the ISO/OSI 8073 Class 4 protocol.

1.3. VSI OSAP and VSI OMNI: H1 Towards AP Towards MMS

VSI OSAP for OpenVMS adopts the programming and management facilities of VSI OMNI, which is the implementation of MMS for OpenVMS systems. VSI OSAP for OpenVMS defines the additional abstract syntax that SINEC-AP (or SINEC-H1) requires and adds that to the VSI OMNI run-time library. This leads to:

- Programming uniformity: the same application programming interface and the same configuration and management tools are used in both the SINEC-AP (or SINEC-H1) and MMS environments.
- System interoperability: an application using the VSI OMNI API can communicate with both SINEC-AP (or SINEC-H1) and MMS devices.

As for MMS services, AP services are grouped into classes. An AP service class that is compatible with an MMS service class is said to be open. In order for an application to be portable, you must use the open AP services. SINEC-AP also defines the Serial Transfer class that is classified as not open, because MMS does not provide a corresponding service class. See Section 1.5.2, “Supported AP Services” for a complete list of the AP services supported by VSI OSAP.

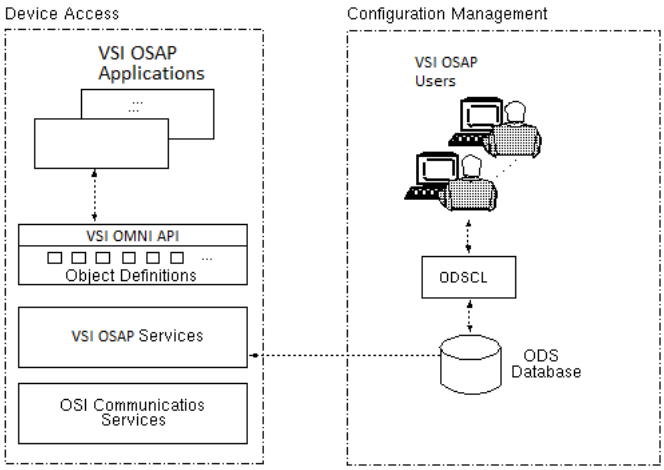
Also H1 services are grouped into open classes (such as the Variable Access services), plus a not open class (Message Services). In this way, H1 services can fit into the MMS- compliant model provided by the VSI OMNI API. See Section 1.6.2, “Supported H1 Services” for a complete list of the H1 services supported by VSI OSAP.

1.4. The VSI OSAP for OpenVMS Environment

The VSI OSAP for OpenVMS software environment provides the facilities to configure, program and monitor an OpenVMS system connected to a SINEC manufacturing plant.

Figure 1.2, “The VSI OSAP Environment” shows the elements of the VSI OSAP environment. Each element is described briefly in the sections below.

Figure 1.2. The VSI OSAP Environment



1.4.1. Users and Applications

VSI OSAP users are network managers responsible for the configuration and management of the environment; for example, they define applications and virtual manufacturing devices (VMDs).

VSI OSAP applications are programs written in any of the languages supported by the OpenVMS operating system. VSI OSAP applications invoke the VSI OMNI API procedures to implement automation tasks.

1.4.2. Configuration Facilities

VSI OSAP configuration facilities allow a manufacturing plant to be represented on the OpenVMS system in terms of:

- AP and H1 objects (such as Virtual Manufacturing Devices, variables and messages).
- Applications that request and fulfill services related to these objects.

Object definitions can be created (and modified) at run time using the Application Programming Interface. Once created, these definitions are directly available in the VSI OSAP application process space.

To define all VSI OSAP objects to the OpenVMS system, you can also use the data configuration facility provided by VSI OMNI: the **Omni Definition Facility (ODF)**. Object definitions are loaded into the VSI OSAP application space at run time, and the VSI OSAP application refers to them when calling the VSI OMNI API procedures.

The information required by the AP and H1 protocols to address both local and remote applications is contained in appropriate data structures called **Application Descriptors**. At run time, VSI OSAP applications use these descriptors to establish associations with remote partner applications, wherever these are located in the network.

You create an Application Descriptor using the **Omni Directory Service Control Language (ODSCL)**.

1.4.3. Programming Facilities

The VSI OMNI API spares the application programmer the task of learning all the details of the SINEC-AP and SINEC-H1 specifications. Instead of using SINEC-AP (or SINEC-H1) services to obtain information on the remote device, a VSI OSAP application interacts with defined objects using the API procedures.

In this way, a VSI OSAP application refers to VMDs and associated objects, and VSI OSAP selects the appropriate service (or services) in order to fulfill the request.

Besides creating object definitions, the VSI OMNI API includes procedures to request local VSI OMNI operations, and to request or fulfill the following AP and H1 services:

- Environment Management
- VMD Support
- Variable Access
- Domain Management
- Program Invocation
- Serial Transfer

1.4.4. Monitoring Facilities

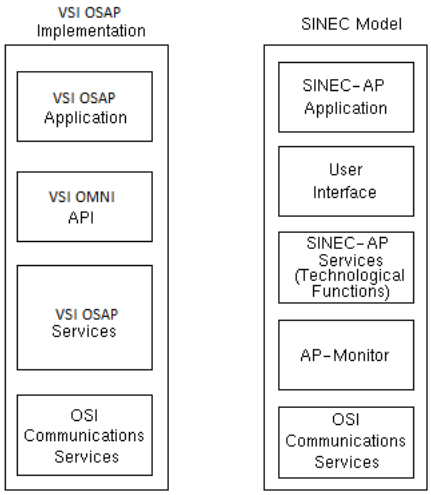
The VOTS trace feature can be used to monitor Transport layer activities.

OmniView is a diagnostic and monitoring tool provided by VSI OMNI that uses DECwindows and the VSI OMNI API to display data associated with a VMD and to write the values of variables that reside on a VMD.

1.5. SINEC-AP Support

Figure 1.3, “VSI OSAP Implementation of SINEC-AP” shows the elements of the SINEC-AP model and their implementations in VSI OSAP.

Figure 1.3. VSI OSAP Implementation of SINEC-AP



The VSI OMNI API procedures have been extended to implement the SINEC-AP application services.

The VSI OSAP Services component implements the SINEC- AP services (that is, the Technological Functions) and the functions of the AP-Monitor. In the SINEC-AP architecture, the AP-Monitor handles all the communication above the Transport Layer (that is, session management, management and matching of requests and acknowledgments, multiplexing and management of Transport Connections).

The OSI Communications Services component implements the OSI stack up to Transport Layer. It is based on VOTS Version 3.0 (included in DECnet-VAX Extensions Version 5.4) and complies with the ISO OSI specifications for the Physical Link (coaxial cable), Data Link, Network Layer and Transport Layer (ISO 8073 Class 4).

1.5.1. Supported AP Objects

VSI OSAP allows definition of the following AP object 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

1.5.2. Supported AP Services

Table 1.1, “Supported AP Services” lists the AP services supported by VSI OSAP:

- *Client support* means that a VSI OSAP application issues a service request related to an object defined at the remote VMD.
- *Server support* means that a VSI OSAP application fulfills a service request related to an object defined at the local VMD.

Table 1.1. Supported AP Services

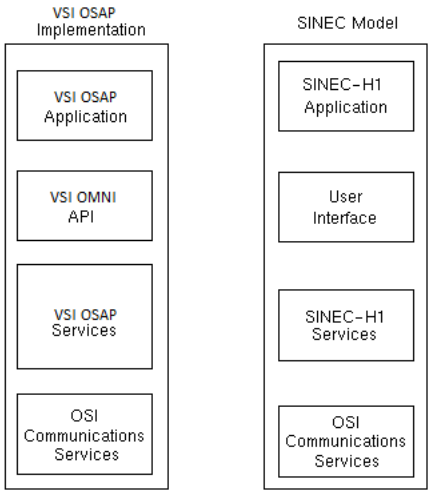
AP Service Name	Client Support	Server Support
Environment Management Services (Open)		
Initiate	x	x
Conclude	x	x
Abort		
VMD Support Services (Open)		
Status	x	x
GetNameList	x	
Identify	x	x
GetCapabilityList	x	
UnsolicitedStatus	x	x
Variable Access Services (Open)		
Read	x	x
Write	x	x
InformationReport	x	x
GetVariableAccessAttributes	x	
Domain Management Services (Open)		
InitiateDownloadSequence	x	
DownloadSegment	x	
TerminateDownloadSequence	x	
InitiateUploadSequence	x	
UploadSegment	x	
TerminateUploadSequence	x	
RequestDomainDownload	x	
RequestDomainUpload	x	
DeleteDomain	x	
GetDomainAttributes	x	
Program Invocation Services (Open)		
CreateProgramInvocation	x	
DeleteProgramInvocation	x	
Start	x	

AP Service Name	Client Support	Server Support
Stop	x	
Resume	x	
Reset	x	
Kill	x	
GetProgramInvocationAttributes	x	
Serial Transfer Services (Not-open)		
Read	x	x
Write (acknowledged or not)	x	x
Exchange	x	x

1.6. SINEC-H1 Support

Figure 1.4, “VSI OSAP Implementation of SINEC-H1” shows the elements of the SINEC-H1 model and their implementations in VSI OSAP.

Figure 1.4. VSI OSAP Implementation of SINEC-H1



The VSI OMNI API procedures have been extended to implement the SINEC-H1 application services.

The VSI OSAP Services component implements the SINEC- H1 services and handles all the communications above the Transport Layer (that is, management and matching of requests and acknowledgment, and management of Transport Connections).

1.6.1. Supported H1 Object

The SINEC-H1 support allows the following object classes to be defined:

- VMD
- Domain
- Program Invocation
- Unnamed Variable
- Message

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

Note

Domain and Program Invocation objects are emulated in SINEC-H1.

1.6.2. Supported H1 Services

Table 1.2, “Supported H1 Services” lists the H1 services supported by VSI OSAP:

- *Client support* means that a VSI OSAP application issues a service request related to an object defined at the remote VMD.
- *Server support* means that a VSI OSAP application fulfills a service request related to an object defined at the local VMD.

Table 1.2. Supported H1 Services

H1 Service	Client Support	Server Support
Environment Management Services		
Connect	x	x
Conclude	x	x
Abort	x	x
VMD Support Services		
Status	x	
Variable Access Services		
Read	x	x
Write	x	x
Domain Management Services		
InitiateDownloadSequence	x	
DownloadSegment	x	
TerminateDownloadSequence	x	
InitiateUploadSequence	x	
UploadSegment	x	
TerminateUploadSequence	x	
Program Invocation Services		
Start	x	
Stop	x	
Message Services		
Send	x	

H1 Service	Client Support	Server Support
Receive	x	

Chapter 2. VSI OSAP Concepts and Facilities

This chapter introduces the concepts on which the VSI OSAP implementation of the Siemens SINEC-AP and SINEC-H1 protocols are based.

The chapter also explains how object definitions are created in the VSI OSAP environment for a sample manufacturing plant, and provides some guidelines for writing VSI OSAP applications.

2.1. AP and H1 Concepts Relevant to VSI OSAP

AP and H1 are Siemens proprietary protocols that support message communications to and from programmable systems in a Computer Integrated Manufacturing (CIM) environment.

AP specifies a set of services to operate on Virtual Manufacturing Devices. AP follows MMS concepts and, as well as MMS, is defined in terms of:

- An abstract model defining interaction among users of the service.
- The procedural requirements associated with the execution of service requests.

Although H1 is not based on an abstract model, it can be integrated into the MMS-compliant VSI OMNI architecture. Moreover, through the VSI OMNI API, the systems can be connected to different and heterogeneous devices.

The elements of this model implemented by VSI OSAP are described in the sections below.

2.1.1. Objects Used to Model a Manufacturing Plant

MMS models a manufacturing plant as a set of abstract objects, called Virtual Manufacturing Devices (VMDs). A VMD is an abstract representation of a specific set of resources and functions of a real manufacturing device, and a mapping of this abstract representation to the physical and functional aspects of a real manufacturing device.

In addition to VMDs and other MMS-compliant objects (such as variables), **messages** can be defined. These are objects specific to the AP and H1 environments (they are not defined in MMS), therefore a user-written application that operates on messages is not portable.

A message is an abstract element of the VMD that operates on strings of bytes. You can use the message object for:

- Reading a value from the remote VMD
- Writing a value to the remote VMD (either acknowledged or not)
- Exchanging values, that is, both reading a value from, and writing a value to the remote VMD.

2.1.2. Client and Server Applications

An application is a program that requests, or fulfills (or both) an AP (or H1) service for one or more VMDs. Service requests always refer to a VMD, and one application can manage several VMDs.

An application can be designed to operate as a Client or a Server, or both:

- The Server is the partner, in the communication, that receives or sends requests related to its own objects.
- The Client is the partner, in the communication, that sends or receives requests related to objects belonging to the peer.

Note

The term **server application** (or client application) used throughout this document indicates either an application, or part of an application acting as a server (or a client).

2.1.3. Associations

Two applications must first establish a logical connection (an association) before they can issue or fulfill service requests.

An application can issue a request to establish an association. The partner can accept or reject the request.

An application can establish one or more associations with remote partner applications running on any devices of the AP manufacturing plant. At any time during the life of the association, either partners can switch from client mode to server mode, and vice versa, independently of which partner requested the association to be established.

Anyone of the applications can issue a request to bring an association to an orderly conclusion. The partner accepts or rejects the request in order to conclude the association.

2.1.4. AP Service Classes

An application can request six classes of AP services to operate on VMDs and other objects. These service classes are described in Table 2.1, “AP Service Classes”

Table 2.1. AP Service Classes

AP Service Class	Description
Environment Management (Open)	Allows an application to establish and terminate an association with a VMD defined on the remote device.
VMD Support (Open)	Allows an application to obtain information on the capabilities and status of a VMD object.
Variable Access (Open)	Allows an application to read, write and get the attributes of variable objects in a VMD.
Domain Management (Open)	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 (Open)	Allows an application to create a program invocation object, and start, stop, delete, kill and resume a program invocation.
Serial Transfer (Not-open)	Allows an application to write (with acknowledgment or not), read and exchange messages.

See Section 1.5.2, “Supported AP Services” for a list of the AP services supported by VSI OSAP for OpenVMS.

2.1.5. H1 Service Classes

An application can request six classes of H1 services to operate on VMDs and other objects. These service classes are described in Table 2.2, “H1 Service Classes”

Table 2.2. H1 Service Classes

H1 Service Class	Description
Environment Management (Open)	Allows an application to establish and terminate an association with a VMD defined on the remote device.
VMD Support (Open)	Allows an application to obtain information on the status of a VMD object.

H1 Service Class	Description
Variable Access (Open)	Allows an application to read and write the attributes of variable objects in a VMD.
Domain Management (Open)	Allows an application to request upload and download operations on a Domain.
Program Invocation Management (Open)	Allows an application to start and stop a program invocation.
Message (Not-open)	Allows an application to send or receive a message.

See Section 1.6.2, “Supported H1 Services” for a list of the H1 services supported by VSI OSAP for OpenVMS.

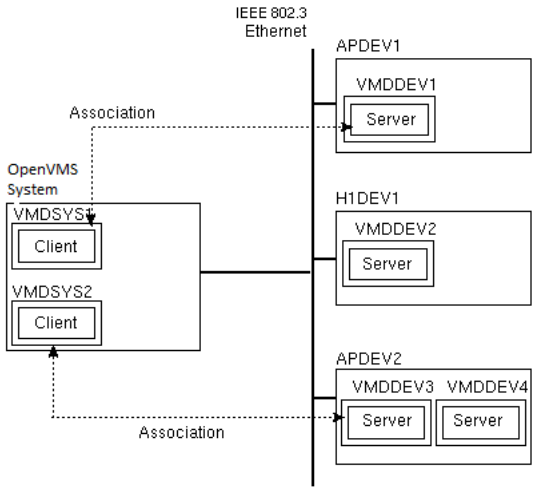
2.2. A Sample Manufacturing Plant

This section presents the facilities provided by the VSI OSAP environment to:

- Create definitions of the plant objects on the OpenVMS system consistent with the AP protocol specifications.
- Create definitions of the plant objects on the OpenVMS system consistent with the AP protocol specifications.

Examples of these facilities are given below. The sample manufacturing plant of Figure 2.1, “Sample Manufacturing Plant” shows a OpenVMS system plus two SINEC-AP and one SINEC-H1 devices, which are interconnected through an IEEE 802.3 Ethernet Local Area Network.

Figure 2.1. Sample Manufacturing Plant



All the devices in the manufacturing plant are modeled using a number of VMDs, where each VMD represents a specific set of resources and functions of the real manufacturing device. A number of applications exist in the plant. At run-time, these applications establish associations to accomplish automation tasks.

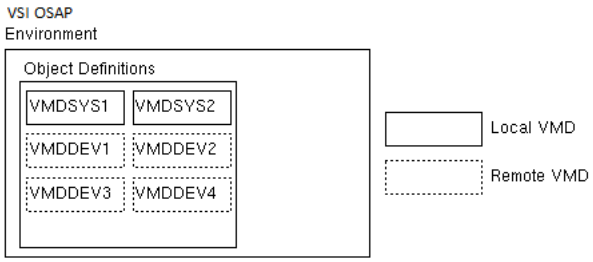
2.2.1. Plant Configuration

The resources of the sample manufacturing plant shown in Figure 2.1, “Sample Manufacturing Plant” are represented on the OpenVMS system as shown in Figure 2.2, “Definitions of Local and Remote VMDs for the Sample Manufacturing Plant” Local definitions are created for:

- Local VMDs, that is, VMDs mapping sets of resources and functions on the OpenVMS system (named VMDSYS1 and VMDSYS2 in Figure 2.2, “Definitions of Local and Remote VMDs for the Sample Manufacturing Plant”)
- Remote VMDs, that is, VMDs mapping sets of resources and functions on the remote devices (named VMDDEV1, VMDDEV2, VMDDEV3 and VMDDEV4 in Figure 2.2, “Definitions of Local and Remote VMDs for the Sample Manufacturing Plant”)

Although they are not shown in Figure 2.2, “Definitions of Local and Remote VMDs for the Sample Manufacturing Plant” definitions of the objects associated with each VMD are also created.

Figure 2.2. Definitions of Local and Remote VMDs for the Sample Manufacturing Plant



At run-time, a VSI OSAP application refers to the local definition of a VMD (and its associated objects) via its name, regardless of where the device and the resources mapped by the VMD are physically located.

To address a VMD, VSI OSAP uses the information contained in a structure called an **Application Descriptor (AD)**.

An AD exists for each VMD definition. It specifies all the communication-related information required by VSI OSAP to address the local or remote VMD.

ADs are created and stored in the OMNI Directory Services by means of ODSCL.

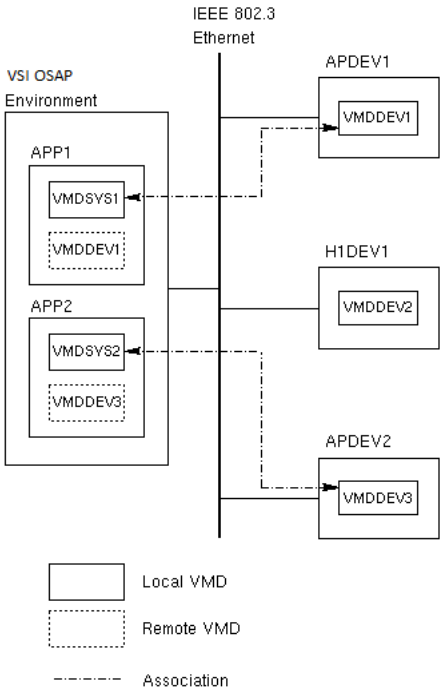
2.2.2. Plant Operation

Once the definitions of the plant objects have been created, VSI OSAP applications can use the procedures of the VSI OMNI API to request and fulfill AP and H1 services.

Several applications can run simultaneously in the VSI OSAP environment, and each application can be designed to perform client functions, server functions, or both.

Figure 2.3, “Associations in the Sample Manufacturing Plant” shows two VSI OSAP applications (APP1 and APP2) that communicate with a device in the sample manufacturing plant.

Figure 2.3. Associations in the Sample Manufacturing Plant



APP1 refers to VMDSYS1 (the local VMD) and VMDDEV1 (the local mapping of the remote VMD) to establish an association between the OpenVMS system and APDEV1.

APP2 refers to VMDSYS2 (the local VMD) and VMDDEV3 (the local mapping of the remote VMD) to establish an association between the OpenVMS system and APDEV2.

APP1 and APP2 contain a sequence of VSI OMNI API procedure calls to perform typical functions, such as:

- To make the object definitions available in the VSI OSAP application process space. This is done by invoking the API procedures for run-time object definition, or by loading the definitions from the OMNI database. ODF commands are used to store object definitions in the OMNI database at configuration time.
- To request VSI OSAP to establish an association with a remote VMD.
- To issue service requests that can be accomplished by the remote VMD (that is, by the remote server application acting as a VMD for the particular service request instance). Service requests that can be issued are for example: reading the value of a variable from the remote device, obtaining the status of a VMD, and so on.
- To request VSI OSAP to terminate the association.

Part II. Working with SINEC-AP

Chapter 3. Configuration

3.1. Plant Object Definitions

The run-time object definition facility of the VSI OMNI API allows the creation of definitions for the AP objects that represent the manufacturing plant. The VSI OMNI API procedures used to create run-time definitions of MMS objects are:

- omni_define
- omni_get_definition
- omni_modify_definition

They have been extended to allow the definition of:

- The message object (which is specific to AP).
- Additional attributes for AP objects that comply with MMS objects (such as VMD).

Part IV, “Extension to VSI OMNI Procedure Calls” explains how to use the extended API procedures to create run-time object definitions. Alternatively, object definitions can be created using the Omni Definition Facility (ODF), and then loaded onto the application process space at run time.

This chapter explains how to use ODF to create local definitions of VSI OSAP objects representing the manufacturing plant. In addition to the commands that are documented in the *VSI OMNI Network Manager's Guide*, ODF provides a specific command for the definition of AP message objects (messages do not exist in the MMS model).

This chapter explains how to use the *VSI OMNI Network Manager's Guide* and provides the additional information required to operate with ODF in the VSI OSAP environment.

3.1.1. ODF Definitions

The following sections explain how to use ODF to create definitions of AP objects.

3.1.1.1. The OSAP Application Profile

ODF has been designed to operate in both the MMS (default) and in the AP contexts (that is, in the VSI OSAP context).

As explained in the *VSI OMNI Network Manager's Guide*, the MMS context is set by default when ODF is invoked.

Before using any other ODF command, you must issue the SET APPLICATION PROFILE command to set ODF command syntax and operation for VSI OSAP definitions created under ODF.

Setting the VSI OSAP context ensures that subsequent ODF commands are interpreted correctly, and the definitions created under ODF refer to the VSI OSAP environment.

Example 3.1, “How to Set the OSAP Application Profile(AP)” shows how to invoke ODF and how to set the OSAP application profile.

Example 3.1. How to Set the OSAP Application Profile(AP)

```
$ ODF
ODF> SET APPLICATION PROFILE OSAP; ❶
```

```
ODF> SHOW APPLICATION PROFILE;     ❷
        Application Profile is OSAP
ODF>
```

❶ After activating ODF, enter the SET APPLICATION PROFILE command, specifying the OSAP attribute for the OSAP application profile.

❷ You are now in the VSI OSAP context; this can be verified by issuing the SHOW APPLICATION PROFILE command.

3.1.1.2. Supported Definition Classes

ODF can be used to create definitions of the following AP objects:

- 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

Refer to the *VSI OMNI Network Manager's Guide* for a general description of the commands used to create the definitions of the objects listed above. Read this chapter also for information on specific considerations on these commands when they are used to create object definitions in the VSI OSAP context. See Chapter 4, *ODF Command Extension Reference* of this manual for the reference to the DEFINE MESSAGE command.

3.1.1.3. Creating VMD Definitions

You create a local definition of a VMD using the ODF DEFINE VMD command. For a general description of this command, refer to the *VSI OMNI Network Manager's Guide*. Specific considerations on the VSI OSAP context are discussed in the sections below which contain the:

- List of attributes for the DEFINE VMD command that assume a specific significance in the VSI OSAP context
- Conformance Building Blocks supported by VSI OSAP
- List of the identifiers for the open and not-open AP ser- vices supported by VSI OSAP.

Specific Attributes for the DEFINE VMD Command

Table 3.1, “Attributes of the DEFINE VMD Command that have a Specific Meaning in the VSI OSAP Context (AP)” contains a list and an explanation of the DEFINE VMD command attributes that assume a specific significance in the VSI OSAP context.

Table 3.1. Attributes of the DEFINE VMD Command that have a Specific Meaning in the VSI OSAP Context (AP)

Attribute	Explanation
APPLICATION SIMPLE NAME	Name of the Application Descriptor (AD) containing the information required by VSI OSAP to address a local or a remote application. ADs are created using ODSCL, as described in Section 3.2, “Application Descriptors” of this manual.
PARAMETER CBB	The set of conformance building blocks supported by the VMD. See *the section called “Conformance Building Blocks”* for further details.
SUPPORTED SERVICES	The set of open Services supported by the user that calls the VMD. See *the section called “VMD Supported Services”* for further details.
NOT OPEN SUPPORTED SERVICES	The set of not-open Services supported by the user that calls the VMD. See *the section called “VMD Supported Services”* for further details.
CPU IDENTIFIER	A VMD attribute used to identify the type of Siemens device. See *the section called “CPU Identifiers”* for further details.

Any attributes that are not included in the table below, retain the same syntax and semantics in both the AP and MMS contexts.

Conformance Building Blocks

Table 3.2, “Supported Conformance Building Blocks (AP)” lists the Conformance Building Blocks (CBB) supported by VSI OSAP. The table also lists the values that can be specified in the PARAMETER CBB attribute of the DEFINE VMD command.

Table 3.2. Supported Conformance Building Blocks (AP)

PARAMETER CBB Value	ISO 9506 Designation
[NO]ARRAY	STR1
[NO]STRUCTURES	STR2
[NO]NAMED VARIABLES	VNAM
[NO]ALTERNATE ACCESS	VALT
[NO]UNNAMED VARIABLES	VADR

Note

VSI OSAP ignores PARAMETER CBB values different from those listed in Table 3.2, “Supported Conformance Building Blocks (AP)”

VMD Supported Services

Table 3.3, “Open AP Service Identifiers” and Table 3.4, “Not Open AP Service Identifiers” list the open and not-open AP services that are supported by VSI OSAP, together with their identifiers. You can specify one or several AP service identifiers in the SUPPORTED SERVICES and NOT OPEN SUPPORTED SERVICES attributes of the DEFINE VMD command to indicate that the corresponding service is supported by the VMD being defined.

The DEFAULT SUPPORT column of the table indicates the AP services for which a default support is provided (Yes); if you want the VMD being defined supports these services, you can omit the corresponding service identifiers in the SUPPORTED SERVICES (or NOT OPEN SUPPORTED SERVICES) attribute.

Table 3.3. Open AP Service Identifiers

AP Service	Service Identifier	Default Support
Environment Management		

AP Service	Service Identifier	Default Support
Initiate	(Whether or not the Initiate service is supported, it is specified in the Application Descriptor associated with the VMD. See Section 3.2, “Application Descriptors” for details).	Yes
Cancel	CANCEL	No
Conclude	CONCLUDE	Yes
Abort	(All VMD definitions support this service).	Yes
VMD Support Services		
Status	STATUS	No
GetNameList	GET NAME LIST	Yes
Identify	IDENTIFY	Yes
GetCapabilityList	GET CAPABILITY LIST	No
UnsolicitedStatus	UNSOLICITED STATUS	Yes
Variable Access Services		
Read	READ	Yes
Write	WRITE	Yes
InformationReport	INFORMATION REPORT	Yes
GetVariableAccessAttributes	GET VARIABLE ACCESS ATTRIBUTES	Yes
Domain Management Services		
InitiateDownloadSequence	INITIATE DOWNLOAD SEQUENCE	Yes
DownloadSegment	DOWNLOAD SEGMENT	Yes
TerminateDownloadSequence	TERMINATE DOWNLOAD SEQUENCE	Yes
InitiateUploadSequence	INITIATE UPLOAD SEQUENCE	Yes
UploadSegment	UPLOAD SEGMENT	Yes
TerminateUploadSequence	TERMINATE UPLOAD SEQUENCE	Yes
RequestDomainDownload	REQUEST DOMAIN DOWNLOAD	Yes
RequestDomainUpload	REQUEST DOMAIN UPLOAD	Yes
DeleteDomain	DELETE DOMAIN	Yes
GetDomainAttributes	GET DOMAIN ATTRIBUTES	No
Program Invocation Services		
CreateProgramInvocation	CREATE PROGRAM INVOCATION	No
DeleteProgramInvocation	DELETE PROGRAM INVOCATION	No
Start	START	No
Stop	STOP	No
Resume	RESUME	No
Reset	RESET	No
Kill	KILL	No
GetProgramInvocationAttributes	GET PROGRAM INVOCATION ATTRIBUTES	No

Table 3.4. Not Open AP Service Identifiers

AP Service	Service Identifier	Default Support
Serial Transfer Services		

AP Service	Service Identifier	Default Support
Read Message	READ	Yes
Write Message	WRITE	Yes
Exchange Message	EXCHANGE	Yes

Note

VSI OSAP ignores SUPPORTED SERVICES and NOT OPEN SUPPORTED SERVICES values different from those listed in Table 3.3, “Open AP Service Identifiers” and Table 3.4, “Not Open AP Service Identifiers” Any attempt made by an application to request the service is rejected by the API at run time.

When defining a VMD, you must specify the supported services by assigning the appropriate values to the SUPPORTED SERVICES and NOT OPEN SUPPORTED SERVICES attributes. While doing this, bear in mind the following considerations:

1. Before using the DEFINE VMD command to create a local definition of a remote VMD, collect accurate information on the model, characteristics and services supported by the remote device. For example, there are Siemens AP devices which do not support the Initiate service. Appendix A, *Overview of SINEC and Related Siemens Products* provides a list of the supported services for a range of Siemens AP products.

2. VSI OSAP uses the values specified in the SUPPORTED SERVICES and NOT OPEN SUPPORTED SERVICES attributes to negotiate the characteristics of the association with the partner application. On completion of the negotiation, a VSI OSAP application can only request the services that are accepted by both partners. Therefore, the values for which no agreement was reached during the negotiation are considered as not supported. Any attempt made by an application to issue a request for services referring to unsupported values, causes the API to return an error.

Example 3.2, “Creating the Local Definition of a Remote AP VMD” shows how to create the local definition of a remote VMD. A VSI OSAP client uses this definition to establish an association, and to issue service requests to a remote VMD.

Example 3.2. Creating the Local Definition of a Remote AP VMD

```
ODF>
ODF> DEFINE VMD VMD1DEV1 APP SIM NAM AD_OSAPPLCI;
ODF>
```

The mnemonic name VMD1DEV1 identifies the local definition of the remote VMD.

AD_OSAPPLCI is the name of the Application Descriptor (AD) that contains the addressing information related to the remote application acting as a server for the remote VMD. The ADs contain information that enable the VSI OSAP client application to access the remote VMD on the DEV1 remote device at run time. A full description of how to create and operate ADs is given in Section 3.2, “Application Descriptors”

Example 3.2, “Creating the Local Definition of a Remote AP VMD” shows that, apart from the specified parameters, all the default parameters have been accepted.

Example 3.3, “Creating the Local Definition of a Local AP VMD” shows how to create the local definition of a local VMD. A VSI OSAP application uses this definition to declare itself to VSI OSAP as the server application, and thus to accept service requests from a remote client application.

Example 3.3. Creating the Local Definition of a Local AP VMD

```
ODF>
ODF> DEFINE VMD VMD3SYS1 APP SIM NAM AD_OSAPCLTI;
ODF>
```

VMD3SYS1 is the mnemonic name that identifies the local definition of a VMD residing on the SYS1 VSI OSAP system.

AD_OSAPCLTI is the name of the Application Descriptor that contains the addressing information associated to the VSI OSAP application acting as a server for the local VMD.

Example 3.3, “Creating the Local Definition of a Local AP VMD” shows that, apart from the specified parameters, all the default parameters have been accepted.

CPU Identifiers

The CPU IDENTIFIER attribute of the VMD definition specifies the type of Siemens device. Possible values are shown in Table 3.5, “CPU Identifiers (AP)”

Table 3.5. CPU Identifiers (AP)

Identifier	Type of Siemens Device
1	SIMATIC_115U-941
2	SIMATIC_115U-942
3	SIMATIC_115U-943
4	SIMATIC_115U-944
5	SIMATIC_135U-921
6	SIMATIC_135U-922
7	SIMATIC_135U-928
8	SIMATIC_150U
9	SIMATIC_155U
10	GRACIS_ELAN

3.1.1.4. Creating Unnamed Variable Definitions

When using the DEFINE UNNAMED VARIABLE command to create the definition of an unnamed variable associated with the local definition of a local or remote VMD, the <address> attribute has the following format:
`<address type><address string>`

where:

- address type must be set to UNCONSTRAINED ADDRESS
- address string must be set to any valid user-defined string, as explained below.

The definitions associated with unnamed variables defined on remote AP devices must specify a valid product-dependent address. Refer to the Siemens product documentation of the specific AP manufacturing device for information on how to formulate a valid address. However, the syntax of the address string attribute requires a VSI OSAP-specific value to specify one of the following addresses:

- Numeric Address format
- Reduced Unconstrained Address format

Numeric Address Format

In SINEC-AP, the Numeric Address format is an internal address information related to a Named Variable defined on the device's Communication Processor. It can be obtained through a GetVariableAccessAttributes service request for the specified named variable. At this point, you can define an unnamed variable by specifying the same type definition as for a named variable and the numeric address already obtained. It is now possible to access the device variable as either a named or an unnamed variable; the latter type of access is faster than the former.

Reduced Unconstrained Address Format

The address of a memory location on the AP device containing the VMD to which the variable is associated. Valid address strings have the following format: `"BlockType:BlockNumber:StartingAddress"`

BlockType can assume any of the following values: DB, FY, IB, QB, PY, CB, TB, RS, AS, DX, DE, OY, RDB, RDX, PDB and PDX.

The defined string must be recorded in an omni_t_address_str word counted structure.

The Reduced Unconstrained Address format can be used for PLC devices only.

3.1.1.5. Creating Message Definitions

AP messages are the objects referred to by the non-open Serial Transfer Services.

Using AP messages, partner applications can read, write and exchange strings of bytes. (Applications themselves are in charge of preparing and interpreting the message contents.)

A message is an object specific to the AP model (it is not part of the MMS model) and its definition is created using the ODF DEFINE MESSAGE command. Only one message object definition can be associated with each VMD created in the VSI OSAP context.

Even though the DEFINE MESSAGE command respects the syntax and operation of the VSI OMNI ODF, it can only be used in the VSI OSAP context. Therefore, how to create a message object definition is fully explained in the sections below, and the DEFINE MESSAGE command reference is documented in Chapter 4, *ODF Command Extension Reference* of this manual.

A message definition includes the following attributes:

- Name of the message
- Description of the message
- Length of the message

Example 3.4, “Creating an AP Message Definition” shows how to create a message definition.

Example 3.4. Creating an AP Message Definition

```
ODF> ❶
ODF> DEFINE MESSAGE VMD1DEV1:MY_MSG, LENGTH 300, ❷
        DESCRIPTION "Used in subsequent examples";
ODF>
```

❶ Make sure that the OSAP Application Profile has been set, by entering the SHOW AP command; the system should respond with the message ``Application Profile is OSAP".

❷ The LENGTH attribute (300) specifies the maximum length of the message object value (a string of bytes).

A VSI OSAP application can now use the message object definition MY_MSG to write, read, or exchange an AP message with the remote AP device represented by the VMD1DEV1 VMD.

3.2. Application Descriptors

For the transparent management of an association between a VSI OSAP application and a remote application, create two Application Descriptors (one for the VSI OSAP application and the other for the remote partner), and provide a link between each AD and the definition of the (local or remote) VMD.

To create an Application Descriptor, create an entry in the OMNI Directory Services using the OMNI Directory Service Control Language (ODSCL).

To provide the link, insert the name of the AD in the Application Simple Name attribute of the local (or remote) VMD definition.

At run-time, when a VSI OSAP application requests an association to be established with a remote VMD, it specifies the name of the local definition of a remote VMD. This carries the name of the AD where VSI OSAP finds the information to address the remote application (such as the NETADDRESS of the remote device and the TSAP of the remote application), in order to establish the requested association.

A VSI OSAP application that wants to accept a connection request coming from a remote VMD, specifies the name of the local VMD on which the association request is accepted. The information about the remote VMD is provided to the VSI OSAP application when the incoming connection request is received. The association request is accepted if the local definition of the remote VMD already exists, or if an AD for the remote VMD has been recorded. In the latter case, first a local definition of the remote VMD is created, and then the association request is accepted.

3.2.1. Application Descriptor Data for a Remote Partner Application

The definition of an AD for a remote partner application contains addressing information which is valid for a remote server or a remote client. For example, when you create an AD for a remote server application (see also Figure 3.1, “An AD for a Remote AP Server Application”) you must specify:

- The NETADDRESS of the remote AP system where the server of the remote VMD is active. (In Figure 3.1, “An AD for a Remote AP Server Application” IEEE%080006010001 also specifies the Ethernet address of the remote AP device on the SINEC-H1 LAN).
- The remote TSAP name associated with the remote VMD (that is, associated with the remote server application).
- The MUX attribute value, which uniquely identifies one of the multiple associations simultaneously active on the same Transport Connection. (This attribute must be set to 0 if no multiplexing is requested).
- The NEGOTIATION attribute, which can be set to TRUE or FALSE:
  - TRUE specifies that the association is established as soon as the Initiate request sent by a partner has been accepted by the other partner.
  - FALSE specifies that the association is established as soon as the underlying Transport Connection has been set up.
- The attributes for the management of the association: COMPLTIMER and RESPTIMER (see also Table 3.6, “Attributes of ODSCL Command Arguments (AP)” These attributes refer to requests that flow from the VSI OSAP client application to the remote server application.

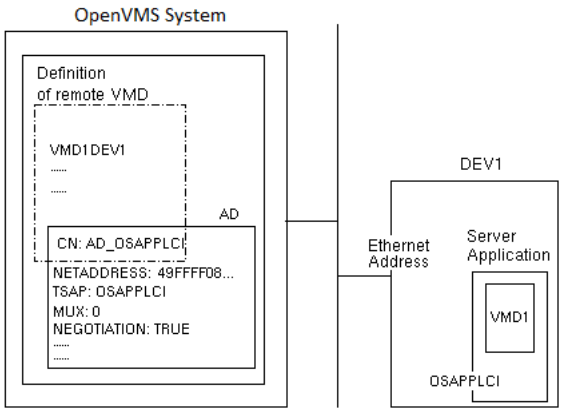
Table 3.6. Attributes of ODSCL Command Arguments (AP)

Attribute	Description
/TSAP= *value*	It is the Transport Service Access Point for the local or remote application. It must be the same value as that defined for the partner device.
/NETADDRESS= *value*	It is a name of maximum 64 alphanumeric characters that corresponds to the Network Address of the device on which the application being defined is active.
/MUX= *value*	A number identifying an association (an integer between 0 and 255). Together with the TSAP, it identifies one of the multiple associations multiplexed on the same Transport Connection. Default is 0 (that is, no multiplexing).
/COMPLTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse between the request for a service and its completion. The API returns

Attribute	Description
	an error to the VSI OSAP application if the time expires before a response is received. If COMPLTIMER is set to 0, the time interval is infinite (that is, timer disabled). Default is 0 seconds.
/RESPTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse before a service request is forwarded again, if no response is received. The value of this parameter is usually a submultiple of the value assigned to COMPLTIMER. A service request is forwarded once if RESPTIMER is set to 0 (that is, timer disabled). Default is 0 seconds.
/NEGOTIATION= *value*	Specifies when an association between the VSI OSAP application and the remote application is established. A value TRUE specifies that the association is established following the successful setup of the Transport Connection and completion of the Initiate request. A value FALSE specifies that the association is established as soon as the underlying Transport Connection has been set up (the Initiate request is not sent). Default is TRUE.

Remember that you must specify the name of the Application Descriptor (AD_OSAPPLCI in Figure 3.1, “An AD for a Remote AP Server Application” when you create the definition of the remote VMD (Application Simple Name attribute).

Figure 3.1. An AD for a Remote AP Server Application



3.2.2. Application Descriptor Data for a VSI OSAP Application

The definition of an AD for a VSI OSAP application contains addressing information which is valid for a local server or a local client. For example, when you create an AD for a VSI OSAP server application (see also Figure 3.2, “An AD for a VSI OSAP AP Server Application”) you must specify:

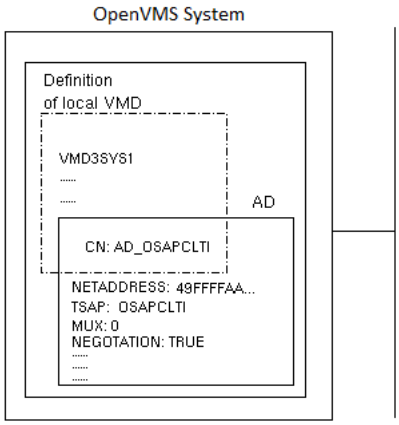
- The NETADDRESS, which also specifies the Ethernet address of the OpenVMS system (that is, IEEE %AA000400DFB9).
- The name of the TSAP on the OpenVMS system. A remote client application refers to this name when requesting an association to be established with the VSI OSAP application acting as the server for the local VMD.
- The MUX attribute value, which identifies one of the multiple associations simultaneously active on the same Transport Connection. (This attribute must be set to 0 if no multiplexing is requested).
- The NEGOTIATION attribute, which can be set to TRUE or FALSE:
  - TRUE specifies that the association is established as soon as the Initiate request sent by a partner has been accepted by the other partner.

- FALSE specifies that the association is established as soon as the underlying Transport Connection has been set up.

The COMPLTIMER and RESPTIMER attributes must not be specified when creating an AD for a VSI OSAP server, because the association is managed by the remote clients.

Remember that you must specify the name of the Application Descriptor (AD_OSAPCLTI in Figure 3.2, “An AD for a VSI OSAP AP Server Application” when you create the definition of the local VMD (Application Simple Name attribute).

Figure 3.2. An AD for a VSI OSAP AP Server Application



3.2.3. ODSCL Commands to Create Application Descriptors

The ODSCL command REGISTER DIRECTORY is used to create an Application Descriptor. It is also possible to:

- Remove an AD (DEREGISTER DIRECTORY command).
- Modify an AD (MODIFY DIRECTORY command).
- Display all AD attributes (READ DIRECTORY command).
- Display ADs in the directory matching the selection criteria specified, or all entries in the directory (LIST DIRECTORY command).
- Establish the current name directory path used to refer to an AD based on its relative position in the name hierarchy (SET command), and display the current name directory path (SHOW command).

General information on ODSCL commands (including com- mand syntax, ODSCL Command Language Interface, invoking and exiting ODSCL) can be found in the *VSI OMNI Network Manager's Guide*.

Tables Table 3.7, “Arguments of ODSCL Commands (AP)” and Table 3.8, “Attributes of ODSCL Command Arguments (AP)” show the VSI OSAP-specific arguments and attributes that must be specified when using ODSCL commands to create Application Descriptors.

Table 3.7. Arguments of ODSCL Commands (AP)

Argument	Description
NAME "/CN= *common_ name*"	Identifies the name of the Application Descriptor to be created. The simple name of an application can consist of up to 64 characters, which must come from the printable string character set. /CN is the name attribute abbreviation as defined in the known attributes types file.

Argument	Description
ATTRIBUTES "OC=OSAP / *attr_abbrev=value...* "	The set of attributes and the attribute values listed in Table 3.8, “Attributes of ODSCL Command Arguments (AP)” OC is the object class abbreviation as defined in the known object classes file.

Table 3.8. Attributes of ODSCL Command Arguments (AP)

Attribute	Description
/TSAP= *value*	It is the Transport Service Access Point for the local or remote application. It must be the same value as that defined for the partner device.
/NETADDRESS= *value*	It is a name of maximum 64 alphanumeric characters that corresponds to the Network Address of the device on which the application being defined is active.
/MUX= *value*	A number identifying an association (an integer between 0 and 255). Together with the TSAP, it identifies one of the multiple associations multiplexed on the same Transport Connection. Default is 0 (that is, no multiplexing).
/COMPLTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse between the request for a service and its completion. The API returns an error to the VSI OSAP application if the time expires before a response is received. If COMPLTIMER is set to 0, the time interval is infinite (that is, timer disabled). Default is 0 seconds.
/RESPTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse before a service request is forwarded again, if no response is received. The value of this parameter is usually a submultiple of the value assigned to COMPLTIMER. A service request is forwarded once if RESPTIMER is set to 0 (that is, timer disabled). Default is 0 seconds.
/NEGOTIATION= *value*	Specifies when an association between the VSI OSAP application and the remote application is established. A value TRUE specifies that the association is established following the successful setup of the Transport Connection and completion of the Initiate request. A value FALSE specifies that the association is established as soon as the underlying Transport Connection has been set up (the Initiate request is not sent). Default is TRUE.

3.2.4. Examples

The following examples show how to use the REGISTER DIRECTORY command to create an AD for a remote server application and an AD for a VSI OSAP server application.

Example 3.5, “Creating an AD for a Remote AP Server” shows how to create an AD that defines the addressing information for a remote server application (that is, the server that owns a VMD mapping resources and functions of a remote AP device).

Example 3.5. Creating an AD for a Remote AP Server

```
REGISTER DIRECTORY NAME "/CN=AD_OSAPPLCI" ATTRIBUTES
                                          "OC=OSAP
                                          /NETADDRESS=IEEE%080006010001
                                          /TSAP=OSAPPLCI
                                          /MUX=0
                                          /NEGOTIATION=TRUE"
```

It is assumed that "IEEE" is the template name defined in NCL for Null Internet support under the OSI Transport.

The AD_OSAPPLCI application name specified in Example 3.5, “Creating an AD for a Remote AP Server” must be the same name assigned to the Application Simple Name attribute of the definition of the VMD1DEV1 VMD.

NETADDRESS consists of the VOTS template and also contains the Ethernet address of the remote system on the Ethernet LAN.

OSAPPLCI is a TSAP name defined on the remote AP system and corresponds to the Transport Address of the remote server application (that is, of the remote VMD).

At run-time, an association initiation request issued by a VSI OSAP client application specifies the name VMD1DEV1 (local VMD definition); VSI OSAP forwards the request to the remote server application identified by the OSAPPLCI TSAP and by the Ethernet address contained in the NETADDRESS.

Example 3.6, “Creating an AD for a VSI OSAP AP Server” shows how to create an AD that defines the addressing information for a local server (that is, the server that owns a VMD mapping resources and functions of the OpenVMS system).

Example 3.6. Creating an AD for a VSI OSAP AP Server

```
REGISTER DIRECTORY NAME "/CN=AD_OSAPCLTI" ATTRIBUTES
                                          "OC=OSAP
                                          /TSAP=OSAPCLTI
                                          /NETADDRESS=IEEE%AA000400DFB9
                                          /MUX=0
                                          /NEGOTIATION=TRUE"
```

Example 3.7, “Modifying an AD (AP)” shows how to use the MODIFY DIRECTORY command to replace the TSAP value (OSAPPLCI) with a new value (OSAPSRVI).

Example 3.7. Modifying an AD (AP)

```
MODIFY DIRECTORY NAME "/CN=AD_OSAPPLCI" REPLACE
                                      "TSAP=OSAPPLCI,OSAPSRVI"
```

Chapter 4. ODF Command Extension Reference

This chapter describes the DEFINE MESSAGE command, which extends the VSI OMNI ODF facility to cover the requirements of the VSI OSAP environment. The DEFINE MESSAGE command is added to the ODF command set to allow the creation of message object definitions.

The following documentation conventions are used:

Convention	Meaning
[ ]	Square brackets enclose optional expressions.

Reviewers need to decipher the lines below and check their meaning in the next column.
 	
8
  <
  :
  <val1>
  <val2>
  <val3>
 	
  9
  =
  ;
	Large braces enclose choices from a group of items. Braces around a single item indicate that this item is mandatory.

< >	Angle brackets enclose tokens that must be expanded.
. . .	Ellipsis indicates an expression that can be repeated.

A local ODF definition name has the same format as an MMS identifier: it is a string of 1 to 32 characters. All alphanumeric characters, the dollar sign ($), and the underscore character (_) are valid. The identifier cannot begin with a numeric character.

Table 4.1, “ODF Naming Format in the AP Context ” shows the valid naming formats for definitions specified when ODF commands are used in the VSI OSAP context.

Table 4.1. ODF Naming Format in the AP Context

Definition	Naming Format and Examples
VMD	*vmd_name* vmd
Named Variable	*[vmd_name]* (NAMED VARIABLE *:var_name)* v(NAMED VARIABLE:n), :(NV:n), (NV:n)
Message	*[vmd_name]* (NAMED VARIABLE *:var_name)* v(NAMED VARIABLE:n), :(NV:n), (NV:n)

DEFINE MESSAGE

DEFINE MESSAGE — Creates a local VSI OMNI definition of a message object and associates it with a previously defined VMD. A given VMD definition can have one associated message object definition.

Format

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

Attributes and Values

<vmd_name>

Name of the VMD to which the variable belongs. If omit- ted, ODF uses the default VMD that was set with the SET SCOPE command.

<vmd_name> is an MMS identifier.

<msg_name>

Name of the message object being defined.

<msg_name> is an MMS identifier.

LENGTH <msg_length>

Maximum length of the message in bytes. This number specifies the amount of memory space to be allocated for the message at run time.

<msg_length> is a positive integer in the range 1 to 8192.

DESCRIPTION<text>

Information identifying the message.

<text> is a quoted character string. Default = "".

Chapter 5. Programming with VSI OSAP

This chapter describes how to use the VSI OMNI API procedures to write VSI OSAP applications.

5.1. Introduction

Programming in the VSI OSAP environment means preparing applications that use the VSI OMNI API procedures to communicate with partner applications running on SINEC-AP devices.

Bear in mind the following considerations when using API procedures to write VSI OSAP applications:

- General syntax and programming rules, described in *VSI OMNI Application Programmer's Guide*, are also valid when the API procedures are used in the VSI OSAP environment.
- Open AP services are a functional subset of the MMS services implemented by the VSI OMNI API. Therefore, when reading the *VSI OMNI Application Programmer's Guide*, ignore those descriptions referring to services and functions that are not supported by the VSI OSAP implementation of AP.

  The VSI OMNI API procedures that can be used to write VSI OSAP applications are listed in Section 5.3, “AP Services and API Procedures”.
- The extensions of the VSI OMNI API procedures that support the AP Serial Transfer services are described in Chapter 12, *Extension to VSI OMNI Procedure Calls* of this manual. Extension means that some procedure attributes require or return values that are related to the services of the Serial Transfer class.

Before a VSI OSAP application can be prepared, the following prerequisites must be met:

- The VSI OSAP environment must have been configured on the OpenVMS system according to the instructions given in Chapter 3, *Configuration*.
- You must have a basic knowledge of the VSI OMNI API procedures. Introductory chapters of the *VSI OMNI Application Programmer's Guide* and this chapter provide the information you require to write VSI OSAP applications.

Chapter 7, *A Complete Example of a VSI OSAP Application* contains an example that shows how to write a C Language VSI OSAP application.

5.2. User Include Files

In addition to the VSI OMNI files (omni_defs. *lang* and omni_ codes. *lang*), VSI OSAP provides the following file:

- osap_codes. *lang*

  Include file containing VSI OSAP-specific completion and error codes. Found in SYS$LIBRARY.

  VSI OSAP-specific definitions are automatically available through omni_defs. *lang*.

5.3. AP Services and API Procedures

Table 5.1, “API Procedures that Can Be Invoked by a VSI OSAP AP Application” lists the API procedures that can be invoked by a VSI OSAP application.

Table 5.1. API Procedures that Can Be Invoked by a VSI OSAP AP Application

Procedure Name	AP Service Class/ Local VSI OMNI Operation
omni_abort	Environment Management

Procedure Name	AP Service Class/ Local VSI OMNI Operation
omni_accept_conclude	Environment Management
omni_accept_connect	Environment Management
omni_cancel	Local VSI OMNI Operation (see Section 5.8.1, “Canceling a Request”)
omni_conclude	Environment Management
omni_connect	Environment Management
omni_create	Program Invocation
omni_define	Local VSI OMNI Operation
omni_delete	Program Invocation and Domain Management
omni_download	Domain Management
omni_end_list	Local VSI OMNI Operation
omni_exchange_data	Serial Transfer
omni_get_attribute	Local VSI OMNI Operation
omni_get_definition	Local VSI OMNI Operation
omni_get_handle_by_name	Local VSI OMNI Operation
omni_get_handle_list	Local VSI OMNI Operation
omni_get_indications	Local VSI OMNI Operation
omni_get_message_text	Local VSI OMNI Operation
omni_get_remote_attributes	VMD Support, Variable Access, Domain Management and Program Invocation
omni_get_value	Variable Access and Serial Transfer
omni_group_variables	Local VSI OMNI Operation
omni_initialize	Local VSI OMNI Operation
omni_kill	Program Invocation
omni_listen	Environment Management
omni_load_definitions	Local VSI OMNI Operation
omni_modify_definition	Local VSI OMNI Operation
omni_poll	Local VSI OMNI Operation
omni_print_message	Local VSI OMNI Operation
omni_put_value	Variable Access and Serial Transfer
omni_reject	Local VSI OMNI Operation
omni_reject_conclude	Environment Management
omni_reject_connect	Environment Management
omni_reset	Program Invocation
omni_resume	Program Invocation
omni_send_value	MD Support, Variable Access and Serial Transfer
omni_set_application_profile	Local VSI OMNI Operation
omni_start	Program Invocation
omni_stop	Program Invocation
omni_terminate	Local VSI OMNI Operation
omni_upload	Domain Management

Note

VSI OMNI API procedures not listed in Table 5.1, “API Procedures that Can Be Invoked by a VSI OSAP AP Application” should be ignored.

5.3.1. Mapping AP Client Service Requests into API Procedures

Table 5.2, “Invoking API Procedures to Issue AP Client Requests” shows the API procedures that a VSI OSAP application calls to issue client requests.

Table 5.2. Invoking API Procedures to Issue AP Client Requests

AP Service	API Procedure Call
Environment Management Services	
Initiate	omni_connect
Conclude	omni_conclude
Abort	omni_abort
VMD Support Services	
Status	omni_get_remote_attributes specifying a VMD and omni_c_attr_all, followed by omni_get_attribute
GetNameList	omni_get_remote_attributes specifying a VMD and omni_c_cls_xxx (where xxx identifies the object type), followed by omni-get_attribute
Identify	omni_get_remote_attributes specifying a VMD and omni_c_attr_all, followed by omni_get_attribute
GetCapabilityList	omni_get_remote_attributes specifying a VMD and omni_c_attr_all, followed by omni_get_attribute
UnsolicitedStatus	omni_get_indications + omni_get_attribute
Domain Management Services	
InitiateDownloadSequence, DownloadSegment, TerminateDownloadSegment	omni_download
InitiateUploadSequence, UploadSegment, TerminateUploadSequence	omni_upload
RequestDomainDownload	Hidden
RequestDomainUpload	Hidden
DeleteDomain	omni_delete
GetDomainAttributes	omni_get_remote_attributes + omni_get_attribute specifying a domain and omni_c_attr_all
Program Invocation Services	
CreateProgramInvocation	omni_create
DeleteProgramInvocation	omni_delete
Start	omni_start
Stop	omni_stop
Resume	omni_resume
Reset	omni_reset

AP Service	API Procedure Call
Kill	omni_kill
GetProgramInvocationAttributes	omni_get_remote_attributes specifying a P.I. and omni_c_attr_all + omni_get_attribute
Serial Transfer Services	
Read	omni_get_value specifying a message
Write (acknowledged)	omni_put_value
Write (not acknowledged)	omni_get_indications + omni_get_value
Exchange	omni_exchange_data

5.3.2. Mapping AP Server Service Requests into API Procedures

Table 5.3, “Invoking API Procedures to Fulfill AP Server Requests” shows the API procedures that a VSI OSAP application calls to fulfill server requests.

Table 5.3. Invoking API Procedures to Fulfill AP Server Requests

AP Service	API Procedure Call
Environment Management	
Initiate	omni_listen + omni_accept_connect (or omni_reject_connect)
Conclude	omni_get_indications + omni_accept_conclude (or omni_reject_conclude)
Abort	omni_get_indications
VMD Support Services	
Status	omni_modify_definition specifying omni_c_attr_logical_status, or omni_c_attr_physical_status, or both
Identify	omni_modify_definition specifying omni_c_attr_vendor, omni_c_attr_model and omni_c_attr_revision
UnsolicitedStatus	omni_modify_definition specifying omni_c_attr_logical_status, or omni_c_attr_physical_status, or both, followed by omni_send_value
Variable Access Services	
Write	omni_get_indications + omni_get_value (or omni_reject)
Read	omni_get_indications + omni_put_value (or omni_reject)
InformationReport	omni_send_value specifying a variable
Serial Transfer Services	
Read	omni_get_indications + omni_put_value (or omni_reject)
Write (acknowledged)	omni_get_indications + omni_get_value (or omni_reject)
Write (not acknowledged)	omni_send_value

AP Service	API Procedure Call
Exchange	omni_get_indications + omni_get_value (optional) + omni_exchange_data (or omni_reject)

5.4. The OSAP Application Profile

A VSI OSAP application invokes the omni_set_application_ profile procedure to specify the OSAP application profile. This ensures that definitions created after the call to omni_set_ application_profile are assumed to be associated with the specified profile.

The following example shows how to invoke the omni_set_ application_profile to set the OSAP application profile.

```
status = omni_set_application_profile (omni_c_app_profile_osap)
```

5.5. Run-time Definitions

The API procedures used to create definitions at run-time are omni_define, omni_get_definition and omni_modify_ definition.

5.5.1. Supported Definition Classes

The VSI OMNI run-time object definition facility can be used to create definitions of the following AP objects:

- 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

Refer to the *VSI OMNI Application Programmer's Guide* for a general description of the procedures used to create run-time definitions of the objects listed above. Read the following sections for specific considerations on these procedures when they are used to create object definitions in the VSI OSAP context. See Chapter 12, *Extension to VSI OMNI Procedure Calls* for the complete reference to the omni_define, omni_modify_definition, and omni_get_definition procedures.

5.5.2. Creating VMD Definitions

The following VMD attributes are specific to VMD definitions created in the VSI OSAP context using the omni_modify_ definition procedure:

- omni_osap_c_attr_notopen_srvspt

The set of non-open AP services supported by the user that calls the VMD. See the description of the omni_ modify_definition procedure in Chapter 12, *Extension to VSI OMNI Procedure Calls* for those supported by VSI OSAP.

- omni_osap_c_attr_cpu_id

  The CPU address of the remote device. Four different CPUs can be addressed (from 1 to 4). This attribute must be specified if Unnamed Variable support is requested.

5.5.2.1. Considerations on VMD Supported Services

VSI OSAP ignores unsupported VMD services. Any attempt made by an application to request VMD services not supported by VSI OSAP is rejected by the API at run time.

The open AP services supported by the VMD are specified by assigning the appropriate values to the omni_c_attr_vmd_ srv_supported attribute of the omni_modify_definition procedure. (See the description of the procedure in Chapter 12, *Extension to VSI OMNI Procedure Calls*). While doing this, bear in mind the following considerations:

- Before using the omni_modify_definition procedure to create a local definition of a remote VMD, collect accurate information on the model, characteristics, and services supported by the remote device. For example, there are Siemens AP devices which do not support the Initiate service. Appendix A, *Overview of SINEC and Related Siemens Products* provides a list of the supported services for a range of Siemens AP products.
- VSI OSAP uses the values specified in the omni_c_attr_vmd_param_supported attribute to negotiate the characteristics of the association with the partner application. On completion of the negotiation, a VSI OSAP application can only request the services that are accepted by both partners. Therefore, the values for which no agreement was reached during the negotiation are considered as not supported. Any attempt made by an application to issue a request for ser- vices referring to unsupported values, causes the API to return an error.
- The omni_c_attr_vmd_srv_supported and omni_osap_c_ attr_notopen_srvspt attributes are used by the VSI OSAP application to inform the remote application of the names of the supported services. VSI OSAP rejects any indications received for unsupported services, without notifying the application.

Example 5.1, “Creating the Run-Time Local Definition of a Remote AP VMD” shows how to create the local definition of a remote VMD. A VSI OSAP client application uses this definition to establish an association, and to issue service requests to a remote VMD.

Example 5.1. Creating the Run-Time Local Definition of a Remote AP VMD

```
int status;
      omni_l_handle H_RemoteVMD
      omni_r_iosb Iosb;
      omni_l_context Context =0;
      .
      .
      .
      /* Create a VMD Definition */
      status = omni_define (omni_c_cls_vmd, &H_RemoteVMD);
      .
      .
      .
      /* Modify VMD Attributes */
      Attribute = omni_c_attr_name;
      Context = 0;
      status = omni_modify_definition (H_RemoteVMD, &Attribute,
                                          &Context, "VMD1DEV1",
 strlen("VMD1DEV1"));
      .
      .
```

```
      .
      Attribute = omni_c_attr_vmd_appl_simple_name;
      Context = 0;
      status = omni_modify_definition (H_RemoteVMD, &Attribute,
                                              &Context, "APP1_DEV1",
 strlen("APP1_DEV1"));
      .
      .
      .
      Attribute = omni_c_attr_scope;
      Context = 0;
      status = omni_modify_definition (H_RemoteVMD, &Attribute,
                                              &Context, 0, 0);
      .
      .
      .
```

The mnemonic name VMD1DEV1 identifies the local definition of the remote VMD.

APP1_DEV1 is the name of the Application Descriptor (AD) that contains the addressing information related to the re- mote application acting as a server for the remote VMD. The AD contains information used by the VSI OSAP Client application to access the remote VMD on the DEV1 remote device at run-time. A full description of how to create and operate ADs is given in Section 3.2, “Application Descriptors”.

Example 5.1, “Creating the Run-Time Local Definition of a Remote AP VMD” shows that, apart from the specified parameters, all the default parameters have been accepted.

Example 5.2, “Creating the Run-Time Local Definition of a Local AP VMD” shows how to create the local definition of a local VMD. A VSI OSAP application uses this definition to declare itself to VSI OSAP as the server application, and thus to accept service requests from a remote client application.

Example 5.2. Creating the Run-Time Local Definition of a Local AP VMD

```
int status;
      omni_l-handle H_LocalVMD;
      omni_r_iosb Iosb;
      omni_l_context Context =0;
      .
      .
      .
      /* Create a VMD Definition */
      status = omni_define (omni_c_cls_vmd, &H_LocalVMD);
      .
      .
      .
      /* Modify VMD Attributes */
      Attribute = omni_c_attr_name;
      Context = 0;
      status = omni_modify_definition (H_LocalVMD, &Attribute,
                                                      &Context,
 "VMD3SYS1",

 strlen("VMD3SYS1"));
      .
      .
      .
      Attribute = omni_c_attr_vmd_appl_simple_name;
      Context = 0;
      status = omni_modify_definition (H_LocalVMD, &Attribute,
```

```
                                                        &Context,
"APP3_SYS1",

strlen("APP3_SYS1"));
     .
     .
     .
     Attribute = omni_c_attr_scope;
     Context = 0;
     status = omni_modify_definition (H_LocalVMD, &Attribute,
                                                        &Context, 0, 0);
     .
     .
     .
```

VMD3SYS1 is the mnemonic name that identifies the local definition of a VMD residing on the SYS1 VSI OSAP system.

APP3_SYS1 is the name of the Application Descriptor that contains the addressing information associated to the VSI OSAP application acting as a server for the local VMD.

Example 5.2, “Creating the Run-Time Local Definition of a Local AP VMD” shows that, apart from the specified parameters, all the default parameters have been accepted.

5.5.2.2. Defaults for VMD Attributes

The following are defaults for omni_c_attr_vmd_param_ supported, omni_c_attr_vmd_srv_supported and omni_osap_c_attr_ notopen_srvspt.

Default for omni_c_attr_vmd_param_supported:

```
       (0) STR1 (Arrays) TRUE
       (1) STR2 (Structures) TRUE
       (2) VNAM (Named Variables) TRUE
       (3) VALT (Alternate Access) TRUE
       (4) VADR (Unnamed Variables) TRUE
       (5) VSCA (Scattered Access) FALSE
       (6) TPY (Third Party) FALSE
       (7) VLIS (Named Variable List) FALSE
       (8) REAL (Real) FALSE
       (9) AKEC (Acknowledgment Event Conditions) FALSE
      (10) CEI (Evaluation Interval) FALSE
```

Default for omni_c_attr_srv_supported:

```
          /* vmd support services */
       (0) Status TRUE
       (1) GetNameList FALSE
       (2) Identify TRUE
       (3) Rename FALSE
             /* variable access services */
       (4) Read TRUE
       (5) Write TRUE
       (6) GetVariableAccessAttributes FALSE
       (7) DefineNamedVariable FALSE
       (8) DefineScatteredAccess FALSE
       (9) GetScatteredAccessAttributes FALSE
     (10) DeleteVariableAccess FALSE
     (11) DefineNamedVariableList FALSE
     (12) GetNamedVariableListAttributes FALSE
```

```
(13) DeleteNamedVariableList FALSE
(14) DefineNamedType FALSE
(15) GetNamedTypeAttributes FALSE
(16) DeleteNamedType FALSE
       /* operator communication services */
(17) Input FALSE
(18) Output FALSE
       /* semaphore management services */
(19) TakeControl FALSE
(20) RelinquishControl FALSE
(21) DefineSemaphore FALSE
(22) DeleteSemaphore FALSE
(23) ReportSemaphoreStatus FALSE
(24) ReportPoolSemaphoreStatus FALSE
(25) ReportSemaphoreEntryStatus FALSE

       /* domain management services */
(26) InitiateDownloadSequence FALSE
(27) DownloadSegment TRUE
(28) TerminateDownloadSequence TRUE
(29) InitiateUploadSequence FALSE
(30) UploadSegment FALSE
(31) TerminateUploadSequence FALSE
(32) RequestDomainDownload TRUE
(33) RequestDomainUpload TRUE
(34) LoadDomainContent FALSE
(35) StoreDomainContent FALSE
(36) DeleteDomain FALSE
(37) GetDomainAttributes FALSE
       /* program invocation management services */
(38) CreateProgramInvocation FALSE
(39) DeleteProgramInvocation FALSE
(40) Start FALSE
(41) Stop FALSE
(42) Resume FALSE
(43) Reset FALSE
(44) Kill FALSE
(45) GetProgramInvocationAttributes FALSE
       /* file management services */
(46) ObtainFile FALSE
       /* event management services */
(47) DefineEventCondition FALSE
(48) DeleteEventCondition FALSE
(49) GetEventConditionAttributes FALSE
(50) ReportEventConditionStatus FALSE
(51) AlterEventConditionmonitoring FALSE
(52) TriggerEvent FALSE
(53) DefineEventAction FALSE
(54) DeleteEventAction FALSE
(55) GetEventActionAttributes FALSE
(56) ReportEventActionStatus FALSE
(57) DefineEventEnrollment FALSE
(58) DeleteEventEnrollment FALSE
(59) AlterEventEnrollment FALSE
(60) ReportEventEnrollmentStatus FALSE
(61) GetEventEnrollmentAttributes FALSE
(62) AcknowledgeEventNotification FALSE
(63) GetAlarmSummary FALSE
```

```
(64) GetAlarmEnrollmentSummary FALSE
       /* journal management services */
(65) ReadJournal FALSE
(66) WriteJournal FALSE
(67) InitializeJournal FALSE
(68) ReportJournalStatus FALSE
(69) CreateJournal FALSE
(70) DeleteJournal FALSE

       /* vmd support services */
(71) GetCapabilityList FALSE
       /* file management services */
(72) FileOpen FALSE
(73) FileRead FALSE
(74) FileClose FALSE
(75) FileRename FALSE
(76) FileDelete FALSE
(77) FileDirectory FALSE
       /* unconfirmed services */
(78) UnsolicitedStatus TRUE
(79) InformationReport TRUE
(80) EventNotification FALSE
(81) AttachToEventCondition FALSE
(82) AttachToSemaphore FALSE
       /* additional services */
(83) Conclude TRUE
(84) Cancel FALSE
```

Default for omni_osap_c_attr_vmd_notopen_srvspt:

```
     /* serial transfer services */
(0) Read TRUE
(1) Write TRUE
(2) Exchange TRUE
```

5.5.3. Creating Unnamed Variable Definitions

When using the omni_modify_definition procedure to create the definition of an unnamed variable associated with the local definition of a local or remote VMD, you must set:

- The address type attribute identified by the omni_c_ attr_address_type constant. The supported types are: Numeric (omni_c_address_numeric) and Unconstrained (omni_c_address_unconstrained).
- The address value attribute. For the Numeric format, it is identified by the constant omni_c_attr_number and must be a longword. For the Unconstrained format, it is identified by the constant omni_c_attr_address_string and its type is omni_t_address_str.

5.5.3.1. Local Definition of an Unnamed Variable

The definitions associated with unnamed variables defined on remote AP devices must specify a valid product-dependent address. Refer to the Siemens product documentation of the specific AP manufacturing device for information on how to formulate a valid address. However, the syntax of the address string attribute requires a VSI OSAP-specific value to indicate one of the following addresses:

- Numeric Address format
- Reduced Unconstrained Address format
- Extended Unconstrained Address format

Numeric Address Format

In SINEC-AP, the Numeric Address format is an internal address information related to a Named Variable defined on the device's Communication Processor. It can be obtained through a GetVariableAccessAttributes service request for the specified named variable. At this point, you can define an unnamed variable by specifying the same type definition as for a named variable and the numeric address already obtained. It is now possible to access the device variable as either a named or an unnamed variable; the latter type of access is faster than the former.

Reduced Unconstrained Address Format

The address of a memory location on the AP device containing the VMD to which the variable is associated. Valid address strings have the following format:

```
"BlockType:BlockNumber:StartingAddress"
```

BlockType can assume any of the following values: DB, FY, IB, QB, PY, CB, TB, RS, AS, DX, DE, OY, RDB, RDX, PDB and PDX.

The defined string must be recorded in an omni_t_address_ str word counted structure.

The Reduced Unconstrained Address format can be used for PLC devices only.

Extended Unconstrained Address Format

Some other manufacturing devices, for example the GRACIS system, support an extended address format. This format has several degrees of freedom so the VSI OSAP user is requested to explicitly provide the entire address structure, as it must appear on the PDU sent to the device.

There are two types of extended format:

- Display Flag = 0 (see Figure 5.1, “Extended Unconstrained AP Address Format (Display Flag = 0)”)
- Display Flag = 1 (see Figure 5.2, “Extended Unconstrained AP Address Format (Display Flag = 1)”)

where:

- CPU Id. is the CPU identifier whose value may be 1, 2, 3 or 4.
- Block Type, Block Number and Starting Address form the Unnamed Variable Address. The Block Type must assume one of the following numeric values:

```
01H (DB)
02H (FY)
03H (IB)
04H (QB)
05H (PY)
06H (CB)
07H (TB)
08H (RS)
09H (AS)
0AH (DX)
10H (DE)
11H (OY)
21H (RDB)
2AH (RDX)
41H (PDB)
4AH (PDX)
```

- Data Length is the length of the variable data. The device model determines whether this value must be expressed in words or in bytes.
- Status Words are reserved words for the device usage.

- Device Dependent Data is a portion containing different information depending on the device model. For the GRACIS system this portion has been already specified (see Figure 5.3, “Device Dependent Data”).

Refer to the GRACIS documentation for the meaning of these fields.

VSI OSAP provides a predefined data structure for the GRACIS system address format, called omni_osap_r_extended_address_wc (see the VSI OSAP include file omni_integrator2_defs_include. *lang*). It is a word- counted structure that must be provided to the omni_ modify_definition call for the attribute omni_c_attr_address_str. Note that the GRACIS format falls in the first type, that is with Display Flag equal to 0.

Other devices could have different layouts for the Device Specific Data portion, therefore you must take care of defining the proper data structures.

Note

The Device Specific Data portion is put unchanged into the PDUs so the user must take care of data conversion. In particular, the bytes order is different between a Siemens machine (Big Endian) and an OpenVMS machine (Little Endian); for example, a data word like the GPV field must be provided in the reverse order (00 01 instead of 01 00).

Figure 5.1. Extended Unconstrained AP Address Format (Display Flag = 0)

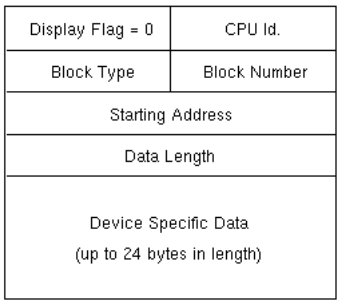


Figure 5.2. Extended Unconstrained AP Address Format (Display Flag = 1)

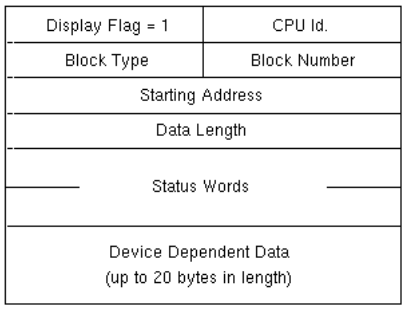
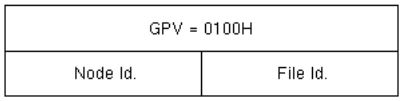


Figure 5.3. Device Dependent Data



5.5.4. Creating Message Definitions

AP messages are the objects referred to by the non-open Serial Transfer Services.

Using AP messages, partner applications can read, write and exchange strings of bytes. (Applications themselves are responsible for preparing and interpreting the message contents.)

Although an AP message is not part of the MMS model, its definition is created and modified using the omni_define and omni_modify_definition procedures.

Only one message object definition can be associated with each VMD definition created in the VSI OSAP context.

A message definition includes the following attributes:

- Name of the message
- Description of the message
- Length of the message
- Handle of a VMD definition

Example 5.3, “Creating a Run-Time AP Message Definition” shows how to create a message object definition.

Example 5.3. Creating a Run-Time AP Message Definition

```
      .
      .
      .
      /* Create a Message Definition */
      status = omni_define (omni_c_cls_msg, &H_TargetMessage);
      .
      .
      .
      /* Modify Message Attributes */
      Attribute = omni_c_attr_name;
      Context = 0;
      status = omni_modify_definition (H_TargetMessage, &Attribute,
                                                        &Context, "MY_MSG",
 strlen("MYMSG"));
      Attribute = omni_c_attr_length;
      msg_length = 300;
      Context = 0;
      status = omni_modify_definition (H_TargetMessage, &Attribute,
                                                        &Context,
 msg_length, sizeof(msg_length));
      Attribute = omni_c_attr_description;
      Context = 0;
      status = omni_modify_definition (H_TargetMessage, &Attribute,
 &Context,
                                                        "Used in subsequent
 examples");
      Attribute = omni_c_attr_scope;
      Context = 0;
      status = omni_modify_definition (H_TargetMessage, &Attribute,
                                                        &Context,
 &H_TargetVMD,

 sizeof(omni_l_handle));
```

For the procedure references, see Chapter 12, *Extension to VSI OMNI Procedure Calls*.

The name of the definition, the length and description attributes are set. The length attribute (300) specifies the maximum length of the message object value (a string of bytes).

The message is committed (that is, associated) to the VMD1DEV1 VMD that was created in Example 5.1, “Creating the Run-Time Local Definition of a Remote AP VMD”.

A VSI OSAP application can now use the message object definition MY_MSG to write, read, or exchange an AP message with the remote AP device represented by the VMD1DEV1 VMD.

5.6. Variable Access Services

A VSI OSAP client application can issue requests for reading or writing the value of a variable (either named or unnamed) associated with a local or remote VMD.

A VSI OSAP server application can fulfill requests for reading or writing the value of a variable (either named or unnamed) associated with a local VMD. (The read or write request can be issued by a local or a remote client application).

Note

VSI OSAP provides client and server support for both named and unnamed variables.

5.7. Serial Transfer Services

Serial Transfer services are specific to AP. Remember that they are classified as non-open in respect to MMS. This sec- tion describes how to use the API procedures to request and fulfill Serial Transfer service requests.

See Table 5.2, “Invoking API Procedures to Issue AP Client Requests” and Table 5.3, “Invoking API Procedures to Fulfill AP Server Requests” for the list of the API procedures that a VSI OSAP application invokes to request or fulfill Serial Transfer Services.

The extensions of the VSI OMNI API procedures that support the Serial Transfer services is given in Chapter 12, *Extension to VSI OMNI Procedure Calls* of this manual. Extension means that some procedure attributes require or return values that are related to Serial Transfer services. Moreover, the omni_exchange_data procedure is specific to the VSI OSAP environment.

Object handles specified in the API procedures called to request or fulfill Serial Transfer services must refer to message type objects.

A VSI OSAP client application can ask a remote server application for the following Serial Transfer Services:

- Read a message from, and write a message to the remote partner.
- Exchange a message with the remote partner.

A VSI OSAP client application can also receive an indication for an unacknowledged message.

A VSI OSAP server application can:

- Receive and fulfill an incoming read or write message request.
- Receive and fulfill an exchange message request.
- Send an unacknowledged message.

The following sections explain how a VSI OSAP application requests and fulfills Serial Transfer Services using the API procedures.

5.7.1. Reading and Writing a Message

To read a message from a remote server application active on an AP device, a VSI OSAP client application invokes the omni_ get_value procedure and supplies the handle of the message definition.

To write a message to a remote server application active on an AP device, a VSI OSAP client application invokes the omni_ put_value procedure and supplies the handle of the message definition.

Received and transmitted data consist of strings of bytes, where each string has a maximum length of 8 192 bytes. The meaning of the received data is defined by the two partner applications.

Example 5.4, “A VSI OSAP Client Application Reads and Writes an AP Message” shows how a VSI OSAP client application reads a message from, and writes a message to the remote server application.

Example 5.4. A VSI OSAP Client Application Reads and Writes an AP Message

```
main () MAIN_PROGRAM
{
    struct
    {
          unsigned short msg_length;
          char msg_buffer[100];
    } Received, Send;
    omni_l_handle TargetVMD;
    omni_l_handle TargetMessage;
    omni_r_iosb Iosb;
    .
    .
    .
    status = omni_get_value (0, TargetMessage, 0, &Received,
                                          sizeof(Received), 0, &Iosb, 0);
    .
    .
    .
    strcpy (Send.msg_buffer, "This is my message");
    MsgDsc.Length = 100;
    status = omni_put_value (0, TargetMessage, 0, &Send,
                                          sizeof(Send), 0, &Iosb, 0);
    .
    .
    .
```

5.7.2. Exchanging a Message

To exchange a message with the remote partner, a VSI OSAP application invokes the omni_exchange_data procedure and supplies:

- The handle of the message definition
- A data structure that contains the message value to be sent:
  - The first two bytes of the data structure specify the size (in bytes) of the message value.
  - The remaining bytes of the data structure contain the message value. The length of the data structure must also be specified.
- A data structure to receive the message:
  - The first two bytes of the data structure must specify the buffer length (in bytes). On completion, they contain the actual length of the message value.
  - The remaining bytes of the data structure contain the value of the received message. The length of the data structure must also be specified.

5.7.3. Receiving and Fulfilling an Incoming Read (or Write) Message Request

A VSI OSAP server application invokes the omni_get_ indications procedure to receive a read (or write) message indication.

To send (or receive) the message value, the application invokes the omni_put_value (or omni_get_value) procedure.

5.7.4. Receiving and Fulfilling an Incoming Exchange Message Request

A VSI OSAP server application invokes the omni_get_indications procedure to receive an Exchange message indication.

Optionally, it invokes the omni_get_value procedure to read the input message (for example, to check the incoming message contents). Finally, it prepares the output message and fulfills the request by invoking the omni_exchange_data procedure.

5.7.5. Sending an Unacknowledged Message

To send an unacknowledged message, a VSI OSAP server application invokes the omni_send_value procedure and supplies:

- The handle of the message definition
- The handle of the remote VMD definition

The remote client application receives a send message indication by invoking the omni_get_indications procedure, and then reads the message by invoking the omni_get_value procedure.

5.8. Local VSI OMNI Operations

5.8.1. Canceling a Request

The MMS Cancel service that is implemented by the omni_ cancel API procedure is not defined in AP.

VSI OSAP emulates this service as a local VSI OMNI operation. When receiving a cancel request, VSI OSAP performs the following operations:

- It executes the request locally (no AP-PDU is sent).
- Once the request has been canceled, it discards the service confirmation, without notifying the application.

Chapter 6. Monitoring the VSI OSAP Environment

The VSI OSAP network manager can use the OmniView monitoring utility. This chapter presents the utility and tells where it is documented.

6.1. VSI OMNI Monitoring Utility

The VSI OMNI monitoring utility for the VSI OSAP environment is OmniView. OmniView is a diagnostics and monitoring tool that uses DECwindows to obtain VSI OSAP services without writing any user application.

Before running OmniView in the VSI OSAP environment you must:

- Define an entry in the VSI OMNI Directory Service (through ODSCL) for each VMD that you wish to use inside OmniView. These entries must belong to a particular ODS Object Class, called VMD, which naming attribute is "/VN" (VMD Name), as shown in the example below. The value provided in the Application Simple Name (APPSN) attribute must be the equal to a "/CN" naming attribute of a valid entry in ODS belonging to the OSAP Object Class.

  ```
  REGISTER DIRECTORY NAME "/VN = VN_OSAPCLTI" ATTRIBUTES
                         "OC = VMD
                          /APROF = OSAP
                          /APPSN = (/CN=AD_OSAPCLTI)
                          /INUM = 11"
  ```

- Define the logical name OMNI_CALLING_VMD equivalent to a valid entry in ODS for the VMD Object Class (see the example below). This entry must correspond to a definition for a local VMD; it will be used by OmniView as the Calling VMD when it establishes associations.

  ```
  $ DEFINE OMNI_CALLING_VMD "/VN=VN_OSAPCLTI"
  ```

The OmniView program is fully documented in the *VSI OMNI Guide to Using OmniView*.

Chapter 7. A Complete Example of a VSI OSAP Application

7.1. Introduction

This chapter describes how to configure a sample manufacturing plant consisting of two AP devices that are interconnected through the IEEE 802.3 network:

- A VAX processor running VSI OSAP for OpenVMS.
- An S5 Programmable Logical Controller (PLC) equipped with the CP 143 Communication Processor (or any other Siemens device that supports the Environment Management and Variable Access Services).

The chapter also includes the C language code of a VSI OSAP client application that establishes an association with the remote server application, and issues a Variable Access service request to read the value of a variable from the PLC (see Figure 7.1, “Association Between the VSI OSAP AP Client Application and the Remote AP Server”).

Figure 7.2, “Sample AP Manufacturing Plant Configuration” shows the static configuration of the sample manufacturing plant.

SYS1 and PLCREAD identify the OpenVMS system and the S5 PLC, respectively. Note that these names are not significant as Ethernet addresses.

Figure 7.1. Association Between the VSI OSAP AP Client Application and the Remote AP Server

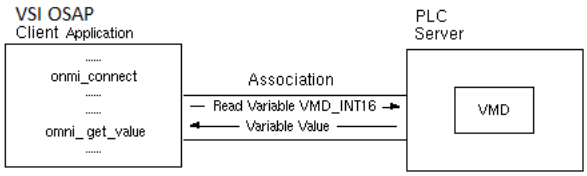
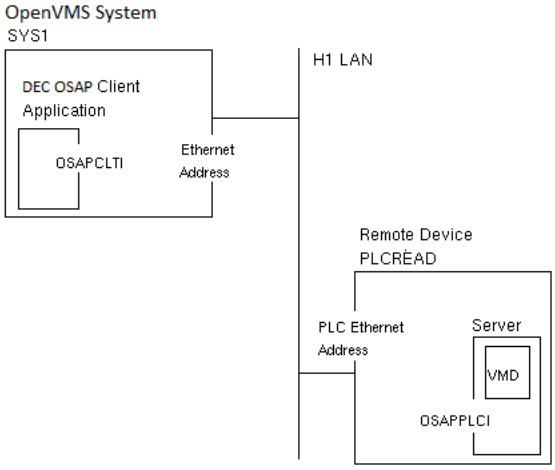


Figure 7.2. Sample AP Manufacturing Plant Configuration



OSAPCLTI and OSAPPLCI are the TSAP names that will be used at run time to set up the Transport Connection.

To configure the sample manufacturing plant, you must configure:

- The CP 143 Communication Processor on the PLC, using the COM 143 program.

- The VSI OSAP environment, using ODF commands and the omni_load_definition procedure, or ODSCL and the run-time object definition procedures.

7.2. PLC Configuration

This example assumes that you configure the CP 143 Communication Processor using a Siemens S5 programmer equipped with the COM143 configuration software.

Remember that a Communication Processor of the S5 series supports one VMD which does not require an explicit definition. Therefore, configuring the CP 143 Communication Processor means creating the following definitions:

- The variable with VMD scope that the VSI OSAP application will read at run time.
- The static characteristics of the AP association that the VSI OSAP client application will establish with the VMD server implemented by the CP 143 firmware.

7.2.1. Definition of the Variable with VMD Scope

Creating the definition of a variable with VMD scope on the CP 143 Communication Processor means creating an association between its logical name and the main memory address on the PLC that will contain the value of the variable.

This example assumes that the VSI OSAP application reads the value of a variable of Integer 16 type, and that has the logical name VMD_INT16.

Example 7.1, “The COM143 Variable Definition Display Form (AP)” shows how to use the COM143 variable definition display form to define the parameters of the VMD_INT16 variable.

Example 7.1. The COM143 Variable Definition Display Form (AP)

```
LOCAL DEFINITION SIMATIC S5 / COM 143
❶ NAME TYP ZGRF S5-ADRESSE ANZW SSNR
```

```
    VMD_INT32 IN 32 DB 100 10 MW 100 0
    VMD_INT16 IN 16 DB 100 14 MW 104 0
    VMD_INT8 IN 8 DB 100 16 MW 108 0
    VMD_VSI32 VS 32 DB 100 20 MW 112 0
```

❶ The COM143 variable definition display form allows definition of variables with VMD scope on the S5 PLC. A variable is defined by the following parameters:

- NAME

  The name of the variable (VMD_INT16). The VSI OSAP client application refers to this name when issuing Variable Access service requests to the remote VMD.

- TYP

  The type of the variable (IN 16, that is, 16-bit integer type).

- S5-ADRESSE

  The CPU memory address on the PLC where the value of the variable is stored (DB 100 14).

- ANZW

  The location of the status word in the CPU memory of the PLC (MW 104). The status word contains run-time status information related to the variable that is at disposal of the PLC program (for example, to know whether or not the remote partner application has modified the value of the variable).

7.2.2. Definition of the Static Characteristics of the Association

The CP 143 Communication Processor must contain the definition of the static characteristics of the association between the VSI OSAP client application and the VMD server (implemented by the CP 143 firmware).

Example 7.2, “The COM143 Static Characteristics Definition Display Form (AP)” shows how to use the COM143 static characteristics definition display form to define the parameters of the association.

Example 7.2. The COM143 Static Characteristics Definition Display Form (AP)

```
LINK SIMATIC S5 / COM143
_____________________________________________________________________
                                SSNR : 0 ANR : 1 ANZW : MW 10 ❶
LOCAL TSAP : - LENGTH : 8 HEX : 4F534150504C4349 ASC : OSAPPLCI ❷
EST TYPE (A4/A7/D4/D7/P4/P7) : P7 ❸ MUX ADDRESS : 0 ❹
LOCAL STF TITLE :
REMOTE TSAP : - LENGTH : 8 HEX : 4F534150434C5449 ASC : OSAPCLTI ❺
REMOTE ETHERNET ADDRESS : 08000601000121 ❻
NUMBER OF STF LINKS FOR CURRENT TSAP : 1 CURRENT JOB : 1
```

❶ The S5 PLC characteristics are defined by the following parameters:

- SSNR, which is the Interface number (0).
- ANR, which is the Job number (1).
- ANZW, which specifies the location of the status word in the CPU memory (MW 10) of the PLC. This parameter contains the status of the association at run time.

❷ OSAPPLCI is the name of the local TSAP, as defined on the CP 143 Communication Processor. The same name must be defined in the VSI OSAP AD_OSAPPLCI Application Descriptor (see Section 7.3, “VSI OSAP Environment Configuration”). This descriptor carries the addressing information necessary to locate the remote partner application acting as a server for the specified VMD.

❸ The value P7 assigned to the EST TYPE parameter indicates that when the VSI OSAP application invokes the omni_connect procedure, the requested association is established in the Application Layer. This means that an Initiate PDU must be exchanged between VSI OSAP and the remote partner in order to set up the association. Instead, the value P4 specifies that the requested association is established as soon as the Transport Connection has been set up.

If you specify P7, you must set to TRUE the NEGOTIATION attribute in the definition of the Application Descriptor (as shown in Section 7.3, “VSI OSAP Environment Configuration”) if you specify P4, you must set this attribute to FALSE.

❹ The MUX ADDRESS parameter is set to 0 to specify that the Transport Connection supports one association (no multiplexing).

❺ OSAPCLTI is the name of the remote TSAP. The CP 143 Communication Processor only accepts requests for set- ting up Transport Connections that carry OSAPCLTI as the calling TSAP and OSAPPLCI as the called TSAP, according to the values specified in Example 7.2, “The COM143 Static Characteristics Definition Display Form (AP)” This implies that the Application Descriptor associated with the VSI OSAP VMD definition (AD_OSAPCLTI in Section 7.3, “VSI OSAP Environment Configuration” must specify the OSAPCLTI TSAP. At run time, the VSI OSAP application must refer to this VMD definition as the calling VMD when invoking the omni_connect procedure. In this way, it passes the name of the calling TSAP to VSI OSAP.

❻ The REMOTE ETHERNET ADDRESS parameter specifies the Ethernet address of the calling partner. This address must be equal to the SYS1 system Ethernet address on the SINEC H1 LAN. You can set this parameter to zero.

7.3. VSI OSAP Environment Configuration

To configure the VSI OSAP environment on the OpenVMS system, you must create:

- An Application Descriptor for the VSI OSAP client application
- An Application Descriptor for the remote server application
- The definition of a local VMD
- The definition of a remote VMD

Example 7.3, “Application Descriptor for the VSI OSAP AP Client Application” shows how to create the Application Descriptor for the VSI OSAP client application, using the ODSCL REGISTER DIRECTORY command.

Example 7.3. Application Descriptor for the VSI OSAP AP Client Application

```
REGISTER DIRECTORY NAME "/CN=AD_OSAPCLTI" ATTRIBUTES "OC=OSAP
                                        /TSAP=OSAPCLTI
                                        /NETADDRESS=IEEE%AA000400DFB921
                                        /NEGOTIATION=TRUE"
```

IEEEA% is a template written to VOTS using the NCL command (for further information refer to the NCL documentation).

Example 7.4, “Application Descriptor for the Remote Server Application” shows how to create the Application Descriptor for the remote server application, using the ODSCL REGISTER DIRECTORY command.

Example 7.4. Application Descriptor for the Remote Server Application

```
REGISTER DIRECTORY NAME "/CN=AD_OSAPPLCI" ATTRIBUTES "OC=OSAP
                                        /TSAP=OSAPPLCI
                                        /NETADDRESS=IEEE%08000601000121
                                        /NEGOTIATION=TRUE"
```

The local VMD, the remote VMD and the VMD_INT16 variable are defined at run time using the VSI OMNI API procedures.

Part III. Working with SINEC-H1

Chapter 8. Configuration

8.1. Plant Object Definitions

The run-time object definition facility of the VSI OMNI API allows the creation of definitions for the H1 objects that represent the manufacturing plant. The VSI OMNI API procedures used to create run-time definitions of MMS objects are:

- omni_define
- omni_get_definition
- omni_modify_definition

They have been extended to allow the definition of:

- The message object
- Additional attributes for H1 objects that comply with MMS objects (such as VMD, Unnamed Variable and Domain).

Part IV, “Extension to VSI OMNI Procedure Calls” explains how to use the extended API procedures to create run-time object definitions. Alternatively, object definitions can be created using the Omni Definition Facility (ODF), and then loaded onto the application process space at run time.

This chapter explains how to use ODF to create local definitions of VSI OSAP objects representing the manufacturing plant. In addition to the commands that are documented in the *VSI OMNI Network Manager's Guide*, ODF provides a specific command for the definition of H1 message objects (messages do not exist in the MMS model).

This chapter explains how to use the *VSI OMNI Network Manager's Guide* and provides the additional information required to operate with ODF in the VSI OSAP environment.

8.1.1. ODF Definitions

The following sections explain how to use ODF to create definitions of H1 objects.

8.1.1.1. The OSH1 Application Profile

ODF has been designed to operate in both the MMS (default) and in the H1 contexts (that is, in the VSI OSAP context).

As explained in the *VSI OMNI Network Manager's Guide* , the MMS context is set by default when ODF is invoked.

Before using any other ODF command, you must issue the SET APPLICATION PROFILE command to set ODF command syntax and operation for VSI OSAP definitions created under ODF.

Setting the VSI OSAP context ensures that subsequent ODF commands are interpreted correctly, and the definitions created under ODF refer to the VSI OSAP environment.

Example 8.1, “How to Set the OSH1 Application Profile (H1)” shows how to invoke ODF and how to set the OSH1 application profile.

Example 8.1. How to Set the OSH1 Application Profile (H1)

```
$ ODF
ODF> SET APPLICATION PROFILE OSH1; ❶
```

```
ODF> SHOW APPLICATION PROFILE; ❷
        Application Profile is OSH1
ODF>
```

❶ After activating ODF, enter the SET APPLICATION PROFILE command, specifying the OSH1 attribute for the OSH1 application profile.

❷ You are now in the VSI OSAP context; this can be verified by issuing the SHOW APPLICATION PROFILE command.

8.1.1.2. Supported Definition Classes

ODF can be used to create definitions of the following H1 objects:

- 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

Refer to the *VSI OMNI Network Manager's Guide* for a general description of the commands used to create the definitions of the objects listed above. Read this chapter also for information on specific considerations on these commands when they are used to create object definitions in the VSI OSAP context. See Chapter 4, *ODF Command Extension Reference* of this manual for the reference to the DEFINE MESSAGE command.

8.1.1.3. Creating VMD Definitions

You create a local definition of a VMD using the ODF DEFINE VMD command. For a general description of this command, refer to the *VSI OMNI Network Manager's Guide*. Specific considerations on the VSI OSAP context are discussed in the sections below which contain the:

- List of attributes for the DEFINE VMD command that assume a specific significance in the VSI OSAP context
- Conformance Building Blocks supported by VSI OSAP
- List of the identifiers for the open and not-open H1 services supported by VSI OSAP.

Specific Attributes for the DEFINE VMD Command

Table 8.1, “Attributes of the DEFINE VMD Command that have a Specific Meaning in the VSI OSAP Context (H1)” contains a list and an explanation of the DEFINE VMD command attributes that assume a specific significance in the VSI OSAP context.

Table 8.1. Attributes of the DEFINE VMD Command that have a Specific Meaning in the VSI OSAP Context (H1)

Attribute	Explanation
APPLICATION SIMPLE NAME	Name of the Application Descriptor (AD) containing the information required by VSI OSAP to address a local or a remote application. ADs are created using ODSCL, as described in Chapter 3, *Configuration* of this manual.
PARAMETER CBB	The set of conformance building blocks supported by the VMD. See *the section called "Conformance Building Blocks"* for further details.
SUPPORTED SERVICES	The set of open Services supported by the user that calls the VMD. See *REFERENCE>(USER_3H1_SUPSERV)* for further details.
NOT OPEN SUPPORTED SERVICES	The set of not-open Services supported by the user that calls the VMD. See *the section called "VMD Supported Services"* for further details.

Any attributes that are not included in the table below, retain the same syntax and semantics in both the AP and MMS contexts.

Conformance Building Blocks

Table 8.2, "Supported Conformance Building Blocks (H1)" lists the Conformance Building Blocks (CBB) supported by VSI OSAP. The table also lists the values that can be specified in the PARAMETER CBB attribute of the DEFINE VMD command.

Table 8.2. Supported Conformance Building Blocks (H1)

PARAMETER CBB Value	ISO 9506 Designation
[NO]ARRAY	STR1
[NO]STRUCTURES	STR1
[NO]NAMED VARIABLES	VNAM
[NO]ALTERNATE ACCESS	VALT
[NO]UNNAMED VARIABLES	VADR

VMD Supported Services

Table 8.3, "Open H1 Service Identifiers" and Table 8.4, "Not Open H1 Service Identifiers" lists the open and not-open AP services that are supported by VSI OSAP, together with their identifiers. One or several AP service identifiers can be specified in the SUPPORTED SERVICES and NOT OPEN SUPPORTED SERVICES attributes of the DEFINE VMD command to indicate that the corresponding service is supported by the VMD being defined.

Table 8.3. Open H1 Service Identifiers

H1 Service	Service Identifier	Default Support
Environment Management		
Connect	(All VMD definitions support this service).	Yes
Cancel	CANCEL	No
Conclude	CONCLUDE	Yes
Abort	(All VMD definitions support this service).	Yes

H1 Service	Service Identifier	Default Support
VMD Support Services		
Status	STATUS	No
Variable Access Services		
Read	READ	Yes
Write	WRITE	Yes
Domain Management Services		
InitiateDownloadSequence	INITIATE DOWNLOAD SEQUENCE	Yes
DownloadSegment	DOWNLOAD SEGMENT	Yes
TerminateDownloadSequence	TERMINATE DOWNLOAD SEQUENCE	Yes
InitiateUploadSequence	INITIATE UPLOAD SEQUENCE	Yes
UploadSegment	UPLOAD SEGMENT	Yes
TerminateUploadSequence	TERMINATE UPLOAD SEQUENCE	Yes
Program Invocation Services		
Start	START	No
Stop	STOP	No

The DEFAULT SUPPORT column of the table indicates the AP services for which a default support is provided (Yes); if you want the VMD being defined supports these services, you can omit the corresponding service identifiers in the SUPPORTED SERVICES (or NOT OPEN SUPPORTED SERVICES) attribute.

Table 8.4. Not Open H1 Service Identifiers

H1 Service	Service Identifier	Default Support
Message Services		
Send	SEND	No
Receive	RECEIVE	No

Note

VSI OSAP ignores SUPPORTED SERVICES and NOT OPEN SUPPORTED SERVICES values different from those listed in Table 8.3, “Open H1 Service Identifiers” and Table 8.4, “Not Open H1 Service Identifiers” Any attempt made by an application to request the service will be rejected by the API at run time.

When defining a VMD, you must specify the supported services by assigning the appropriate values to the SUPPORTED SERVICES and NOT OPEN SUPPORTED SERVICES at- tributes. Before using the DEFINE VMD command to create a local definition of a remote VMD, collect accurate information on the model, characteristics and services supported by the remote device.

Example 8.2, “Creating the Local Definition of a Remote H1 VMD” shows how to create the local definition of a remote VMD. A VSI OSAP client uses this definition to establish an association, and to issue service requests to a remote VMD.

Example 8.2. Creating the Local Definition of a Remote H1 VMD

```
ODF>
ODF> DEFINE VMD OSH1PLC APP SIM NAM AD_OSH1PLC;
ODF>
```

The mnemonic name OSH1PLC identifies the local definition of the remote VMD.

AD_OSH1PLC is the name of the Application Descriptor (AD) that contains the addressing information related to the remote application acting as a server for the remote VMD. The ADs contain information that enable the VSI OSAP Client application to access the remote VMD on the DEV1 remote device at run-time. A full description of how to create and operate ADs is given in Chapter 3, *Configuration*.

Example 8.2, “Creating the Local Definition of a Remote H1 VMD” shows that, apart from the specified parameters, all the default parameters have been accepted.

Example 8.3, “Creating the Local Definition of a Local H1 VMD” shows how to create the local definition of a local VMD. A VSI OSAP application uses this definition to declare itself to VSI OSAP as the server application, and thus to accept service requests from a remote client application.

Example 8.3. Creating the Local Definition of a Local H1 VMD

```
ODF>
ODF> DEFINE VMD OSH1CLT APP SIM NAM AD_OSH1CLT;
ODF>
```

OSH1CLT is the mnemonic name that identifies the local definition of a VMD residing on the SYS1 VSI OSAP system.

AD_OSH1CLT is the name of the Application Descriptor that contains the addressing information associated to the VSI OSAP application acting as a server for the local VMD.

Example 8.3, “Creating the Local Definition of a Local H1 VMD” shows that, apart from the specified parameters, all the default parameters have been accepted.

8.1.1.4. Creating Unnamed Variable Definitions

When using the DEFINE UNNAMED VARIABLE command to create the definition of an unnamed variable associated with the local definition of a local or remote VMD, the address attribute has the following format:

```
<address type><address string>
```

where:

- address type must be set to UNCONSTRAINED ADDRESS
- address string must be set to any valid user-defined string, as explained below.

A client application that wishes to read or write the value of an unnamed variable must specifies the values listed above.

The definitions associated with unnamed variables defined on remote H1 devices must specify a valid product-dependent address. Refer to the Siemens product documentation of the specific H1 manufacturing device for information on how to formulate a valid address. However, the syntax of the address string attribute requires a VSI OSAP-specific value to specify one of the following addresses:

- Numeric Address format
- Reduced Unconstrained Address format

Numeric Address Format

In SINEC-H1, the Numeric Address format is an internal address information related to a Named Variable defined on the device's Communication Processor. It can be obtained through a GetVariableAccessAttributes service request for the specified named variable. At this point, you can define an unnamed variable by specifying the same type definition as for a named variable and the numeric address already obtained. It is now possible to access the device variable as either a named or an unnamed variable; the latter type of access is faster than the former.

Reduced Unconstrained Address Format

The address of a memory location on the H1 device containing the VMD to which the variable is associated. Valid address strings have the following format:

```
"BlockType:BlockNumber:StartingAddress"
```

BlockType can assume any of the following values: DB, FY, IB, QB, PY, CB, TB, RS, AS, DX, DE, OY, RDB, RDX, PDB and PDX.

The defined string must be recorded in an omni_t_address_str word counted structure.

The Reduced Unconstrained Address format can be used for PLC devices only.

Note

The Extended Unconstrained Address format is not supported in SINEC-H1.

Local Definition of a Local Unnamed Variable

The definitions associated with Unnamed Variables defined on an OpenVMS system must specify a valid product-dependent address. Valid address strings have the following format:

```
"BlockType:BlockNumber:StartingAddress"
```

BlockType can assume any of the following values: DB, FY, IB, QB, PY, CB, TB, RS, AS, DX, DE, OY, RDB, RDX, PDB and PDX.

The defined string must be recorded in an omni_t_address_str word counted structure.

8.1.1.5. Creating Message Definitions

H1 messages are the objects referred to by the non-open Serial Transfer Services.

Using H1 messages, partner applications can read, write and exchange strings of bytes. (Applications themselves are in charge of preparing and interpreting the message contents.)

A message is an object specific to H1 (it is not part of the MMS model) and its definition is created using the ODF DEFINE MESSAGE command. Only one message object definition can be associated with each VMD created in the VSI OSAP context.

Even though the DEFINE MESSAGE command respects the syntax and operation of the VSI OMNI ODF, it can only be used in the VSI OSAP context. Therefore, how to create a message object definition is fully explained in the sections below, and the DEFINE MESSAGE command reference is documented in Chapter 4, *ODF Command Extension Reference* of this manual.

A message definition includes the following attributes:

- Name of the message.
- Description of the message.
- Length of the message.

Example 8.4, “Creating an H1 Message Definition” shows how to create a message definition.

Example 8.4. Creating an H1 Message Definition

```
ODF> ❶
```

```
ODF> DEFINE MESSAGE VMD1DEV1:MY_MSG, LENGTH 300, ❷
        DESCRIPTION "Used in subsequent examples";
ODF>
```

❶ Make sure that the OSH1 Application Profile has been set, by entering the SHOW AP command; the system should respond with the message "Application Profile is OSH1".

❶ The LENGTH attribute (300) specifies the maximum length of the message object value (a string of bytes).

A VSI OSAP application can now use the message object definition MY_MSG to write, read, or exchange an H1 message with the remote H1 device represented by the VMD1DEV1 VMD.

8.1.1.6. Creating Domain Definitions

When using the DEFINE DOMAIN command to create the definition of a Domain, you must specify the BLOCK ADDRESS LIST attribute, which is specific to H1. You must supply a list of elements, and each element has the following format:

```
"BlockType:BlockNumber"
```

BlockType can assume any of the following values: DB, FY, IB, QB, PY, CB, TB, RS, AS, DX, DE, OY, RDB, RDX, PDB and PDX.

Each element in the list must be separated from the next one by means of a comma.

8.2. Application Descriptors

For the transparent management of an association between a VSI OSAP application and a remote application, create two Application Descriptors (one for the VSI OSAP application and the other for the remote partner), and provide a link between each AD and the definition of the (local or remote) VMD.

To create an Application Descriptor, create an entry in the OMNI Directory Services using the OMNI Directory Service Control Language (ODSCL).

To provide the link, insert the name of the AD in the Common Name attribute of the local (or remote) VMD definition.

At run-time, when a VSI OSAP application requests an association to be established with a remote VMD, it specifies the name of the local definition of the remote VMD. This carries the name of the AD where VSI OSAP finds the information to address the remote application (such as the NETADDRESS of the remote device and the TSAP of the remote application), in order to establish the requested association.

In the H1 environment, an association is not just a general purpose logical link above a Transport Connection like it is in the AP protocol; in fact, the SINEC-H1 protocol needs different Transport Connections to perform different services:

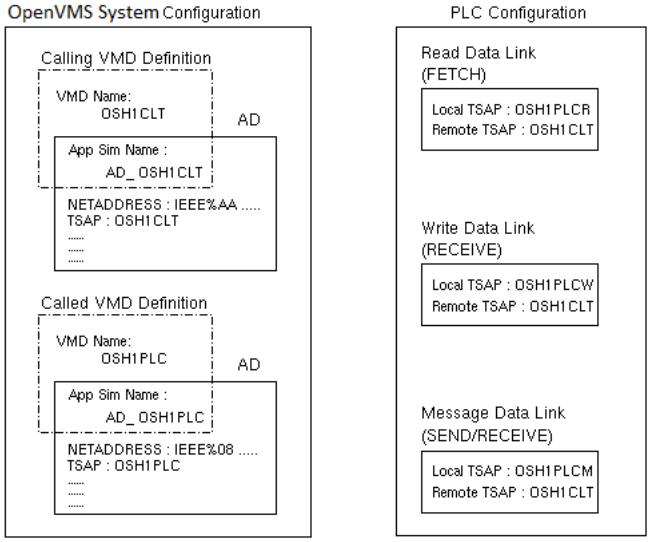
- One Transport connection for reading Variables only.
- One Transport connection for writing Variables only.
- One Transport connection for sending, receiving Messages (or both).
- One Transport connection for Device management operations (Start, Stop, Upload and so on).

VSI OSAP sets up all the above Transport Connections automatically when the user application requests an H1 association to be established. In particular, the first three connections are "static", that is they are established and maintained for all the life of the association. The fourth association listed above is dynamic, that is, it is only

established when a Device Management service is requested, and it is concluded on completion of the requested service.

By default, only the Read and Write static connections are established underneath an H1 association. Figure 8.1, “An AD for a Called H1 Application” shows how to enable the Message support and how to disable the Read support, the Write support, or both.

Figure 8.1. An AD for a Called H1 Application



8.2.1. Application Descriptor Data for a Called Application

A called application is the application that in the association establishment phase acts as the responder. This is the application that is waiting for an association request coming from the peer application (the calling application).

The definition of an AD for a remote partner application contains addressing information which is valid for a remote server or a remote client. For example, when you create an AD for a remote server application, (see also Figure 8.1, “An AD for a Called H1 Application”), you must specify:

- The NETADDRESS of the calling H1 system where the server of the remote VMD is active.
- The called TSAP name associated with the remote VMD; actually, there can be one, two or three TSAPs, depending on the services that are enabled for the remote VMD. To avoid the creation of three ADs, the following conventions have been adopted:
  - The TSAP configured in the AD represents just a common leading part for the three TSAPs (for example, OSH1PLC).
  - The TSAP for the Read connection is automatically built using the common part plus the R character (for example, OSH1PLCR).
  - The TSAP for the Write connection is automatically built using the common part plus the W character (for example, OSH1PLCW).
  - The TSAP for the Message connection is automatically built using the common part plus the M character (for example, OSH1PLCM).

  These three TSAPs must be configured explicitly on the PLC in the corresponding Data Link Blocks as either local or remote TSAPs, according to the role of the PLC in that specific association.

  The Device Management connection uses a hidden fixed TSAP that does not need to be configured.

- The attributes for the management of the association: COMPLTIMER and RESPTIMER (see also Table 8.5, “Attributes of ODSCL Command Arguments (H1)”). These attributes refer to requests that flow from the VSI OSAP client application to the remote server application.

Table 8.5. Attributes of ODSCL Command Arguments (H1)

Attribute	Description
/TSAP= *value*	Transport Service Access Point for the local or remote application. It must be seven characters in length and must have the same value as that defined for the partner device.
/NETADDRESS= *value*	It is a name of maximum 64 alphanumeric characters that corresponds to the Network Address of the device on which the application being defined is active.
/COMPLTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse between the request for a service and its completion. The API returns an error to the VSI OSAP application if the time expires before a response is received. If COMPLTIMER is set to 0, the time interval is infinite (that is, timer disabled). Default is 0 seconds.
/RESPTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse before a service request is forwarded again, if no response is received. The value of this parameter is usually a submultiple of the value assigned to COMPLTIMER. A service request is forwarded once if RESPTIMER is set to 0 (that is, timer disabled). Default is 0 seconds.

8.2.2. Application Descriptor Data for a Calling Application

A calling application is the application that in the association establishment phase acts as the requester, that is the application that sends an association request to the peer application (the called application).

The definition of an AD for a VSI OSAP application contains addressing information which is valid for a local server or a local client. For example, when you create an AD for a VSI OSAP server application you must specify:

- The NETADDRESS of the calling H1 system, which also specifies the Ethernet address of the calling H1 device on the IEEE 802.3 LAN.
- The calling TSAP name associated with the calling VMD; this TSAP will be used in all the three Transport Connections.

  This TSAP must be configured on the PLC in the corresponding Data Link Blocks as either local or remote TSAP, according to the role of the PLC in that specific association.

  The Device Management connection uses a hidden fixed TSAP that does not need to be configured.
- The attributes for the management of the association: COMPLTIMER and RESPTIMER. These attributes refer to requests that flow from the VSI OSAP client application to the remote server application.

8.3. ODSCL Commands to Create Application Descriptors

The ODSCL command REGISTER DIRECTORY is used to create an Application Descriptor. It is also possible to:

- Remove an AD (DEREGISTER DIRECTORY command).

- Modify an AD (MODIFY DIRECTORY command).
- Display all AD attributes (READ DIRECTORY command).
- Display ADs in the directory matching the selection criteria specified or all entries in the directory (LIST DIRECTORY command).
- Establish the current name directory path used to refer to the AD based on its relative position in the name hierarchy (SET command), and display the current name directory path (SHOW command).

General information on ODSCL commands (including command syntax, ODSCL Command Language Interface, invoking and exiting ODSCL) is given in the *VSI OMNI Network Manager's Guide* .

Tables Table 8.6, “Arguments of ODSCL Commands (H1)” and Table 8.7, “Attributes of ODSCL Command Arguments (H1)” show the VSI OSAP-specific arguments and attributes that must be specified when using ODSCL commands to create Application Descriptors.

Table 8.6. Arguments of ODSCL Commands (H1)

Argument	Description
NAME "/CN= *simple_ name* "	Identifies the name of the Application Descriptor to be created. The simple name of an application can consist of up to 64 characters, which must come from the printable string character set. /CN is the name attribute abbreviation as defined in the known attributes types file.
ATTRIBUTES "OC=OSH1 / *attr_abbrev=value...* "	The set of attributes and the attribute values listed in Table 8.7, “Attributes of ODSCL Command Arguments (H1)”. OC is the object class abbreviation as defined in the known object classes file.

Table 8.7. Attributes of ODSCL Command Arguments (H1)

Attribute	Description
/TSAP= *value*	Transport Service Access Point for the local or remote application. It must be seven characters in length and must have the same value as that defined for the partner device.
/NETADDRESS= *value*	It is a name of maximum 64 alphanumeric characters that corresponds to the Network Address of the device on which the application being defined is active.
/COMPLTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse between the request for a service and its completion. The API returns an error to the VSI OSAP application if the time expires before a response is received. If COMPLTIMER is set to 0, the time interval is infinite (that is, timer disabled). Default is 0 seconds.
/RESPTIMER= *value*	Indicates the maximum time interval (in seconds) that can elapse before a service request is forwarded again, if no response is received. The value of this parameter is usually a submultiple of the value assigned to COMPLTIMER. A service request is forwarded once if RESPTIMER is set to 0 (that is, timer disabled). Default is 0 seconds.

8.3.1. Examples

The following examples show how to use the REGISTER DIRECTORY command to create an AD for a remote server application and an AD for a VSI OSAP server application.

Example 8.5, “Creating an AD for a Called H1 Application” shows how to create an AD that defines the addressing information for a remote server application (that is, the server that owns a VMD mapping resources and functions of a remote AP device).

Example 8.5. Creating an AD for a Called H1 Application

```
REGISTER DIRECTORY NAME "/CN=AD_OSH1PLC" ATTRIBUTES
                                         "OC=OSH1
                                         /NETADDRESS=IEEE%080006010001
                                         /TSAP=OSH1PLC"
```

It is assumed that "IEEE" is the Template Name defined in NCL for Null Internet support under the OSI Transport.

Example 8.6, “Modifying an AD (H1)” shows how to use the MODIFY DIRECTORY command to replace the TSAP value (OSH1PLC) with a new value (OSH1SRV).

Example 8.6. Modifying an AD (H1)

```
MODIFY DIRECTORY NAME "/CN=AD_OSH1PLC" REPLACE
                                      "TSAP=OSH1PLC,OSH1SRV"
```

Chapter 9. Programming with VSI OSAP

This chapter describes how to use the VSI OMNI API procedures to write VSI OSAP applications.

9.1. Introduction

Programming in the VSI OSAP environment means preparing applications that use the VSI OMNI API procedures to communicate with partner applications running on SINEC-H1 devices.

Bear in mind the following considerations when using API procedures to write VSI OSAP applications:

- General syntax and programming rules, described in *VSI OMNI Application Programmer's Guide* , are also valid when the API procedures are used in the VSI OSAP environment.
- Open H1 services are a functional subset of the MMS services implemented by the VSI OMNI API. Therefore, when reading the *VSI OMNI Application Programmer's Guide*, ignore those descriptions referring to services and functions that are not supported by the VSI OSAP implementation of H1.

  The VSI OMNI API procedures that can be used to write VSI OSAP applications are listed in Table 9.1, “API Procedures that Can be Invoked by a VSI OSAP H1 Application”.

Table 9.1. API Procedures that Can be Invoked by a VSI OSAP H1 Application

Procedure Name	**H1 Service Class/ Local VSI OMNI Operation**
omni_abort	Environment Management
omni_accept_connect	Environment Management
omni_cancel	Local VSI OMNI Operation
omni_conclude	Environment Management
omni_connect	Environment Management
omni_define	Local VSI OMNI Operation
omni_download	Domain Management
omni_end_list	Local VSI OMNI Operation
omni_get_attribute	Local VSI OMNI Operation
omni_get_definition	Local VSI OMNI Operation
omni_get_handle_by_name	Local VSI OMNI Operation
omni_get_handle_list	Local VSI OMNI Operation
omni_get_indications	Local VSI OMNI Operation
omni_get_message_text	Local VSI OMNI Operation
omni_get_remote_attributes	VMD Support
omni_get_value	Variable Access and Message Transfer
omni_group_variables	Local VSI OMNI Operation
omni_initialize	Local VSI OMNI Operation
omni_listen	Environment Management
omni_load_definitions	Local VSI OMNI Operation
omni_modify_definition	Local VSI OMNI Operation
omni_poll	Local VSI OMNI Operation

Procedure Name	H1 Service Class/ Local VSI OMNI Operation
omni_print_message	Local VSI OMNI Operation
omni_put_value	Variable Access and Message Transfer
omni_reject	Local VSI OMNI Operation
omni_reject_connect	Environment Management
omni_send_value	VMD Support, Variable Access and Message Transfer
omni_set_application_profile	Local VSI OMNI Operation
omni_start	Program Invocation
omni_stop	Program Invocation
omni_terminate	Local VSI OMNI Operation
omni_upload	Domain Management

- The extensions of the VSI OMNI API procedures that support the H1 message services is given in Chapter 12, *Extension to VSI OMNI Procedure Calls* of this manual. Extension means that some procedure at- tributes require or return values that are related to the services of the message service class.

Before a VSI OSAP application can be prepared, the following prerequisites must be met:

- The VSI OSAP environment must have been configured on the OpenVMS system according to the instructions given in Chapter 8, *Configuration*.
- You should have a basic knowledge of the VSI OMNI API procedures. Introductory chapters of the *VSI OMNI Application Programmer's Guide* and this chapter provide the information you require to write VSI OSAP applications.

Chapter 11, *A Complete Example of a VSI OSAP Application* contains an example that shows how to write a C Language VSI OSAP application.

9.2. User Include Files

In addition to the VSI OMNI files (omni_defs. *lang* and omni_ codes. *lang*), VSI OSAP provides the following file:

- osh1_codes. *lang*

  Include file containing VSI OSAP-specific completion and error codes. Found in SYS$LIBRARY.

  VSI OSAP-specific definitions are automatically available through omni_defs. *lang*.

9.3. H1 Services and API Procedures

Table 9.1, “API Procedures that Can be Invoked by a VSI OSAP H1 Application” lists the API procedures that can be invoked by a VSI OSAP application.

Note

VSI OMNI API procedures not listed in Table 9.1, “API Procedures that Can be Invoked by a VSI OSAP H1 Application” should be ignored.

9.3.1. Mapping H1 Client Service Request into API Procedures

Table 9.2, “Invoking API Procedures to Issue H1 Client Requests” shows the API procedures that a VSI OSAP application calls to issue client requests.

Table 9.2. Invoking API Procedures to Issue H1 Client Requests

H1 Service	API Procedure Call
Environment Management Services	
Connect	omni_connect
Conclude	omni_conclude
Abort	omni_abort
VMD Support Services	
Status	omni_get_remote_attributes specifying a VMD and omni_c_attr_all, followed by omni_get_attribute
Variable Access Services	
Read	omni_get_value
Write	omni_put_value
Domain Management Services	
InitiateDownloadSequence, DownloadSegment, TerminateDownloadSegment	omni_download
InitiateUploadSequence, UploadSegment, TerminateUploadSequence	omni_upload
Program Invocation Services	
Start	omni_start
Stop	omni_stop
Message Services	
Receive	omni_get_indications + omni_get_value

9.3.2. Mapping H1 Server Service Request into API Procedures

Table 9.3, “Invoking API Procedures to Fulfill H1 Server Requests” shows the API procedures that a VSI OSAP application calls to fulfill server requests.

Table 9.3. Invoking API Procedures to Fulfill H1 Server Requests

H1 Service	API Procedure Call
Environment Management	
Connect	omni_listen + omni_accept_connect (or omni_reject_connect)
Conclude	omni_get_indications
Abort	omni_get_indications
VMD Support Services	
Status	omni_modify_definition specifying omni_c_attr_logical_status, or omni_c_attr_physical_status, or both
Variable Access Services	
Write	omni_get_indications + omni_get_value
Read	omni_get_indications + omni_put_value
Message Services	
Send	omni_send_value

9.4. The OSH1 Application Profile

A VSI OSAP application invokes the omni_set_application_ profile procedure to specify the OSH1 application profile. This ensures that definitions created after the call to omni_set_ application_profile are assumed to be associated with the specified profile.

The following example shows how to invoke the omni_set_ application_profile to set the OSH1 application profile.

```
status = omni_set_application_profile (omni_c_app_profile_osh1)
```

9.5. Run-time Definitions

The API procedures used to create definitions at run-time are omni_define, omni_get_definition and omni_modify_ definition.

9.5.1. Supported Definition Classes

The VSI OMNI run-time object definition facility can be used to create definitions of the following H1 objects:

- VMD
- Domain
- Program Invocation
- Unnamed Variable
- Message
- MMS Named Type
- Application Named Type
- MMS Type Specification
- Application Type Specification
- MMS Structure Component
- Application Structure Component

9.5.2. Creating VMD Definitions

The following VMD attribute is specific to VMD definitions created in the VSI OSAP context using the omni_modify_ definition procedure:

- omni_osh1_c_attr_notopen_srvspt

The set of non-open H1 services supported by the user that calls the VMD.

9.5.2.1. Considerations on VMD Supported Services

VSI OSAP ignores unsupported VMD services. Any attempt made by an application to request VMD services not supported by VSI OSAP is rejected by the API at run time.

The open H1 services supported by the VMD are specified by assigning the appropriate values to the omni_c_attr_ vmd_srv_supported attribute of the omni_modify_definition procedure.

The VSI OSAP application uses the omni_c_attr_vmd_srv_ supported and omni_osh1_c_attr_notopen_srvspt attributes to inform the remote application of the supported services. VSI OSAP rejects any received indications for services not supported, without notifying the application. These attributes have a further purpose in H1, related to the association establishment phase: the three static Transport Connections that are underneath an association

are established or not depending on whether or not the corresponding services (Read Variable, Write Variable, Send/Receive Message) are enabled. By default, only the Read and Write Variable services are enabled, so only the two corresponding Transport connections are established. Refer to Chapter 11, *A Complete Example of a VSI OSAP Application* for further details.

9.5.2.2. Model Attribute for H1 VMDs

When you create a VMD definition you must specify a value for the model attribute (omni_c_attr_model). You can specify one of the following values:

- SIMATIC_115U-941
- SIMATIC_115U-942
- SIMATIC_115U-943
- SIMATIC_115U-944
- SIMATIC_135U-921
- SIMATIC_135U-922
- SIMATIC_135U-928
- SIMATIC_150U
- SIMATIC_155U

9.5.2.3. Defaults for VMD Attributes

The following are defaults for omni_c_attr_vmd_param_ supported, omni_c_attr_vmd_srv_supported and omni_osh1_c_attr_ notopen_srvspt.

Default for omni_c_attr_vmd_param_supported:

```
  (0) STR1 (Arrays) TRUE
  (1) STR2 (Structures) TRUE
  (2) VNAM (Named Variables) FALSE
  (3) VALT (Alternate Access) FALSE
  (4) VADR (Unnamed Variables) TRUE
  (5) VSCA (Scattered Access) FALSE
  (6) TPY (Third Party) FALSE
  (7) VLIS (Named Variable List) FALSE
  (8) REAL (Real) FALSE
  (9) AKEC (Acknowledgment Event Conditions) FALSE
 (10) CEI (Evaluation Interval) FALSE
```

Default for omni_c_attr_srv_supported:

```
        /* vmd support services */
  (0) Status FALSE
  (1) GetNameList FALSE
  (2) Identify FALSE
  (3) Rename FALSE
        /* variable access services */
  (4) Read TRUE
  (5) Write TRUE
  (6) GetVariableAccessAttributes FALSE
  (7) DefineNamedVariable FALSE
  (8) DefineScatteredAccess FALSE
  (9) GetScatteredAccessAttributes FALSE
 (10) DeleteVariableAccess FALSE
```

```
(11) DefineNamedVariableList FALSE
(12) GetNamedVariableListAttributes FALSE
(13) DeleteNamedVariableList FALSE
(14) DefineNamedType FALSE
(15) GetNamedTypeAttributes FALSE
(16) DeleteNamedType FALSE
        /* operator communication services */
(17) Input FALSE
(18) Output FALSE
        /* semaphore management services */
(19) TakeControl FALSE
(20) RelinquishControl FALSE
(21) DefineSemaphore FALSE
(22) DeleteSemaphore FALSE
(23) ReportSemaphoreStatus FALSE
(24) ReportPoolSemaphoreStatus FALSE
(25) ReportSemaphoreEntryStatus FALSE
        /* domain management services */
(26) InitiateDownloadSequence FALSE
(27) DownloadSegment TRUE
(28) TerminateDownloadSequence TRUE
(29) InitiateUploadSequence FALSE
(30) UploadSegment FALSE
(31) TerminateUploadSequence FALSE
(32) RequestDomainDownload FALSE
(33) RequestDomainUpload FALSE
(34) LoadDomainContent FALSE
(35) StoreDomainContent FALSE
(36) DeleteDomain FALSE
(37) GetDomainAttributes FALSE
        /* program invocation management services */
(38) CreateProgramInvocation FALSE
(39) DeleteProgramInvocation FALSE
(40) Start FALSE
(41) Stop FALSE
(42) Resume FALSE
(43) Reset FALSE
(44) Kill FALSE
(45) GetProgramInvocationAttributes FALSE
        /* file management services */
(46) ObtainFile FALSE

        /* event management services */
(47) DefineEventCondition FALSE
(48) DeleteEventCondition FALSE
(49) GetEventConditionAttributes FALSE
(50) ReportEventConditionStatus FALSE
(51) AlterEventConditionmonitoring FALSE
(52) TriggerEvent FALSE
(53) DefineEventAction FALSE
(54) DeleteEventAction FALSE
(55) GetEventActionAttributes FALSE
(56) ReportEventActionStatus FALSE
(57) DefineEventEnrollment FALSE
(58) DeleteEventEnrollment FALSE
(59) AlterEventEnrollment FALSE
(60) ReportEventEnrollmentStatus FALSE
(61) GetEventEnrollmentAttributes FALSE
```

```
(62) AcknowledgeEventNotification FALSE
(63) GetAlarmSummary FALSE
(64) GetAlarmEnrollmentSummary FALSE
       /* journal management services */
(65) ReadJournal FALSE
(66) WriteJournal FALSE
(67) InitializeJournal FALSE
(68) ReportJournalStatus FALSE
(69) CreateJournal FALSE
(70) DeleteJournal FALSE
       /* vmd support services */
(71) GetCapabilityList FALSE
       /* file management services */
(72) FileOpen FALSE
(73) FileRead FALSE
(74) FileClose FALSE
(75) FileRename FALSE
(76) FileDelete FALSE
(77) FileDirectory FALSE
       /* unconfirmed services */
(78) UnsolicitedStatus TRUE
(79) InformationReport TRUE
(80) EventNotification FALSE
(81) AttachToEventCondition FALSE
(82) AttachToSemaphore FALSE
       /* additional services */
(83) Conclude TRUE
(84) Cancel FALSE
```

Default for omni_osh1_c_attr_vmd_notopen_srvspt:

```
       /* message services */
  (0) Send FALSE
```

9.5.3. Creating Domain Definitions

The Domain object is not defined in SINEC-H1, but it can be emulated by defining it as a collection of PLC's memory blocks. The omni_osh1_c_attr_block_addr_list multi-valued attribute is used to emulate a Domain.

If you do not specify this attribute when you create a Domain definition, the Domain represents the entire PLC memory.

If you want that the Domain be a portion of the PLC memory, you must specify this attribute and supply a list of values to identify the addresses of the PLC memory blocks that must be part of the Domain. Each value in the list must be a string having the following format:

```
"BlockType:BlockNumber"
```

Where:

- Block Type Can assume any of the following values: DB, FB, OB, SB, DX, FX and PB.
- Block Number Is an integer in the range 0 to 255.

The following is an example of a valid value.

```
"DB:100"
```

The following Domain attributes are not used in the SINEC- H1 context:

- omni_c_attr_capability_file
- omni_c_attr_deletable
- omni_c_attr_sharable

9.5.4. Creating Program Invocation Definitions

The following Program Invocation attributes are not used in the SINEC-H1 context:

- omni_c_attr_exec_arg
- omni_c_attr_deletable
- omni_c_attr_monitor

9.5.5. Creating Unnamed Variable Definitions

When using the omni_modify_definition procedure to create the definition of an unnamed variable associated with the local definition of a local or remote VMD, you must set:

- The address type attribute (identified by the omni_c_ attr_address_type constant) to the omni_c_address_ unconstrained value.
- The address string attribute (identified by the omni_c_ attr_address_string constant) to any valid user-defined string, as explained in the following two sections.

A client application that wishes to read or write the value of an unnamed variable must specify the values listed above.

Local Definition of a Remote Unnamed Variable

The definitions associated with unnamed variables defined on remote H1 PLC devices must specify a valid product-dependent address. Refer to the Siemens product documentation of the specific S5 PLC device for information on how to formulate a valid address. However, the syntax of the ad- dress string attribute requires a VSI OSAP-specific value to indicate the following address:

- Reduced Unconstrained Address format

  The address of a memory location on the H1 device containing the VMD to which the variable is associated. Valid address strings have the following format:

  ```
  "BlockType:BlockNumber:StartingAddress"
  ```

  BlockType can assume any of the following values: DB, FY, IB, QB, PY, CB, TB, RS, AS, DX, DE and OY.

  The defined string must be recorded in an omni_t_ address_str word counted structure.

Local Definition of a Local Unnamed Variable

The definitions associated with Unnamed Variables defined on an OpenVMS system must specify a valid product-dependent address. Valid address strings have the following format:

```
"BlockType:BlockNumber:StartingAddress"
```

BlockType can assume any of the following values: DB, FY, IB, QB, PY, CB, TB, RS, AS, DX, DE, OY, RDB, RDX, PDB and PDX.

The defined string must be recorded in an omni_t_address_ str word counted structure.

The following Unnamed Variable attribute is not used in the SINEC-H1 context:

- omni_c_attr_address_number

9.5.6. Creating MMS Named Type Definitions

The following MMS Named Type attribute is not used in the SINEC-H1 context:

- omni_c_attr_deletable

9.5.7. Creating Message Definitions

Messages are the objects referred to by the non-open Serial Transfer Services.

Using messages, partner applications can read, write and exchange strings of bytes. (Applications themselves are responsible for preparing and interpreting the message contents.)

Although a message is not part of the MMS model, its definition is created and modified using the omni_define and omni_modify_definition procedures.

Only one message object definition can be associated with each VMD definition created in the VSI OSAP context.

A message definition includes the following attributes:

- Name of the message
- Description of the message
- Length of the message
- Handle of a VMD definition

Example 9.1, “Creating a Run-Time H1 Message Definition” shows how to create a message object definition.

Example 9.1. Creating a Run-Time H1 Message Definition

```
      .
      .
      .
      /* Create a Message Definition */
      status = omni_define (omni_c_cls_msg, &H_TargetMessage);
      .
      .
      .
      /* Modify Message Attributes */
      Attribute = omni_c_attr_name;
      Context = 0;
      status = omni_modify_definition (H_TargetMessage, &Attribute,
                                                        &Context, "MY_MSG",
 strlen("MYMSG"));
      Attribute = omni_c_attr_length;
      msg_length = 300;
      Context = 0;
      status = omni_modify_definition (H_TargetMessage, &Attribute,
                                                        &Context,
 msg_length,

 sizeof(msg_length));
      Attribute = omni_c_attr_description;
```

```
     Context = 0;
     status = omni_modify_definition (H_TargetMessage, &Attribute,
&Context,
                                                    "Used in subsequent
examples");
     Attribute = omni_c_attr_scope;
     Context = 0;
     status = omni_modify_definition (H_TargetMessage, &Attribute,
                                                    &Context,
&H_TargetVMD,

sizeof(omni_l_handle));
     .
     .
     .
```

For the procedure references, see Chapter 12, *Extension to VSI OMNI Procedure Calls*.

The name of the definition, the length and description attributes are set. The length attribute (300) specifies the maximum length of the message object value (a string of bytes).

The message is committed (that is, associated) to the VMD1DEV1 VMD that was created in Example 5.1, “Creating the Run-Time Local Definition of a Remote AP VMD”.

A VSI OSAP application can now use the message object definition MY_MSG to write, read, or exchange an H1 message with the remote AP device represented by the VMD1DEV1 VMD.

9.6. Message Services

A VSI OSAP client application can ask a remote server application for the following Message Service:

- Receive a message from the remote partner.

A VSI OSAP server application can:

- Send a message to the remote partner.

The following sections explain how a VSI OSAP application sends and receives messages using the API procedures.

9.6.1. Sending a Message

To send a message to a remote server application active on an H1 device, a VSI OSAP client application invokes the omni_ send_value procedure and supplies the handle of the message definition.

Data consist of strings of bytes, where each string has a maximum length of 8 192 bytes. The meaning of the received data is defined by the two partner applications.

Example 9.2, “A VSI OSAP Application Sends an H1 Message” shows how a VSI OSAP application receives a message from, and sends a message to, the remote server application.

Example 9.2. A VSI OSAP Application Sends an H1 Message

```
main ()
{
    struct
    {
          unsigned short Length;
          char Data[100];
```

```
    } MsgDsc;
    omni_l_handle remote_vmd;
    omni_l_handle msg_handle;
    omni_r_iosb omni_iosb;
    .
    .
    .
    strcpy (MsgDsc.Data, "This is my message");
    MsgDsc.Length = 100;
    status = omni_send_value (remote_vmd,
                                          msg_handle,
                                          0,
                                          &MsgDsc,
                                          sizeof(MsgDsc),
                                          &omni_iosb,
                                          0);
}
    .
    .
    .
```

9.6.2. Receiving a Message

A VSI OSAP client application invokes the omni_get_ indications procedure to receive a message indication.

To get the message value, the application invokes the omni_ get_value procedure (see Example 9.3, “A VSI OSAP Application Receives an H1 Message”).

Example 9.3. A VSI OSAP Application Receives an H1 Message

```
main ()
{
    struct
    {
          unsigned short Length;
          char Data[100];
    } MsgDsc;
    omni_r_iosb omni_iosb;
    omni_l_context ctx;
    omni_l_indications code;
    omni_l_handle remote_vmd;
.
.
.
/*
  * Get indications.
  */
      status = omni_get_indications (remote_vmd,
                                                  &obj,
                                                  &ctx,
                                                  &code,
                                                  0,
                       &omni_iosb,
                             0);
      switch (code)
      {
          case omni_c_ind_info_report:
    /*
```

```
       * Get the message.
       */
               MsgDsc.Length = 100;
               status = omni_get_value (0,
                                                  ctx,
                                                  0,
                                                  &MsgDsc,
                                                  sizeof(MsgDsc),
                                                  0,
                                                  &omni_iosb,
                                                  0);
               printf ("Message Length : %d \n", MsgDsc.Length);
               printf ("Received Message: %s \n", MsgDsc.Data);
               break;
.
.
.
    }
}
```

Chapter 10. Monitoring the VSI OSAP Environment

The VSI OSAP network manager can use the OmniView monitoring utility. This chapter presents the utility and tells where it is documented.

10.1. VSI OMNI Monitoring Utility

The VSI OMNI monitoring utility for the VSI OSAP environment is OmniView. OmniView is a diagnostics and monitoring tool that uses DECwindows to obtain VSI OSAP services without writing any user application.

Before running OmniView in the VSI OSAP environment you must:

- Define an entry in the VSI OMNI Directory Service (through ODSCL) for each VMD that you wish to use inside OmniView. These entries must belong to a particular ODS Object Class, called VMD, which naming attribute is "/VN" (VMD Name), as shown in the example below. The value provided in the Application Simple Name (APPSN) attribute must be the equal to a "/CN" naming attribute of a valid entry in ODS belonging to the OSAP Object Class.

```
REGISTER DIRECTORY NAME "/VN = VN_OSH1CLT" ATTRIBUTES
                                                    "OC = VMD
                                                    /APROF = OSH1
                                                    /APPSN = (/CN=AD_OSH1CLT)
                                                    /INUM = 21
                                                    /MODEL = SIMATIC_155U"
```

- Define the logical name OMNI_CALLING_VMD equivalent to a valid entry in ODS for the VMD Object Class (see the example below). This entry must correspond to a definition for a local VMD; it will be used by OmniView as Calling VMD when it establishes Associations.

```
$ DEFINE OMNI_CALLING_VMD "/VN=VN_OSH1CLT"
```

The OmniView program is fully documented in the *VSI OMNI Guide to Using OmniView*.

Chapter 11. A Complete Example of a VSI OSAP Application

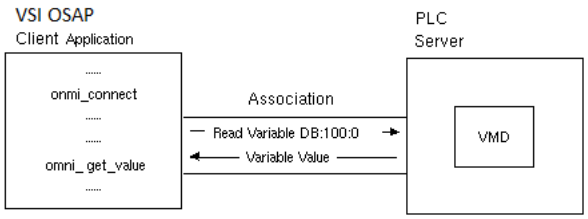
11.1. Introduction

This chapter describes how to configure a sample manufacturing plant consisting of two H1 devices that are interconnected through the IEEE 802.3 network:

- A VAX processor running VSI OSAP for OpenVMS.
- An S5 PLC equipped with either a CP 143 or a CP 535 Communication Processor.

The chapter also includes the C language code of a VSI OSAP client application that establishes an association with the remote server application, and issues a Variable Access service request to read the value of a variable from the PLC. This variable (of type word) is referred to by the VSI OSAP application as VMD_INT16 and its S5 address is DB:100:0 (see Figure 11.1, “Association Between the VSI OSAP H1 Client Application and the Remote H1 Server”).

Figure 11.1. Association Between the VSI OSAP H1 Client Application and the Remote H1 Server



To configure the sample manufacturing plant, you must configure:

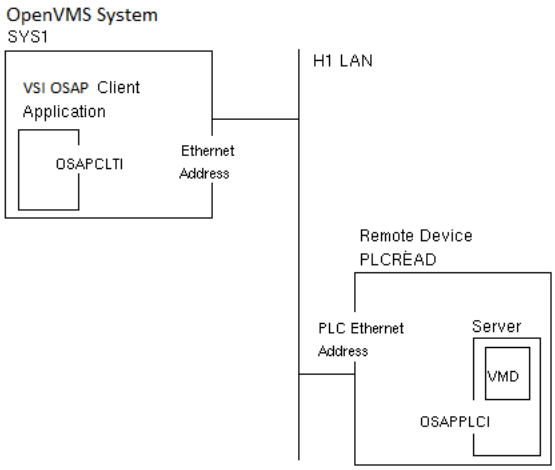
- The CP 535 (or 143) Communication Processor on the PLC, using the COM 535 (or 143) program.
- The VSI OSAP environment, using ODF commands and the omni_load_definition procedure, or ODSCL and the run-time object definition procedures.

11.2. PLC Configuration

In order to establish an association between the VSI OSAP application and the SINEC-H1 PLC, you must configure on the PLC Communication Processor some Data Link Blocks. These describe the characteristics of the Transport Connection (or Connections) corresponding to the services you wish to use. This example assumes that only the Read Variable service is supported.

Figure 11.2, “Association Between the VSI OSAP H1 Client Application and the Remote H1 Server” shows the static configuration of the sample manufacturing plant.

Figure 11.2. Association Between the VSI OSAP H1 Client Application and the Remote H1 Server



SYS1 and PLCREAD identify the OpenVMS system and the S5 PLC, respectively. Note that these names are not significant as Ethernet addresses.

OSAPCLTI and OSAPPLCI are the TSAP names that is used at run time to set up the Transport Connection.

11.2.1. Definition of the Static Characteristics of the Read Connection

Example 11.1, “The COM 535 Display Form (H1)” shows how to use the COM 535 (or 143) display form to define the parameters of the Transport Connection.

Example 11.1. The COM 535 Display Form (H1)

```
 DATA LINK BLOCK

-------------------------------------------------------------------------------
 FROM LOCAL PLC:
 SSNR : 0 ANR : 10 ❶
 JOB TYPE : FETCH ❷ ACTIVE/PASSIVE (A/P): P ❸
 TO REMOTE PLC:
 ETHERNET ADDRESS: 000000000000 H ❹ SSNR: ANR:

-------------------------------------------------------------------------------
 DATA LINK BLOCK - SCREEN II

-------------------------------------------------------------------------------
 MULTICAST (Y/N): N MULTICAST SET: ETHERNET ADDRESS: H
 DATAGRAM (Y/N): N
 PRIORITY : 2 READ/WRITE (Y/N): Y ❺
 SOURCE/DEST : LENGTH:
 STATUS WORD :
 INTERPRETER : ADDRESS: H
 LOCAL TSAP-ID : LENGTH: 8 HEX: 4F 53 48 31 50 4C 43 52 ASC: OSH1PLCR ❻
 REMOTE TSAP-ID: LENGTH: 7 HEX: 4F 53 48 31 43 4C 54 ASC: OSH1CLT ❼
 REMOTE NSAP-ID: LENGTH:12 ASC:
 NUMBER OF JOBS PER TSAP : 1

-------------------------------------------------------------------------------
```

❶ SSNR: Interface Number (0) ANR: Job Number (10)

❷ See point 5 below.

❸ ACTIVE/PASSIVE:

Specify the role in the connection establishment: active mode means that the PLC starts the connection establishment phase, passive mode means that it waits for a connection request coming from a remote system.

❹ ETHERNET ADDRESS:

Specify the Ethernet Address of the remote peer; a string of NULLs in the Passive operation means that the PLC accepts connection requests coming from whatever other node.

❺ JOB TYPE:

READ/WRITE: Specify the type of activity on this connection:

- FETCH, Yes: Read Variable
- RECEIVE, Yes: Write Variable
- SEND, No: Send Message
- RECEIVE, No: Receive Message

❻ LOCAL TSAP-ID:

Specify the TSAP associated with the PLC itself; it is either the Called TSAP (Passive mode, the current one) or the Calling TSAP (Active mode); note the VSI OSAP for SINEC-H1 TSAP convention adopted for the Called TSAP.

❼ REMOTE TSAP-ID:

Specify the TSAP associated to the remote system; it will be either the Calling TSAP (PLC Passive mode) or the Called TSAP (PLC Active mode).

11.2.2. Definition of the Variable

In SINEC-H1 you do not need to perform any configuration for the variables on the PLC side; in fact, the variables are referred to directly with their S5 addresses. (Of course they should refer to an existing PLC memory block.) In this example, the S5 address is DB:100:0, that is the first word of the Data Block number 100.

11.3. VSI OSAP Environment Configuration

To configure the VSI OSAP environment on the OpenVMS system, you must create:

- An Application Descriptor for the VSI OSAP client application
- An Application Descriptor for the remote server application
- The definition of a local VMD
- The definition of a remote VMD

Example 11.2, “Application Descriptor for the VSI OSAP H1 Client Application” shows how to create the Application Descriptor for the VSI OSAP client application, using the ODSCL REGISTER DIRECTORY command.

Example 11.2. Application Descriptor for the VSI OSAP H1 Client Application

```
REGISTER DIRECTORY NAME "/CN=AD_OSH1CLT" ATTRIBUTES
```

```
                                        "OC=OSH1
                                        /TSAP=OSH1CLT
                                        /NETADDRESS=IEEE%AA000400DFB9"
```

IEEEA% is a template written to VOTS using the NCL command (for further information refer to the NCL documentation).

Example 11.3, “Application Descriptor for the Remote H1 Server Application” shows how to create the Application Descriptor for the remote server application, using the ODSCL REGISTER DIRECTORY command.

Example 11.3. Application Descriptor for the Remote H1 Server Application

```
REGISTER DIRECTORY NAME "/CN=AD_OSH1PLC" ATTRIBUTES
                                        "OC=OSH1
                                        /TSAP=OSH1PLC
                                        /NETADDRESS=IEEE%080006010001"
```

The local VMD, the remote VMD and the VMD_INT16 variable are defined at run time using the VSI OMNI API procedures.

Part IV. Extension to VSI OMNI Procedure Calls

Chapter 12. Extension to VSI OMNI Procedure Calls

12.1. The list of VSI OMNI Procedure Calls

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 VSI OMNI.

Syntax

```
extern unsigned long int omni_define (
          omni_l_attributes class,
          omni_l_handle *handle);
```

Arguments

`class`

access: read only

mechanism: by value

Class identifies the class of definition to create.

The value of the class parameter is one of the values shown in Table 12.1, “Class Constants”.

Table 12.1. Class Constants

Constant	Meaning
omni_c_cls_vmd	VMD
omni_c_cls_dom	Domain
omni_c_cls_pi	Program Invocation
omni_c_cls_named_var	Named Variable (AP only)
omni_c_cls_unnamed_var	Unnamed Variable
omni_c_cls_msg	Message
omni_c_cls_mms_named_type	MMS Named Type
omni_c_cls_mms_type_spec	MMS Type Specification
omni_c_cls_app_named_type	Application Named Type
omni_c_cls_app_type_spec	Application Type Specification
omni_c_cls_mms_struct_comp	MMS Structure Component
omni_c_cls_app_struct_comp	Application Structure Component
omni_c_cls_var_list	Variable List

`handle`

access: write only

mechanism: by reference

Handle specifies the return address of the definition handle.

omni_exchange_data

omni_exchange_data — Requests or fulfills Exchange Message Serial Transfer services: 1. A VSI OSAP client application calls this procedure once to receive and transmit a message value to a remote server application, and to wait for a reply message value. 2. A VSI OSAP server application calls this procedure to transmit the message value which fulfills an exchange message indication received from a remote client application.

Syntax

```
extern unsigned long int omni_exchange_data [_a] (
          unsigned long int *invoke_id,
          omni_l_handle object_handle,
          char *request_data,
          long int request_data_length,
          omni_l_handle req_method_handle,
          char *response_data,
          unsigned long int response_data_length,
          omni_l_handle res_method_handle,
          omni_l_handle modifier_object,
          omni_r_iosb *iosb,
          omni_r_ctrl *ctrl);
```

Arguments

`invoke_id`

access: write only

mechanism: by reference

An identifier assigned by VSI OMNI. The argument is used for asynchronous calls only.

`object_handle`

access: read only

mechanism: by value

In a client call, the longword identifier of a loaded message definition.

In a server call, the context value returned by the omni_ get_indications call that delivered the exchange message indication.

The *object_handle* argument is the address of the handle.

`request_data`

access: read only

mechanism: by reference

A data structure that contains the message value to be sent. The first two bytes of the structure specify the size (in bytes) of the message value, the remaining bytes contain the message value.

`request_data_length`

access: read only

mechanism: by value The actual size (in bytes) of the message value to be sent.

`req_method_handle`

access: read only

mechanism: by value

optional argument

Reserved for future use.

`response_data`

access: read only

mechanism: by reference

A data structure to receive the message. The first two bytes will contain the actual length of the message value; the remaining bytes will contain the value of the received message.

`response_data_length`

access: read only

mechanism: by value

The actual size (in bytes) of the received message.

`res_method_handle`

access: read only

mechanism: by value

optional argument

Reserved for future use.

`modifier_object`

access: read only

mechanism: by value

optional argument

Reserved for future use.

`iosb`

access: write only

mechanism: by reference

optional argument

The VSI OMNI I/O status block. The *iosb* argument is the address of the status block.

`ctrl`

access: read only

mechanism: by reference

optional argument

A control structure to handle an event flag. The *ctrl* argument is the address of the control structure.

Usage Notes

The omni_exchange_data procedure requires the value of the message to be sent in the *request_data* argument, and returns the received message value in the *response_data* argument; with a single procedure call, the application sends a message and receives the reply.

When a VSI OSAP server application invokes the omni_exchange_data procedure to fulfill a message exchange indication, the received message is only returned after the message to be sent has been transmitted, that is, once the execution of the procedure is terminated. The server application can read the value of the received message before executing the operation, by invoking the omni_get_value procedure.

Note

The omni_exchange_data procedure is not supported by H1.

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 in a similar way to omni_get_handle_list. Each call to either of the routines returns one value. The value of the attribute parameter must specify the address of the attribute on the first call, and must be NULL thereafter. A value of NULL for the attribute parameter indicates that the next value must be specified or retrieved. In the case of omni_get_definition, when the last value has been retrieved, omni_s_endoflist is returned as the status value. The omni_end_list routine must be called after a list of values has been retrieved.

Syntax

```
extern unsigned long int omni_get_definition (
          omni_l_handle def_handle,
          omni_l_attributes *attribute,
          omni_l_context *context,
          void *value,
          unsigned long int value_length);
```

Arguments

`def_handle`

access: read only

mechanism: by value

Def_handle 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

`attribute`

access: read only

mechanism: by reference

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

`context`

access: read/write

mechanism: by reference

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

The value of context must be initialized to NULL. In general, the value of the context must be modified when as many values as required have been retrieved from a multi-valued attribute. In this case, omni_end_list must be called to free any unnecessary space allocated by VSI OMNI. The value of the context must then be reset to zero.

`value`

access: write only

mechanism: by reference

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

`value_length`

access: read only

mechanism: by reference

The size of the value in bytes.

The attributes supported for each definition class and the expected type of the buffer to receive the value are listed in Table 12.2, “VMD Attributes and Expected Data Types” and Table 12.3, “Domain Attributes and Expected Data Types”

Table 12.2, “VMD Attributes and Expected Data Types” shows VMD data information.

Table 12.2. VMD Attributes and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt

Attribute	Expected Data Type
omni_c_app_profile	Longword
omni_c_attr_logical_status	LongInteger
omni_c_attr_physical_status	LongInteger
omni_c_attr_local_detail	omni_a_local_detail
omni_c_attr_appl_simple_name	omni_t_appl_simple_name_wc
omni_c_attr_vmd_max_segment	Longword
omni_c_attr_vmd_max_srv_called	Word
omni_c_attr_vmd_max_srv_calling	Word
omni_c_attr_model	omni_t_visible_str_127_nt
omni_c_attr_vmd_nesting	Byte
omni_c_attr_vmd_param_ supported	omni_a_param_cbb
omni_c_attr_revision	omni_t_visible_str_127_nt
omni_c_attr_vmd_srv_supported	omni_a_services_supported
omni_c_attr_syntax_list	omni_t_object_id (AP only)
omni_c_attr_vendor	omni_t_visible_str_127_nt
omni_c_attr_vmd_version	Word
omni_c_attr_cls_dom	omni_t_mms_id_nt
omni_c_attr_cls_pi	omni_t_mms_id_nt
omni_c_attr_cls_named_var	omni_t_mms_id_nt (AP only)
omni_c_attr_cls_unnamed_var	omni_t_mms_id_nt
omni_c_attr_cls_mms_named_type	omni_t_mms_id_nt
omni_c_attr_cls_app_named_type	omni_t_mms_id_nt
omni_c_attr_user_param	Longword
AP Specific Attributes	
omni_osap_c_attr_notopen_srvspt	omni_osap_a_notopen_srvspt
omni_osap_c_attr_cpu_id	Long Integer
omni_osap_c_attr_cls_msg	omni_t_mms_id_nt
H1 Specific Attributes	
omni_osh1_c_attr_notopen_srvspt	omni_osh1_a_notopen_srvspt

Table 12.3, “Domain Attributes and Expected Data Types” shows Domain data information.

Table 12.3. Domain Attributes and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_profile	Longword
omni_c_attr_capability_file	omni_t_file_name (AP only)
omni_c_attr_capability	omni_t_visible_string_255 (AP only)
omni_c_attr_deletable	omni_b_boolean (AP only)
omni_c_attr_description	omni_t_description
omni_c_attr_dom_content_file	omni_t_file_name
omni_c_attr_scope	omni_l_handle

Attribute	Expected Data Type
omni_c_attr_sharable	omni_b_boolean (AP only)
omni_c_cls_named_var	omni_t_mms_id_nt (AP only)
omni_c_cls_unnamed_var	omni_t_mms_id_nt (AP only)
omni_c_cls_mms_named_type	omni_t_mms_id_nt (AP only)
omni_c_cls_app_named_type	omni_t_mms_id_nt (AP only)
omni_c_attr_user_param	Longword
H1 Specific Attributes	
omni_osh1_c_attr_blk_addr_list	omni_t_address_str

Table 12.4, “PI Attributes and Expected Data Types” shows PI data information.

Table 12.4. PI Attributes and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
mni_c_attr_profile	Longword
omni_c_attr_deletable	omni_b_boolean (AP only)
omni_c_attr_description	omni_t_description
omni_c_attr_ref_dom_names	omni_t_mms_id_nt
omni_c_attr_exec_arg	omni_t_visible_str_255_nt (AP only)
omni_c_attr_reusable	omni_b_boolean (AP only)
omni_c_attr_monitor	omni_l_pi_monitor (AP only)
omni_c_attr_scope	omni_l_handle
omni_c_attr_user_param	Longword

Table 12.5, “Named Variable Attributes and Expected Data Types (AP Only)” shows Named Variable data information.

Table 12.5. Named Variable Attributes and Expected Data Types (AP Only)

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_description	omni_t_description
omni_c_attr_mms_type_desc	omni_r_mms_type_spec
omni_c_attr_app_type_desc	omni_r_app_type_spec
omni_c_attr_scope	omni_l_handle
omni_c_attr_user_param	Longword
omni_c_attr_deletable	omni_b_boolean (AP only)

Table 12.6, “Unnamed Variable Attributes and Expected Data Types” shows Unnamed Variable data information.

Table 12.6. Unnamed Variable Attributes and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_description	omni_t_description

Attribute	Expected Data Type
omni_c_attr_mms_type_desc	omni_r_mms_type_spec
omni_c_attr_app_type_desc	omni_r_app_type_spec
omni_c_attr_scope	omni_l_handle
omni_c_attr_user_param	Longword
omni_c_attr_supply_type_spec	omni_b_boolean
omni_c_attr_address_type	omni_l_address_types
omni_c_attr_address_string	omni_t_address_str
omni_c_attr_address_number	Longword (AP only)

Table 12.7, “MMS Named Type Attributes and Expected Data Types” shows MMS Named Type data information.

Table 12.7. MMS Named Type Attributes and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_desc	omni_t_description
omni_c_attr_mms_type_description	omni_r_mms_type_spec
omni_c_attr_scope	omni_l_handle
omni_c_attr_user_param	Longword
omni_c_attr_deletable	omni_b_boolean

Table 12.8, “MMS Type Specification Attributes and Expected Data Types” shows MMS Type Specification data information.

Table 12.8. MMS Type Specification Attributes and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_mms_type_desc	omni_r_mms_type_spec
omni_c_attr_user_param	Longword
omni_c_attr_struct_comp	omni_l_handle
omni_c_attr_array_elem_mms_type	omni_l_handle
omni_c_attr_description	omni_t_description
omni_c_attr_scope	omni_l_handle

Usage Notes

- 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 12.9, “Application Named Type and Expected Data Types” shows Application Named Type data information.

Table 12.9. Application Named Type and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_description	omni_t_description
omni_c_attr_app_type_desc	omni_r_app_type_spec

Attribute	Expected Data Type
omni_c_attr_mms_named_type	omni_l_handle
omni_c_attr_scope	omni_l_handle
omni_c_attr_user_param	Longword
omni_c_attr_alternate_access	omni_b_boolean

Table 12.10, “Application Type Specification and Expected Data Types” shows Application Type Specification data information.

Table 12.10. Application Type Specification and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_app_type_spec	omni_r_app_type_spec
omni_c_attr_scope	omni_l_handle
omni_c_attr_user_param	Longword
omni_c_attr_desc	omni_t_description
omni_c_attr_mms_type_desc	omni_r_mms_type_spec
omni_c_attr_struct_comp	omni_l_handle
omni_c_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 12.11, “MMS Structure Component and Expected Data Types” shows MMS Structure Component data information.

Table 12.11. MMS Structure Component and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_description	omni_t_description
omni_c_attr_mms_type_desc	omni_r_mms_type_spec
omni_c_attr_scope	omni_l_handle
omni_c_attr_user_param	Longword

Table 12.12, “Application Structure Component and Expected Data Types” shows Application Structure Component data information.

Table 12.12. Application Structure Component and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_desc	omni_t_description
omni_c_attr_mms_struct_comp	omni_l_handle
omni_c_attr_app_type_desc	omni_r_app_type_spec
omni_c_attr_user_param	Longword
omni_c_attr_byte_offset	Longword
omni_c_attr_alternate_access	omni_b_boolean
omni_c_attr_scope	omni_l_handle

Table 12.13, “Message Attributes and Expected Data Types” shows message data information.

Table 12.13. Message Attributes and Expected Data Types

Attribute	Expected Data Type
omni_c_attr_name	omni_t_mms_id
omni_c_attr_description	omni_t_description
omni_c_attr_scope	omni_l_handle
omni_c_attr_msg_length	Word

- The value of the scope attribute is the handle of a VMD definition.

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.

Syntax

```
extern unsigned long int omni_get_handle_by_name (
          omni_l_handle scope,
          omni_l_classes class,
          omni_t_mms_id_nt def_name,
          unsigned long int def_name_length,
          omni_l_handle *handle);
```

Arguments

`scope`

access: read only

mechanism: by value

A handle indicating the scope of the search. The *scope* parameter is normally the handle of a VMD or a domain. VSI OMNI limits its search to the specified VMD or domain. The *scope* parameter is the handle of a VMD (obtained using omni_define) or a domain (obtained by a previous call to omni_get_handle_by_name).

`class`

access: read only

mechanism: by value

One of the constants shown in Table 12.14, “Class Constants” to indicate the class of definition to search for.

Table 12.14. Class Constants

Constant	Meaning
omni_c_cls_dom	Domain
omni_c_cls_pi	Program invocation
omni_c_cls_var	First variable type found
omni_c_cls_named_var	Named variable (AP only)

Constant	Meaning
omni_c_cls_unnamed_ ar	Unnamed variable
omni_c_cls_var_list	Variable list
omni_c_cls_mms_named_type	MMS Named Type
omni_c_cls_app_named_type	Application Named Type
omni_c_cls_msg	Message

`def_name`

access: read only

mechanism: by reference

The name of the definition to search for.

`def_name_length`

access: read only

mechanism: by value

The length of the def_name in characters.

`handle`

access: write only

mechanism: by reference

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.

Syntax

```
extern unsigned long int omni_get_handle_list (
          omni_l_handle scope,
          omni_l_classes *class,
          omni_l_context *context,
          omni_l_handle *handle);
```

Arguments

`scope`

access: read only

mechanism: by value

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_define, omni_get_handle_by_name, or omni_get_handle_list.

`class`

access: read only

mechanism: by reference

One of the constants shown in Table 12.15, “Definition Class Constants” (defined in OMNI_DEFS) to indicate the class of the definition who handle is to be retrieved.

Table 12.15. Definition Class Constants

Constant	Definition Class
omni_c_cls_dom	Domain
omni_c_cls_pi	Program invocation
omni_c_cls_named_var	Named variable (AP only)
omni_c_cls_unnamed_var	Unnamed variable
omni_c_cls_mms_named_type	MMS Named Type
omni_c_cls_app_named_type	Application Named Type
omni_c_cls_msg	Message

`context`

access: read only

mechanism: by reference

A variable for use by VSI OMNI. The value of this variable is modified by VSI OMNI. It must be initialized to NULL before the first call and must not be modified by the caller until the value of the return status is omni_s_endoflist or until omni_end_list has been called.

`handle`

access: write only

mechanism: by reference

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 receiv- ing 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 (related to AP services) from a remote application: 1. Read/write indications. 2. Unsolicited status. 3. Conclude indications 4. Abort indications. 5. Information reports

Syntax

```
extern unsigned long int omni_get_indications [_a](
          omni_l_handle vmd_handle,
          omni_l_handle *def_handle,
          omni_l_context *context,
          omni_l_attributes indication_type,
          char *reserved,
          omni_r_iosb *iosb,
```

```
            omni_r_ctrl *ctrl);
```

Arguments

vmd_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

access: write only

mechanism: by reference

An object definition handle returned by VSI OMNI. 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

access: write only

mechanism: by reference

Context information returned by VSI OMNI to be used in subsequent calls.

indication_type

access: write only

mechanism: by reference

One of the following constants listed in Table 12.16, “omni_get_indications Function Values” to indicate the type of indication received.

Table 12.16. omni_get_indications Function Values

Value	Meaning
omni_c_ind_acse_concl	Incoming conclude has completed.
omni_c_ind_pres_concl	Incoming conclude has been received. User must accept or reject the conclude (AP only).
omni_c_ind_abort	Incoming abort has been received.
omni_c_ind_read	Incoming read request has been received.
omni_c_ind_write	Incoming write request has been received.
omni_c_ind_msg_exch	Incoming exchange message request has been received (AP only).
omni_c_ind_unsol_status	Incoming status report has been received (AP only).
omni_c_ind_info_report	Incoming information report has been received.
omni_c_ind_status	Incoming status request has been received.

reserved

access: read only

mechanism: by reference

A place holder.

iosb

access: write only

mechanism: by reference

optional argument

The VSI OMNI I/O status block. The *iosb* parameter is the address of the status block.

ctrl

access: read only

mechanism: by reference

optional argument

A control structure to handle an event flag, AST routine, and AST parameter. The *ctrl* parameter is the address of the control structure.

Usage Notes

Issue one omni_get_indications call per remote VMD.

The omni_c_ind_read (omni_c_ind_write) indication specifies that the remote client application has issued either a read (or write) variable request for a Variable Access service, or a read (or write) message request for a Serial Transfer service. The VSI OSAP application must first check whether the returned object handle refers to a variable or to a message, and then perform the appropriate actions.

The omni_c_ind_info_report indication specifies that the server application has issued an unacknowledged send value request either for a variable or a message. The VSI OSAP client application must first check whether the returned object handle refers to a variable or to a message, and then perform the appropriate actions.

omni_get_value

omni_get_value — Obtains the value of a variable or of a message on a remote VMD. As a server procedure, omni_get_value obtains the value referred to by a write or an exchange message indication.

Syntax

```
extern unsigned long int omni_get_value [_a](
          unsigned long int *invoke_id,
          omni_l_handle handle,
          omni_l_handle method_handle,
          void *receiving_struct,
          unsigned long int receiving_struct_length,
          omni_l_handle modifier_object,
          omni_r_iosb *iosb,
          omni_r_ctrl *ctrl);
```

Arguments

`invoke_id`

access: write only

mechanism: by reference

optional argument

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

In a server call, the context value returned by the get_indications procedure call that delivered the write or the message exchange indication.

`handle`

access: read only

mechanism: by value

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

`method_handle`

access: read only

mechanism: by value

optional argument

Method_handle modifies the default presentation of a variable or a message. The identifier of a loaded access method. Use this parameter to override the method associated with this object in ODF.

`receiving_struct`

access: write only

mechanism: by reference

A data structure to receive the value of the object.

`receiving_struct_length`

access: read only

mechanism: by value

The size of the structure to receive data.

`modifier_object`

access: read only

mechanism: by value

optional argument

Reserved for future use.

`iosb`

access: write only

mechanism: by reference

optional argument

The VSI OMNI I/O status block. The *iosb* parameter is the address of the status block.

`ctrl_struc`

access: read only

mechanism: by reference

optional argument

A control structure to handle an event flag, AST routine, and AST parameter. The *ctrl* parameter is the address of the control structure.

omni_listen

omni_listen — Receives an association request from a remote application.

Syntax

```
extern unsigned long int omni_listen [_a](
          omni_l_handle called_vmd_handle,
          unsigned long int translate_flag,
          omni_l_handle calling_vmd_handle,
          omni_r_vmd_def *incoming_vmd_struc,
          omni_r_iosb *iosb,
          omni_r_ctrl *ctrl);
```

Arguments

`called_vmd_handle`

access: read only

mechanism: by value

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`

access: read only

mechanism: by value

One of the values shown in Table 12.17, “Indications Received” to specify the way VSI OMNI handles initiation indications received from a remote VMD.

Table 12.17. Indications Received

Value	Meaning
0	VSI OMNI rejects the initiation if the calling application specifies a VMD whose definition is not currently loaded.

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

`incoming_vmd_struc`

access: write only

mechanism: by reference

optional argument

Service parameters proposed by the calling (remote) application.

`iosb`

access: write only

mechanism: by reference

optional argument

The VSI OMNI I/O status block. The *iosb* parameter is the address of the status block.

`ctrl`

access: read only

mechanism: by reference

optional argument

A control structure to handle an event flag, AST routine, and AST parameter. The *ctrl* parameter is the address of the control structure.

omni_modify_definition

omni_modify_definition —

Syntax

```
extern unsigned long int omni_modify_definition (
          omni_l_handle def_handle,
          omni_l_attributes *attribute,
          omni_l_context *context,
          void *value,
          unsigned long int value_length);
```

Arguments

`def_handle`

access: read only

mechanism: by value

Def_handle 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

`attribute`

access: read only

mechanism: by reference

Attribute specifies the address of a variable whose value is the attribute to modify or NULL to specify multiple valued attribute.

`context`

access: read/write

mechanism: by reference

optional argument

Context is the address of a variable of type omni_l_context. This parameter is used by VSI OMNI only if you modify a multi-valued attribute.

The value of context must be initialized to NULL. In general, the value of the context must be modified when as many values as required have been retrieved from a multi-valued attribute. In this case, omni_end_list must be called to free any unnecessary space allocated by VSI OMNI. The value of the context must then be reset to zero.

`value`

access: read only

mechanism: by reference

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

`value_length`

access: read only

mechanism: by reference

The length of the attribute value buffer in bytes.

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.

Table 12.18, “VMD Attributes, Expected Data Types, and Defaults” shows VMD values.

Table 12.18. VMD Attributes, Expected Data Types, and Defaults

VMD Attributes	Expected Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_appl_simple_name	omni_t_appl_simple_name_wc	Name of VMD
omni_c_attr_vmd_max_segment	Longword 512	
omni_c_attr_vmd_max_srv_calling	Longword 5	
omni_c_attr_vmd_max_srv_called	Longword 5	
omni_c_attr_vmd_nesting	Byte 10	
omni_c_attr_vmd_param_supported	omni_a_param_cbb	See note
omni_c_attr_vmd_srv_supported	omni_a_services_supported	See note
omni_c_attr_vmd_version	Word 1	
omni_c_attr_vendor	omni_t_visible_str_127	""
omni_c_attr_model	omni_t_visible_str_127	""
omni_c_attr_revision	omni_t_visible_str_127	""
omni_c_attr_scope	null	None
omni_c_attr_description	omni_t_description	""
omni_c_attr_user_param	Longword	None
omni_c_attr_logical_status	Longword	None
omni_c_attr_physical_status	Longword	None
omni_c_attr_local_detail	omni_a_local_detail	None
AP Specific Attributes		
omni_osap_c_attr_notopen_srvspt	omni_osap_a_notopen_srvspt	See note
H1 Specific Attributes		
omni_osh1_c_attr_notopen_srvspt	omni_osh1_a_notopen_srvspt	See note

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 must be specified as NULL.
- The omni_osap_c_attr_cpu_id must be assigned the same value as specified in the configuration of the remote device. The default value is 1.

- In SINEC-H1, if you want to use Variable Access, Program Invocation and Domain Services, you must specify the omni_c_attr_model attribute.
- The following are defaults for omni_c_attr_vmd_param_ supported, omni_c_attr_vmd_srv_supported and omni_ osap_c_attr_notopen_srvspt.

  Default for omni_c_attr_vmd_param_supported:

```
  (0) STR1 (Arrays) TRUE
  (1) STR2 (Structures) TRUE
  (2) VNAM (Named Variables) TRUE
  (3) VALT (Alternate Access) TRUE
  (4) VADR (Unnamed Variables) TRUE
  (5) VSCA (Scattered Access) FALSE
  (6) TPY (Third Party) FALSE
  (7) VLIS (Named Variable List) FALSE
  (8) REAL (Real) FALSE
  (9) AKEC (Acknowledgment Event Conditions) FALSE
 (10) CEI (Evaluation Interval) FALSE
```

  Default for omni_c_attr_srv_supported:

```
        /* vmd support services */
  (0) Status TRUE
  (1) GetNameList FALSE
  (2) Identify TRUE
  (3) Rename FALSE
        /* variable access services */
  (4) Read TRUE
  (5) Write TRUE
  (6) GetVariableAccessAttributes FALSE
  (7) DefineNamedVariable FALSE
  (8) DefineScatteredAccess FALSE
  (9) GetScatteredAccessAttributes FALSE
 (10) DeleteVariableAccess FALSE
 (11) DefineNamedVariableList FALSE
 (12) GetNamedVariableListAttributes FALSE
 (13) DeleteNamedVariableList FALSE
 (14) DefineNamedType FALSE
 (15) GetNamedTypeAttributes FALSE
 (16) DeleteNamedType FALSE
        /* operator communication services */
 (17) Input FALSE
 (18) Output FALSE
        /* semaphore management services */
 (19) TakeControl FALSE
 (20) RelinquishControl FALSE
 (21) DefineSemaphore FALSE
 (22) DeleteSemaphore FALSE
 (23) ReportSemaphoreStatus FALSE
 (24) ReportPoolSemaphoreStatus FALSE
 (25) ReportSemaphoreEntryStatus FALSE
        /* domain management services */
 (26) InitiateDownloadSequence FALSE
 (27) DownloadSegment TRUE
 (28) TerminateDownloadSequence TRUE
 (29) InitiateUploadSequence FALSE
 (30) UploadSegment FALSE
 (31) TerminateUploadSequence FALSE
```

```
(32) RequestDomainDownload TRUE
(33) RequestDomainUpload TRUE
(34) LoadDomainContent FALSE
(35) StoreDomainContent FALSE
(36) DeleteDomain FALSE
(37) GetDomainAttributes FALSE
       /* program invocation management services */
(38) CreateProgramInvocation FALSE
(39) DeleteProgramInvocation FALSE
(40) Start FALSE
(41) Stop FALSE
(42) Resume FALSE
(43) Reset FALSE
(44) Kill FALSE
(45) GetProgramInvocationAttributes FALSE
       /* file management services */
(46) ObtainFile FALSE
       /* event management services */
(47) DefineEventCondition FALSE
(48) DeleteEventCondition FALSE
(49) GetEventConditionAttributes FALSE
(50) ReportEventConditionStatus FALSE
(51) AlterEventConditionmonitoring FALSE
(52) TriggerEvent FALSE
(53) DefineEventAction FALSE
(54) DeleteEventAction FALSE
(55) GetEventActionAttributes FALSE
(56) ReportEventActionStatus FALSE
(57) DefineEventEnrollment FALSE
(58) DeleteEventEnrollment FALSE
(59) AlterEventEnrollment FALSE
(60) ReportEventEnrollmentStatus FALSE
(61) GetEventEnrollmentAttributes FALSE
(62) AcknowledgeEventNotification FALSE
(63) GetAlarmSummary FALSE
(64) GetAlarmEnrollmentSummary FALSE
       /* journal management services */
(65) ReadJournal FALSE
(66) WriteJournal FALSE
(67) InitializeJournal FALSE
(68) ReportJournalStatus FALSE
(69) CreateJournal FALSE
(70) DeleteJournal FALSE
       /* vmd support services */
(71) GetCapabilityList FALSE
       /* file management services */
(72) FileOpen FALSE
(73) FileRead FALSE
(74) FileClose FALSE
(75) FileRename FALSE
(76) FileDelete FALSE
(77) FileDirectory FALSE
       /* unconfirmed services */
(78) UnsolicitedStatus TRUE
(79) InformationReport TRUE
(80) EventNotification FALSE
(81) AttachToEventCondition FALSE
(82) AttachToSemaphore FALSE
```

```
        /* additional services */
(83) Conclude TRUE
(84) Cancel FALSE
```

Default for omni_osap_c_attr_vmd_notopen_srvspt:

```
        /* serial transfer services */
  (0) Read TRUE
  (1) Write TRUE
  (2) Exchange TRUE
```

Table 12.19, “Domain Attributes, Expected Data Type, and Defaults” shows Domain values.

Table 12.19. Domain Attributes, Expected Data Type, and Defaults

Domain Constants	Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_capability_file	omni_t_file_name	omni_domains: [VMD]domain.cap where VMD is the name of the parent VMD and domain is the domain name
omni_c_attr_dom_content_file	omni_t_file_name	""
omni_c_attr_deletable	omni_b_boolean	True
omni_c_attr_sharable	omni_b_boolean	False
omni_c_attr_scope	omni_l_handle	None
omni_c_attr_description	omni_t_description	""
omni_c_attr_user_param	Longword	None
H1 Specific Attributes		
omni_osh1_c_attr_blk_addr_list	omni_t_address_str	See note

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 terrible.
- The value of the scope attribute must be a VMD handle.
- If you do not specify the omni_osh1_c_attr_blk_addr_list attribute, the Domain is the entire memory of the PLC.

Table 12.20, “PI Attributes, Expected Data Types, and Defaults” shows PI values.

Table 12.20. PI Attributes, Expected Data Types, and Defaults

PI Constants	Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_ref_dom_names	omni_l_handle	""
omni_c_attr_exec_arg	omni_t_visible_str_255_nt	""
omni_c_attr_deletable	omni_b_boolean	True
omni_c_attr_reusable	omni_b_boolean	False
omni_c_attr_monitor	omni_b_boolean	iiomni_c_pimnt_not_present
omni_c_attr_scope	omni_l_handle	None

PI Constants	Data Type	Defaults
omni_c_attr_description	omni_t_description	None
omni_c_attr_user_param	Longword	None
omni_c_attr_ec_ref	omni_r_mms_object_name	None
omni_c_attr_ea_ref	omni_r_mms_object_name	None
omni_c_attr_ee_ref	mni_r_mms_object_name	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.

Table 12.21, “Named Variable Attributes, Expected Data Types, and Defaults” shows Named Variable values.

Table 12.21. Named Variable Attributes, Expected Data Types, and Defaults

Named Variable	Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_app_type_desc	omni_l_handle	None
omni_c_attr_deletable	omni_b_boolean	True
omni_c_attr_scope	omni_l_handle	None
omni_c_attr_description	omni_t_description	""
omni_c_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.

Table 12.22, “Unnamed Variable Attributes, Expected Data Types, and Defaults” shows Unnamed Variable values.

Table 12.22. Unnamed Variable Attributes, Expected Data Types, and Defaults

Unnamed Variable	Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_app_type_desc	omni_l_handle	None
omni_c_attr_address_type	omni_l_address_types	None
omni_c_attr_address_string	omni_t_address_str	None

Unnamed Variable	Data Type	Defaults
omni_c_attr_address_number	Longword	None
omni_c_attr_supply_type_spec	omni_b_boolean	True
omni_c_attr_scope	omni_l_handle	None
omni_c_attr_description	omni_t_description	""
omni_c_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.
- VSI OMNI 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, VSI OMNI does not compare the value of the Address Type attribute with the type of the address.

Table 12.23, “MMS Named Type Attributes, Expected Data Types, and Default” shows MMS Named Type values.

Table 12.23. MMS Named Type Attributes, Expected Data Types, and Default

MMS Named Types	Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_mms_type_desc	omni_l_handle	None
omni_c_attr_deletable	omni_b_boolean	True
omni_c_attr_scope	omni_l_handle	None
omni_c_attr_description	omni_t_description	""
omni_c_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.

Table 12.24, “Application Named Type Attributes, Expected Data Types, and Defaults” shows Application Named Type values.

Table 12.24. Application Named Type Attributes, Expected Data Types, and Defaults

Application Named Types	Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_app_type_desc	omni_l_handle	None
omni_c_attr_mms_named_type	omni_l_handle	None
omni_c_attr_scope	omni_l_handle	None
omni_c_attr_description	omni_t_description	""
omni_c_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.

Table 12.25, “MMS Type Specification Attributes, Expected Data Types, and Defaults” shows MMS Type Specification values.

Table 12.25. MMS Type Specification Attributes, Expected Data Types, and Defaults

MMS Types	Data Type	Defaults
omni_c_attr_mms_type	omni_l_mms_data_types	None
omni_c_attr_array_elem_num	Longword	None
omni_c_attr_array_elem_mms_type	omni_l_handle	None
omni_c_attr_simple_size	Longword	0
omni_c_attr_simple_varying	omni_b_boolean	False
omni_c_attr_date_included	omni_b_boolean	False
omni_c_attr_description	omni_t_description	None
omni_c_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_ c_mmstype_array then the value of omni_c_attr_ array_elem_num attribute has been specified, or
  - If the value of the MMS Type Attribute is omni_c_ mmstype_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_c_mmstype_ * enumeration in omni_defs. The following constants are acceptable values:
  - omni_c_mmstype_array
  - omni_c_mmstype_structure
  - omni_c_mmstype_boolean
  - omni_c_mmstype_bit_str
  - omni_c_mmstype_integer
  - omni_c_mmstype_unsigned
  - omni_c_mmstype_floating_point
  - omni_c_mmstype_octet_str
  - omni_c_mmstype_visible_str
  - omni_c_mmstype_generalized_time
  - omni_c_mmstype_binary_time

Table 12.26, “Application Type Specification Attributes, Expected Data Types, and Defaults” shows Application Type Specification values.

Table 12.26. Application Type Specification Attributes, Expected Data Types, and Defaults

Application Types	Data Type	Defaults
omni_c_attr_app_type	omni_l_app_data_type	None
omni_c_attr_mms_type_spec	omni_l_handle	None
omni_c_attr_array_elem_num	Longword	None
omni_c_attr_array_elem_app_type	omni_l_handle	None
omni_c_attr_array_low	Longword	None
omni_c_attr_simple_size	Longword	None
omni_c_attr_description	omni_t_description	None
omni_c_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 omni_ c_apptype_array then the values of the omni_c_attr_ array_elem_app_type attribute and the omni_c_attr_ array_elem_num attribute have been specified, or
  - If the value of the Application Type Attribute is omni_ c_apptype_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 omni_c_apptype_ * enumeration. The following constants are acceptable values:
  - omni_c_apptype_array
  - omni_c_apptype_structure
  - omni_c_apptype_boolean
  - omni_c_apptype_bit_str
  - omni_c_apptype_integer
  - omni_c_apptype_unsigned
  - omni_c_apptype_f_float
  - omni_c_apptype_terminated_str
  - omni_c_apptype_word_counted_str
  - omni_c_apptype_scalar_str
  - omni_c_apptype_omni_time
  - omni_c_apptype_boolean_array

Table 12.27, “MMS Structure Component Attributes, Expected Data Types, and Defaults” shows MMS Structure Component values.

Table 12.27. MMS Structure Component Attributes, Expected Data Types, and Defaults

MMS Structure Component	Data Type	Defaults
omni_c_attr_name	omni_t_mms_id_nt	None
omni_c_attr_mms_type_desc	omni_l_handle	None
omni_c_attr_description	omni_t_description	None
omni_c_attr_user_param	Longword	None
omni_c_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_c_mmstype_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.

Table 12.28, “App Structure Component Attributes, Values, and Defaults” shows App Structure Component values.

Table 12.28. App Structure Component Attributes, Values, and Defaults

App Structure Component	Value	Defaults
omni_c_attr_name	omni_t_app_id_nt	None
omni_c_attr_description	omni_t_description	None
omni_c_attr_app_type_desc	omni_l_handle	None
omni_c_attr_mms_struct_comp	omni_l_handle	None
omni_c_attr_scope	omni_l_handle	None
omni_c_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_c_apptype_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.

Table 12.29, “Message Attributes and Expected Data Types” shows message data information.

Table 12.29. Message Attributes and Expected Data Types

Attributes	Expected Data Type
omni_c_attr_name	omni_t_mms_id_nt
omni_c_attr_description	omni_t_description
omni_c_attr_scope	omni_l_handle
omni_c_attr_msg_length	Word

The value of the scope attribute is the handle of a VMD definition.

omni_put_value

omni_put_value — Modifies the value of a variable or a message on a remote VMD. As a server procedure, omni_put_value transmits the value of the variable or of the message specified by a read indication.

Syntax

```
extern unsigned long int omni_put_value [_a](
          unsigned long int *invoke_id,
          omni_l_handle obj_handle,
          omni_l_handle method_handle,
          void *value_struct,
          unsigned long int value_struct_length,
          omni_l_handle modifier_object,
          omni_r_iosb *iosb,
          omni_r_ctrl *ctrl);
```

Arguments

`invoke_id`

access: write only

mechanism: by reference

optional argument

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

`obj_handle`

access: read only

mechanism: by value

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

In a server call, the context value returned by the omni_get_indications call that delivered the read indication.

`method_handle`

access: read only

mechanism: by value

optional argument

Method_handle modifies the default presentation of a variable. The identifier of a loaded access method. Use this parameter to override the method associated with this object in ODF.

The *method_handle* parameter is the handle.

`value_struct`

access: read only

mechanism: by reference

A data structure containing the value of the object.

`value_struct_length`

access: read only

mechanism: by value

The length of the value_struct to send.

modifier_object

access: read only

mechanism: by value

optional argument

Reserved for future use.

iosb

access: write only

mechanism: by reference

optional argument

The VSI OMNI I/O status block. The *iosb* parameter is the address of the status block.

ctrl

access: read only

mechanism: by reference

optional argument

A control structure to handle an event flag, AST routine, and AST parameter. The *ctrl* parameter is the address of the control structure.

omni_send_value

omni_send_value — Returns the status of unconfirmed services on the server side specifically, InformationReport, UnsolicitedStatus, and SendMessage.

Syntax

```
extern unsigned long int omni_send_value [_a](
          omni_l_handle remote_vmd_handle,
          omni_l_handle object_handle,
          omni_l_handle method_handle,
          void *value_struct,
          unsigned long int value_struct_length,
          omni_r_iosb *iosb,
          omni_r_ctrl *ctrl);
```

Arguments

remote_vmd_handle

access: read only

mechanism: by value

Handle of the VMD that receives the unsolicited_status.

object_handle

access: read only

mechanism: by value

Handle of a loaded object. The class of the object determines the service provided. If vmd_class, the service is UnsolicitedStatus; if var_class, the service is InformationReport; If msg_class, the service is SendMessage.

`method_handle`

access: read only

mechanism: by value

optional argument

Handle to the access method for this operation.

`value_struct`

access: read only

mechanism: by reference

optional argument

Address of a data structure that contains the value of the variable or the message object.

`value_struct_length`

access: read only

mechanism: by value

optional argument

The length of the value_struct to send.

`iosb`

access: write only

mechanism: by reference

optional argument

The VSI OMNI I/O status block. The *iosb* parameter is the address of the status block.

`ctrl`

access: read only

mechanism: by reference

optional argument

A control structure to control the asynchronous completion of this function. The *ctrl* parameter is the address of the control structure.

12.2. AP Error Messages

In addition to the error messages returned in the omni_ l_iosb_general field of the I/O Status Block, the VSI OMNI API procedures invoked by a VSI OSAP application return additional error information in the omni_l_iosb_network field.

Error messages returned in the omni_l_iosb_general field are documented in the *VSI OMNI Application Programmer's Guide.*

Error messages returned in the omni_l_iosb_network field are specific to the VSI OSAP context. Table 12.30, “AP Error Messages” lists all these errors together with an indication of which API procedures return them.

Table 12.30. AP Error Messages

AP Error Messages	Returned by
abortfail	the omni_abort procedure
accdenied	the procedures implementing the AP Serial Transfer Services
applparmism	the omni_connect and omni_listen procedures
asoabort	the get_indications procedure
asonotcrea	the omni_connect procedur
callapplmiss	the omni_connect and omni_listen procedures
conclrej	the omni_conclude procedure
concludefail	the omni_conclude procedure
datarefused	the procedures implementing the AP Serial Transfer Services
fewdata	the procedures implementing the AP Serial Transfer Services
initrej	the omni_connect procedure
internal	all the procedures implementing the AP Services
mismatch	the omni_get_value, omni_put_value and omni_send_value procedures
moredata	the procedures implementing the AP Serial Transfer Services
nosuchasn	the omni_connect and omni_listen procedure
nosuchreq	the omni_cancel procedure
objaddrerr	the omni_get_value and omni_put_value procedures
proterr	all the procedures implementing the AP Services
remapmerr	all the API procedures
reqabort	all the procedures implementing the AP Services
reqtimeout	the omni_get_value, omni_put_value and omni_exchange_data procedures
success	all the procedures implementing the AP services
syntaxfail	the omni_connect and omni_listen procedures
unaligned	the omni_get_value, omni_put_value and omni_send_value procedures
vartyperr	the procedures implementing the AP Variable Access Services

12.3. H1 Error Messages

In addition to the error messages returned in the omni_ l_iosb_general field of the I/O Status Block, the VSI OMNI API procedures invoked by a VSI OSAP application return additional error information in the omni_l_iosb_network field.

Error messages returned in the omni_l_iosb_general field are documented in the *VSI OMNI Application Programmer's Guide.*

Error messages returned in the omni_l_iosb_network field are specific to the VSI OSAP context. Table 12.31, “H1 Error Messages” lists all these errors, together with an indication of which API procedures return them.

Table 12.31. H1 Error Messages

Error Message	Returned by
abortfail	the omni_abort procedure
asoabort	the get_indications procedure
asonotcrea	the omni_connect procedure.
callapplmiss	the omni_connect and omni_listen procedures
concludefail	the omni_conclude procedure
internal	all the procedures implementing the H1 Services
nosuchasn	the omni_connect and omni_listen procedure
nosuchreq	the omni_cancel procedure
objaddrerr	the omni_get_value and omni_put_value procedures
proterr	all the procedures implementing the H1 Services
reqabort	all the procedures implementing the H1 Services
reqtimeout	the omni_get_value, omni_put_value and omni_exchange_data procedures
success	all the procedures implementing the H1 services
unaligned	the omni_get_value, omni_put_value and omni_send_value procedures
vartyperr	the procedures implementing the H1 Variable Access Services

Part V. Appendices

Appendix A. Overview of SINEC and Related Siemens Products

The information contained in this appendix is intended to introduce VSI OSAP readers to SINEC, the architecture designed by Siemens for communication among factory automation systems and components.

For detailed information on the topics presented here, you should refer to the appropriate official documentation issued by Siemens.

A.1. SINEC Overview

Siemens provides plant floor automation components that constitute the building blocks of a Flexible Manufacturing system.

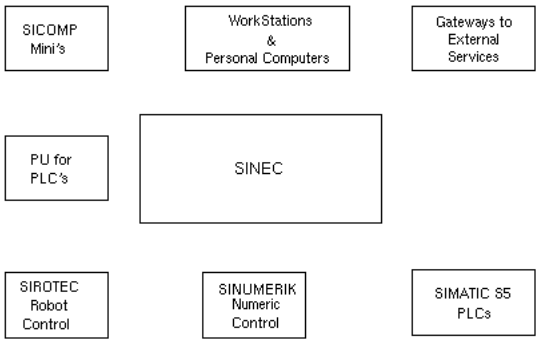
Communication among these automation components is made possible by the Siemens SINEC architecture and the products that support it.

SINEC is a means of communication for Siemens automation products in the industrial communications field (for example, control systems, personal computers and process control computers). It offers a uniform communication capability to all Siemens systems and leads to open communication using international standardized communication protocols.

Figure A.1, “Siemens SINEC: Integration of Automation Components” identifies SINEC as a common tool for the:

- Interconnection of all Siemens equipment
- Integration of third-party systems into a Siemens CIM system, provided that they implement appropriate Siemens-defined communication protocols.

Figure A.1. Siemens SINEC: Integration of Automation Components



A.1.1. Architecture

Figure A.2, “SINEC and the ISO/OSI Reference Model” highlights the two main parts of the layered SINEC architecture:

- SINEC-AP (Automation Protocol)
- An application-to-application protocol which provides services equivalent to those defined by the Session, Presentation and Application layers of the ISO/OSI reference model (layers 5, 6 and 7). . SINEC-H1
- A LAN-based Transport network for manufacturing cells. SINEC-H1 is based on international standards IEEE 802.3 (CSMA/CD) and ISO/OSI Network and Transport layers (layers 1, 2, 3 and 4).

Figure A.2. SINEC and the ISO/OSI Reference Model

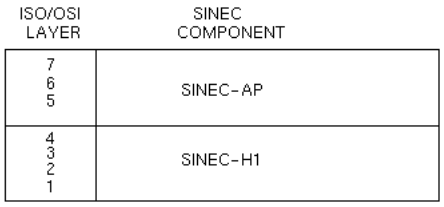
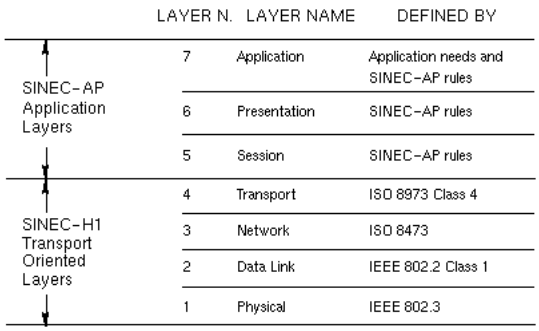


Figure A.3, “Comparing SINEC with the ISO/OSI Reference Model” shows the functional correspondence that exists between the layers of the SINEC architecture and the ISO /OSI reference model.

Figure A.3. Comparing SINEC with the ISO/OSI Reference Model



A.1.2. The SINEC-H1 Local Area Transport Network

The objective of SINEC-H1 is to provide fast communications on a high-speed local area transport network between all the Siemens automation components in a shop floor plant.

In the overall SINEC architecture, SINEC-H1 provides a common and standardized backbone transport network to the upper application layers of SINEC-AP.

SINEC-H1 is a LAN for manufacturing cells, based on international standards IEEE 802.3 (CSMA/CD) and ISO/OSI (layers 3 and 4).

The main characteristics of SINEC H1 are as follows:

- Transport Protocol: ISO 8073 Class 4
- Data Link Protocol: IEEE 802.2 Class 1
- Bus arbitration method: CSMA/CD following IEEE 802.3
- Medium: Triaxial Cable, for noise immunity
- Transmission mode: Serial . Gross baud rate: 10 megabits/sec.
- Topology: Up to two repeaters between two nodes (maximum one remote repeater pair)
- Distances: 500 m (1600 ft.) without repeater; 1.5 km with two repeaters; 2.5 km with remote repeater pair
- Number of Nodes: 100 nodes per segment; 1024 nodes in the overall network.

Most of the Siemens automation components can be connected to the SINEC-H1 LAN, for example:

- SICOMP Minicomputers (M20, M25, M30, M50, M70, and M80)

- SIMATIC S5 Programmable Controllers (S5-115U, S5-135U, S5-150U, and S5 155U)
- SINUMERIK Numerical Controls (NC850 and others)
- SIROTEC Robot Controls (RCM) . SICOMP WS Workstations (WS30)
- SICOMP PC Personal Computers (PC16 and PC32) . PC 32-XX
- SICOMP MMC 216 Multi-Microcomputer System . SINEC Concentrators (for example, SINEC 131 for connecting systems and devices that do not have a SINEC capability, and SINEC 132 for connection to SICOMP M peripherals).

A.1.3. The SINEC-AP Protocol

SINEC-AP is an application-to-application protocol developed on top of the ISO/OSI standard Transport layer. This protocol handles communication issues that conceptually be- long to the Session, Presentation and Application layers of ISO /OSI.

SINEC-AP defines the rules for communications between **AP-applications**, that is, the conceptual unit which implements one or more application functions of the overall automation task. AP-applications can be distributed all over in the SINEC network and they communicate transparently with respect to the SINEC lower levels.

As well as defining the guidelines for the creation of associations between AP-applications, and managing communication transparently with respect to the user level, SINEC-AP also provides a standard definition of the so called **AP Technological Functions**, that is, the Application layer functions used to specify what one partner application wants from the other.

Note

To maintain a correspondence with the MMS model and its terminology, the AP Technological Functions are referred to as **AP services** in this manual.

An AP-application requests the execution of a service by specifying the symbolic name of the association to be established, together with what the partner application should perform, via the appropriate AP service.

The set of rules that makes up the SINEC-AP protocol includes:

- Protocol sequences
- Coding and syntax.

SINEC-AP rules are defined by the Session, Presentation and Application layers of SINEC architecture. This makes it possible for communications partners residing on different automation systems to exchange standardized messages across the SINEC-H1 Transport network.

A.1.3.1. AP-Association

In a manufacturing application environment based on SINEC architecture, communication between AP-applications takes the form of a set of logical relationships, called **AP-associations**.

An AP-association is established dynamically at run time be- tween two applications, and allows commands, responses, data and other information needed to perform the required task in the automation isle, to be exchanged among the partner applications.

An AP-association is identified by a symbolic name local to each side of the actual AP-association. SINEC-AP is able to map the local symbolic name of a given AP-association onto the physical location in the network of the partner application to which it refers. The mapping is performed by AP in a transparent mode with respect to the applications.

The AP-association makes the overall automation application independent of both the network characteristics and topology, and the partner application's physical location in the network.

An AP-association is logically identified by the names of the two partners' applications. From a physical point of view, an AP-association is identified by the following parameters:

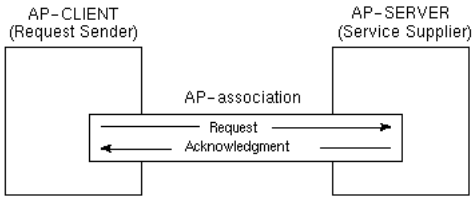
- The underlying Transport Connection (this is identified by the TSAPs of the calling user and the called user), and the Network Addresses of the systems on which the two applications reside.
- The Multiplex Identifier, which identifies one of the multiple AP-associations on the same Transport Connection.

Each AP-application is uniquely identified in the network by means of NSAP, TSAP and Multiplex Identifier. Two sets of these parameters are used by the AP-Monitor at run time, to establish an AP-association between two AP-applications.

A.1.3.2. AP-Client and AP-Server Applications

SINEC-AP defines the requirements that must be met in order for AP-applications to collaborate (see also Figure A.4, “Client and Server Roles in SINEC-AP”).

Figure A.4. Client and Server Roles in SINEC-AP



An AP-application that actively uses a corresponding AP- application on a remote system in order to request the execution of AP services is called an **AP-Client** application.

An AP-application that fulfills the AP services requested by an AP-Client is called an **AP-Server** application.

An AP-application can represent either of these roles, or both of them.

A.1.3.3. SINEC-AP Message Structure

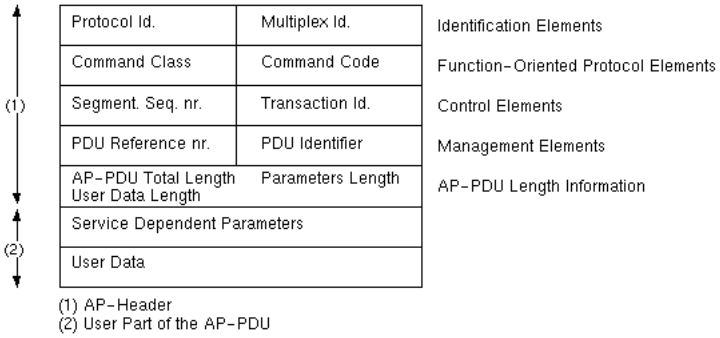
The elements of the dialogue for protocol sequences are defined in SINEC-AP together with message sequences. Dialogue elements are types of message (for example, requests and acknowledgments).

The syntax of a message, that is, an **AP-Protocol Data Unit (AP-PDU)** , is defined in SINEC-AP mainly in the form of setting parameters (AP-Header and parameter). However, keyword parameters (in the form of Format-ID) are also accepted when flexibility is required.

As shown in Figure A.5, “Syntax of a Generic AP-PDU”, a dialogue element is made up of the AP-Header and the User Part. The AP-Header consists of the following groups of protocol elements:

- *Identification elements*. These identify an AP-association.
- *Function-oriented protocol elements*. These define the meaning of the message content, that is, the AP service.
- *Control elements*. These control sequences of dialogue elements, and provide information on them.
- *Management elements*. These provide information on the management of AP-application requests.
- *AP-PDU length information*. This provides information on the data structure of the AP-PDU.

Figure A.5. Syntax of a Generic AP-PDU



A.1.3.4. The AP-Monitor

The AP-Monitor is the kernel of each AP implementation and is present on all the AP systems.

The AP-Monitor handles all communications-related issues on top of the Transport layer. Its main task is to implement the Session layer services of the SINEC architecture. It is responsible for the set-up and shut-down of Transport Connections, the dispatching of AP-messages (in accordance with the AP-protocol sequences), and the management of AP-message flow control. It also handles the multiplexing of several AP-associations on the same Transport Connection.

In addition to the AP-Session layer functions, the AP- Monitor parses the AP-messages and dispatches the AP-requests and acknowledgments. These functions are equivalent to those performed by the MMS Provider.

The main AP-Monitor functions are as follows:

- *AP-association management.* An AP-application initiates (or concludes) an AP-association by requesting the appropriate services from the AP-Monitor.
- *Request management.* The AP-Monitor matches the acknowledgments received with the proper requests, based on the management protocol elements of the AP-Header. In the case of a Transport Connection break-down, the AP-Monitor must recover the requests currently being processed.
- *Transparent distribution of AP-messages.* The AP-Monitor determines the proper Transport Connection to be used to send the message, based on the identification protocol elements of the AP-Header.
- *Flow control.* The AP-protocol specification describes a strategy for the implementation of flow control. This is an implementation recommendation rather than a protocol element, because flow control does not involve network activity (the partner is not notified of flow control problems on the local node). However, the AP-Monitor must implement some sort of flow control strategy.
- *Management of timing issues of messages.* Time-out problems on responses and message re-transmission must be handled by the AP-Monitor.
- *Multiplexing of Transport Connections.* AP allows for the multiplexing of Transport communication resources among numerous AP-applications. For unambiguous identification of the communication path, a multiplexing identifier is associated to the name of the Transport Service Access Point (TSAP).
- *Transport Connections management: set-up, recovery and clear-down.* The AP-Monitor is driven by the addressing information stored in the AP-database in order to set up the Transport Connection to the remote partner, perform restart and recovery actions if the connection crashes, and request disconnection when the communication path is no longer needed.

A.2. AP Objects and Services

As already stated in Section A.1, “SINEC Overview”, SINEC-AP provides AP-applications with tools for the management of AP-associations and the flow of data over them. Furthermore, a set of standardized AP services is defined in the Application and Presentation layers of SINEC-AP.

An AP-Client requests the execution of an AP service from an AP-Server to perform automation tasks, for example:

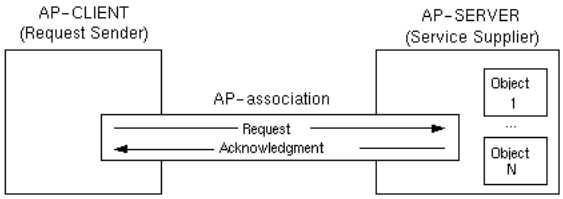
- Reading and writing variables
- Uploading and downloading Domains
- Obtaining attributes of objects that reside on remote systems
- Exchanging messages.

SINEC-AP codes all the standardized services that an AP- Client can request from the AP-Server (or AP-Servers). These services operate on remotely defined **objects**, such as Virtual Manufacturing Devices (VMDs), Variables, Domains, Program Invocations, and Messages.

These objects are owned by an AP-Server application (Object Handler) and are accessed by AP-Client applications. AP services always involve communication between two AP- applications. In this context, the term "communicating" means giving access to the objects owned by an application.

An object is accessed by means of a service request which flows from the AP-Client to the AP-Server. If required, the AP-Server sends the acknowledgment (see also Figure A.6, “How an AP-Client Accesses the Objects Owned by an AP-Server”).

Figure A.6. How an AP-Client Accesses the Objects Owned by an AP-Server



A.2.1. Classes of AP Objects

The AP-specification uses an abstract object modeling technique to fully describe the device model and service procedures of AP. This modeling technique describes the abstract objects, together with their characteristics and the operations performed on them.

The objects defined are abstract, and when implementing AP, a real system maps the concepts described in the model onto the characteristics and functions of the real system.

AP defines a number of object classes. Each object is an instance of a class and constitutes an abstract entity which exhibits certain characteristics, and may be affected by certain AP-services and operations.

A.2.1.1. Object Scope

Every AP-object has a lifetime (or scope) within the VMD, which is inferred by the scope of its name. Possible AP object scopes are:

- VMD-specific scope

  These objects exist for as long as the VMD exists (unless explicitly deleted).

- Domain-specific scope

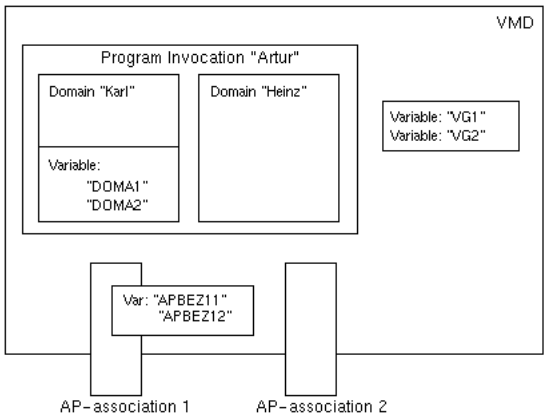
  These objects exist for as long as the Domain that they depend on exists (unless explicitly deleted).

- AA-specific (Application Association) scope

  These objects exist for as long as the AP-association on which they were defined continues to exist (unless explicitly deleted).

Figure A.7, “Example of Object Definition” illustrates an overview of a VMD, two Domains, a Program Invocation and some allocated Variables.

Figure A.7. Example of Object Definition



Two logical partner names are connected with the VMD (AP-association 1 and AP-association 2).

The scope of variables APBEZ11 and APBEZ12 is specific to an AP-association. These variables are allocated to AP-association 1 and can only be accessed on that association. The scope of variables VG1 and VG2 is specific to the VMD. Since these variables are associated to the VMD, they can be accessed on both AP-association 1 and AP-association 2.

The VMD contains two domains, called Karl and Heinz. Both variables DOMA1 and DOMA2 have a Domain-specific scope and are allocated to Domain Karl. These variables can therefore be accessed on both association 1 and association 2, and are under the definition of Domain Karl.

Domain Karl and Domain Heinz are used by the Program Invocation called "Artur". This Program Invocation can therefore be addressed by means of both AP-association 1 and AP-association 2.

A.2.2. Classes of AP Services

The AP services currently available are grouped into the classes shown in Table A.1, “Classes of AP Services”.

Table A.1. Classes of AP Services

AP Service	Class Description
Environment Management	Allows AP-applications to establish and control a peer-to-peer association.
VMD Support Services	Allows an application to obtain information on the capabilities and status of a VMD.
Variable Access Services	Allows operations to be performed on variables and their attributes.
Domain Management Services	Allows an AP-Client to operate on the Domains defined on the AP-Server.
Program Invocation Services	Allows remote program execution under the control of the AP-Client.
Serial Transfer Services	Allows the partners of an AP-association to exchange unformatted data messages.
Time Functions	Provides a network-wide timer that is used for control, protocol and supervision purposes.

Depending on whether or not they are "functionally compatible" with MMS, the classes of AP services listed in Table A.1, “Classes of AP Services” can be grouped into:

- Open: services that are "functionally compatible" with MMS.
- Not-open: services that are only provided by SINEC AP (not provided by MMS).

"Functionally compatible" with MMS means:

- The services of a given open AP class also belong to the corresponding MMS class.
- The parameters specified by the AP services of a given class are a subset of those of the corresponding MMS class.

The open classes of AP services are:

- Environment Management
- VMD Support
- Variable Access
- Domain Management
- Program Invocation.

For a complete and formal description of open AP services, refer to the appropriate Siemens documentation, and MMS documentation (where applicable).

The non-open classes of AP services are:

- Serial Transfer Services
- Time Functions

These classes are derived from the corresponding functions implemented on older Siemens protocols and products (the exchange of unstructured strings of data between applications is also implemented in the old SINEC-H1 services).

The non-open AP services are briefly described in Section A.2.2.2, “Description of the Serial Transfer Service Class”.

A.2.2.1. List of AP Services

Table A.2, “List of AP Services” lists the services belonging to each AP service class.

Table A.2. List of AP Services

Service	Description
Environment Management	
Initiate	Initiates communication with another user in the SINEC-AP environment, and establishes the requirements and capabilities which support that communication.
Conclude	Concludes communication with another user in the SINEC-AP environment in an ordered manner.
Abort	Aborts communication with another user in the SINEC-AP environment in an abrupt manner.
VMD Support	
Status	Gets the status of a VMD.
GetNameList	Gets the list of various defined objects.
Identify	Obtains identifying information from a responding AP-user.
GetCapabilityList	Gets the list of the VMD's capabilities.
UnsolicitedStatus	Receives an unsolicited message about the status of the VMD.
Variable Access	
Read	Returns the value of one or more variables defined at the VMD.

Service	Description
Write	Replaces the content of one or more variables with values specified in the request.
InformationReport	Informs the other AP-user of the value of one or more specified variables, as read by the issuing AP-user.
GetVariableAccessAttributes	Returns the attributes of a variable object defined at the VMD.
Domain Management	
InitiateDownloadSequence	Used by an AP-Client to begin a download sequence, to load a Domain at an AP-Server.
DownloadSegment	Used by an AP-Server to obtain elements of the download information.
TerminateDownloadSequence	Used by an AP-Server to terminate the download sequence.
InitiateUploadSequence	Used by an AP-Client to begin the process of uploading the contents of a specific Domain.
UploadSegment	Used by an AP-Client to obtain an upload segment from the AP-Server.
TerminateUploadSequence	Used by an AP-Client to terminate the upload sequence.
RequestDomainDownload	Used by an AP-Server to request the AP-Client to initiate a Download Sequence for the server.
RequestDomainUpload	Used by the AP-Server to request the AP-Client to initiate an Upload Sequence for the AP-Server, specifying the content of a named Domain.
LoadDomainContent	Used by an AP-Client to request the AP-Server to take action to load a Domain.
DeleteDomain	Used by an AP-Client to request the AP-Server to delete the specified Domain and make its resources available.
GetDomainAttributes	Used by an AP-Client to request the AP-Server to list the attributes of the specified Domain.
Program Invocation	
CreateProgramInvocation	Used by an AP-Client to create a new Program Invocation object at a VMD.
DeleteProgramInvocation	Used by an AP-Client to delete a Program Invocation object at a VMD.
Start	Used by an AP-Client to cause a previously defined Program Invocation to transit to the RUNNING state.
Stop	Used by an AP-client to cause a Program Invocation in the RUNNING state to transit to the STOPPED state.
Resume	Used by an AP-Client to cause a Program Invocation in the STOPPED state to transit to the RUNNING state.
Reset	Used by an AP-Client to cause a Program Invocation in the STOPPED state to transit to the IDLE state.
Kill	Used by an AP-Client to terminate a Program Invocation by causing it to transit to the UNRUNNABLE state.
GetProgramInvocationAttribute	Used by an AP-Client to determine the attributes of a Program Invocation, together with its state and list of dependent Domains.
Serial Transfer	
ReadMessage	Used by an AP-Client to transfer octets of data to the AP-Server.

Service	Description
WriteMessage	Used by an AP-Client to request the AP-Server to transfer a data buffer.
ExchangeMessage	Used by the partners of an AP-association to transfer data in both directions.
Time Function	
TimeRead	Used by an AP-Client to obtain the time from an AP-Server.
TimeForce	Used by an AP-Client to automatically transfer the current value of the timer to the AP-Server. The AP-Server uses this value to synchronize its clock.
TimeWrite	Used by an AP-Client to transfer the AP-time of its clock to the AP- Server.

A.2.2.2. Description of the Serial Transfer Service Class

The services of the Serial Transfer class allow the partners of an AP-association to exchange unformatted data messages. These services can either be acknowledged or unacknowledged.

The communication system does not insert any protocol information in the exchanged data, that is, the AP-applications must agree on the semantics of the data. Notwithstanding their simplicity, the Serial Transfer services cannot be considered as mere entry points to the Transport layer; they are in fact part of the AP-protocol and therefore take advantage of the services provided by the AP-Monitor.

Bearing in mind that an AP service is requested by an AP-Client application, the following services belong to the Serial Transfer class:

- *WriteMessage.* The AP-Client uses this service to transfer octets of data to the AP-Server.
- *ReadMessage.* The AP-Client uses this service to request the AP-Server to transfer a data buffer. The protocol specification does not provide any information on the number of requested data octets (this number should be agreed on by the two partners).
- *ExchangeMessage.* This service allows for data to be transferred in both directions between the partners of the AP-association.

A.2.2.3. Description of the Time Function Service Class

The Time Functions service class has been introduced to provide a network-wide timer for control, protocol and supervision purposes.

The AP-time is coded as a 48-bit binary number, whose value is relative to the starting value of the AP-time (1.1.1989 0-h, 0-m, 0-s and 0-ms). The timer resolution is the millisecond.

The most significant byte is the first one after the AP-Header, the other bytes follow in decreasing order.

The following services currently belong to the class of the Time Function service class:

- *TimeRead.* The AP-Client reads the time available from an AP-Server.
- *TimeForce.* The AP-Client transfers the unsolicited current value of the timer to the AP-Server. The AP-Server uses this value to synchronize its clock.
- *TimeWrite.* The AP-Client transfers the AP-time of its clock to the AP-Server.

A.3. Siemens Communication Processors and Supported AP Services

Although AP services are standardized and perform general-purpose functions, not all of them are implemented on the entire range of Siemens products.

This can be inferred from Table A.3, “Siemens Communication Processors and Supported AP Services” which includes the Siemens Communication Processors that currently support SINEC-AP:

- CP143A0 CP for S5 Programmable Logic Controllers
- CP231A CP for Sinumerik
- CP231B CP for Sirotec
- CP141 CP for AT-compatible PC, plus AP-Monitor software, plus TF software
- KS100 CP for Sicomp

These products currently support the AP services listed in Table A.3, “Siemens Communication Processors and Supported AP Services”, where:

Y = supported (Client and Server)

YC = supported (Client only)

YS = supported (Server only)

N = not supported.

Table A.3. Siemens Communication Processors and Supported AP Services

AP Service	Communication Processors
Environment Management CP143A0 KS100 CP231A CP231B CP141	
Initiate	Y N N N Y
Conclude	Y N N N Y
Abort	Y N N N N
VMD Support CP143A0 KS100 CP231A CP231B CP141	
Status	Y N N N Y
GetNameList	YS N N N Y
Identify	Y N N N Y
GetCapabilityList	YS N N N Y
UnsolicitedStatus	Y N N N Y
Variable Access CP143A0 KS100 CP231A CP231B CP141	
Read	Y N N N Y
Write	Y N N N Y
InformationReport	Y N N N Y
GetVariableAccessAttributes	YS N N N Y
Domain Management CP143A0 KS100 CP231A CP231B CP141	
InitiateDownloadSequence	YS N N N YC
DownloadSegment	YS N N N YC

AP Service	**Communication Processors**
TerminateDownloadSequence	YS N N N YC
InitiateUploadSequence	YS N N N YC
UploadSegment	YS N N N YC
TerminateUploadSequence	YS N N N YC
RequestDomainDownload	YS N N N YC
RequestDomainUpload	YS N N N YC
LoadDomainContent	YS N N N YC
StoreDomainContent	YS N N N N
DeleteDomain	YS N N N N
SetDomainAttributes	YS N N N YC
ProgramInvocation CP143A0 KS100 CP231A CP231B CP141	
CreateProgramInvocation	Y N N N YC
DeleteProgramInvocation	Y N N N YC
Start	Y N N N YC
Stop	Y N N N YC
Resume	Resume
Reset	Y N N N YC
Kill	Y N N N YC
GetProgramInvocationAttribute	Y N N N YC
Serial Transfer CP143A0 KS100 CP231A CP231B CP141	
ReadMessage	Y Y N N Y
WriteMessage	Y Y N N Y
ExchangeMessage	Y Y Y Y Y
Time Functions CP143A0 KS100 CP231A CP231B CP141	
TimeRead	N N YS YS YC

Appendix B. Comparison Between the MMS and the SINEC AP Models

B.1. Introduction

This appendix provides a concise comparison between the objects and services of the SINEC Automation Protocol (AP) and those of the Manufacturing Message Specification (MMS).

The VSI OSAP reader will find this information helpful when reading the VSI OMNI documentation, which uses MMS terminology.

For a formal discussion of the concepts presented here, refer to the appropriate official documentation issued by OSI and Siemens.

Note

The current version of VSI OSAP supports a subset of the objects and services defined by SINEC AP. Refer to Section 1.5.2, “Supported AP Services” for a list of supported AP objects and services.

B.2. Scope of the MMS and the AP Models

The most obvious distinction between MMS and AP lies in the different scope of the two specifications.

MMS is the ISO Standard for application level communication within the manufacturing environment. It therefore aims at maximum generalization and addresses the widest range of shop-floor equipment. The MMS specification must be unambiguous and formal.

However, AP provides a common method of communication between different Siemens automation devices. Although the Siemens offer is vast (programmable controllers, numeric control machines, personal computers and minicomputers), the scope of AP is much more limited than that of MMS. Notwithstanding its limited scope, AP does specify some services that are not part of the MMS standard, but which de- rive from functions implemented on older Siemens protocols and products.

B.3. Entities and Objects

Both MMS and AP specifications rely on a hierarchical description of the manufacturing environment. In fact, in both cases the domain of interest is structured by means of hierarchically organized entities.

Several similarities can be found in the two environment descriptions. These similarities are summarized in Table B.1, “Comparing Entities and Objects”, where entities that appear in the same row can be placed at the same hierarchical level.

Table B.1. Comparing Entities and Objects

MMS Entity/Object	AP Entity/Object
MMS Application	AP Application
MMS Server	AP Server
MMS Client	AP Client
MMS Association	AP Association
Virtual Manufacturing Device (VMD)	Virtual Manufacturing Device (VMD)

MMS Entity/Object	AP Entity/Object
Domain	Domain
Variable	Variable
Program Invocation	Program Invocation
Semaphore	(Not defined in AP)
Event	(Not defined in AP)
Journal	(Not defined in AP)
MMS Service Class	AP Service Class
MMS Service	AP Service
(Not defined in MMS)	Message

In Table B.1, “Comparing Entities and Objects” objects and entities defined by the MMS specification are mapped onto the corresponding AP objects and entities. Table rows that contain names in both columns show that the same concept is included in both specifications, and apart from the name and any limitations of AP with respect to MMS, the concepts are identical.

B.4. Services and Service Classes

The services (that is, the Technological Functions) provided by AP can be grouped into:

- **Open**: services that are "functionally compatible" with MMS, that is, they provide the same functions and capabilities (or a subset of them) as those supplied by the corresponding MMS services.
- **Not-open**: services that are only provided by SINEC-AP (not foreseen by MMS).

SINEC-AP services are continuously evolving as a result of new product needs, new application functions and requirements, and evolution of MMS. Additional service classes in the File Transfer and Directory Services areas are expected to be specified in the near future.

A comparison between AP and MMS service classes is shown in Table B.2, “Comparison between SINEC-AP and MMS Service Classes”. Table rows that contain names in both columns identify open AP services which, apart from the name and any limitations of AP with respect to MMS, are identical in both Standards.

Table B.2. Comparison between SINEC-AP and MMS Service Classes

MMS Service Class	AP Service Class
Environment and General Management	Environment Management
VMD Support	VMD Support
Variable Access Services	Variable Access Services
(Not defined in MMS)	Serial Transfer
(Not defined in MMS)	Synchronized Time
Domain Management	Domain Management
Program Invocation Management	Program Invocation Management
Event Management	(Not defined in AP)
Semaphore Management	(Not defined in AP)
Operator Communication	(Not defined in AP)
Journal Management	(Not defined in AP)
File Access	(Not defined in AP)
File Transfer	SINEC File Transfer (not corresponding to MMS File Transfer)

B.4.1. Comparison between Open AP Services and MMS Services

Open AP services include the following classes:

- Environment Management
- VMD Support Services
- Variable Access Services
- Domain Management Services
- Program Invocation Services.

Each open AP service class is a functional subset of the MMS Service class of the same name, as shown in Table B.3, “MMS Services and Open AP Services”.

Table B.3. MMS Services and Open AP Services

Environment and General Management	
Initiate	x x
Conclude	x x
Cancel	x x
Abort	x x
Reject	x
VMD Support Services	
Status	x x
GetNameList	x x
Identify	x x
GetCapabilityList	x x
UnsolicitedStatus	x x
Rename	x
Variable Access Services	
Read	x x
Write	x x
InformationReport	x x
GetVariableAccessAttribute	x x
DefineNamedVariable	x
DefineScatteredAccess	x
GetScatteredAccessAttributes	x
DeleteVariableAccess	x
DefineNamedVariableList	x
GetNamedVariableListAttributes	x
DeleteNamedVariableList	x
DefineNamedType	x
GetNamedTypeAttributes	x

DeleteNamedType	x
Domain Management Services	
InitiateDownloadSequence	x x
DownloadSegment	x x
TerminateDownloadSequence	x x
InitiateUploadSequence	x x
UploadSequence	x x
TerminateUploadSequence	
RequestDomainDownload	x x
RequestDomainUpload	x x
LoadDomainContent	x x
StoreDomainContent	x x
DeleteDomain	x x
GetDomainAttributes	x x
Program Invocation Services	
CreateProgramInvocation	x x
DeleteProgramInvocation	x x
Start	x x
Stop	x x
Resume	x x
Reset	x x
Kill	x x
GetProgramInvocationAttributes	x x

For a complete and formal description of the open AP services, refer to the appropriate Siemens documentation.

B.4.2. AP Open Classes Limitations

The following sections highlight the differences that exist between each open AP class and the corresponding class of MMS services (where applicable).

B.4.2.1. VMD Support Services

The SINEC AP definition of the VMD Support services does not currently include the Rename service.

B.4.2.2. Variable Access Services

These services are a subset of the corresponding MMS service class. MMS defines the following five variable access objects:

1. The Unnamed Variable object
2. The Named Variable object
3. The Scattered Access object
4. The Named Variable List object
5. The Named Type object.

SINEC-AP only supports objects 1) and 2).

Appendix C. ODF Predefined Types

In the OSAP context, ODF supports the predefined types listed in Table C.1, “Predefined Application Types Provided by VSI OSAP for OpenVMS”.

Table C.1. Predefined Application Types Provided by VSI OSAP for OpenVMS

Application Type Names	Type Description
OMNI$LONG	32 bit signed integer
OMNI$WORD	16 bit signed integer
OMNI$BYTE	8 bit signed integer
OMNI$ULONG	32 bit signed integer
OMNI$UWORD	16 bit signed integer
OMNI$UBYTE	8 bit signed integer
OMNI$BOOLEAN	8 bit boolean
OMNI$BIT32	32 bit bitstring
OMNI$BIT16	16 bit bitstring
OMNI$BIT8	8 bit bitstring
OMNI$F_FLOAT	single precision floating point
OMNI$NT_STR4	5 byte null-terminated string (holds 4 characters)
OMNI$NT_STR6	7 byte null-terminated string (holds 6 characters)
OMNI$NT_STR8	10 byte null-terminated string (holds 8 characters)
OMNI$NT_STR10	11 byte null-terminated string (holds 10 characters)
OMNI$NT_STR16	17 byte null-terminated string (holds 16 characters)
OMNI$NT_STR18	18 byte null-terminated string (holds 18 characters)
OMNI$NT_STR32	33 byte null-terminated string (holds 32 characters)
OMNI$NT_FIXED_STR4	5 byte null-terminated string (holds 4 characters)
OMNI$NT_FIXED_STR6	7 byte null-terminated string (holds 6 characters)
OMNI$NT_FIXED_STR8	10 byte null-terminated string (holds 8 characters)
OMNI$NT_FIXED_STR10	11 byte null-terminated string (holds 10 characters)
OMNI$NT_FIXED_STR16	17 byte null-terminated string (holds 16 characters)
OMNI$NT_FIXED_STR18	18 byte null-terminated string (holds 18 characters)
OMNI$NT_FIXED_STR32	33 byte null-terminated string (holds 32 characters)
OMNI$WC_STR4	5 byte word-counted string (holds 4 characters)
OMNI$WC_STR6	byte word-counted string (holds 6 characters)
OMNI$WC_STR8	9 byte word-counted string (holds 8 characters)
OMNI$WC_STR10	11 byte word-counted string (holds 10 characters)
OMNI$WC_STR16	17 byte word-counted string (holds 16 characters)
OMNI$WC_STR18	18 byte word-counted string (holds 18 characters)
OMNI$WC_STR32	33 byte word-counted string (holds 32 characters)
OMNI$WC_FIXED_STR4	5 byte word-counted string (holds 4 characters)
OMNI$WC_FIXED_STR6	7 byte word-counted string (holds 6 characters)
OMNI$WC_FIXED_STR8	10 byte word-counted string (holds 8 characters)

Application Type Names	Type Description
OMNI$WC_FIXED_STR10	11 byte word-counted string (holds 10 characters)
OMNI$WC_FIXED_STR16	17 byte word-counted string (holds 16 characters)
OMNI$WC_FIXED_STR18	18 byte word-counted string (holds 18 characters)
OMNI$WC_FIXED_STR32	33 byte word-counted string (holds 32 characters)
OMNI$ARRAY_64W	Array 64 of OMNI$WORDs
OMNI$ARRAY_32W	Array 32 of OMNI$WORDs
OMNI$ARRAY_16W	Array 16 of OMNI$WORDs
OMNI$ARRAY_8W	Array 8 of OMNI$WORDs
OMNI$ARRAY_4W	Array 4 of OMNI$WORDs
OMNI$ARRAY_2W	Array 2 of OMNI$WORDs
OMNI$ARRAY_32L	Array 32 of OMNI$LONGs
OMNI$ARRAY_16L	Array 16 of OMNI$LONGs
OMNI$ARRAY_8L	Array 8 of OMNI$LONGs
OMNI$ARRAY_4L	Array 4 of OMNI$LONGs
OMNI$ARRAY_2L	Array 2 of OMNI$LONGs
OMNI$ARRAY_128B	Array 128 of OMNI$BYTEs
OMNI$ARRAY_64B	Array 64 of OMNI$BYTEs
OMNI$ARRAY_32B	Array 32 of OMNI$BYTEs
OMNI$ARRAY_16B	Array 16 of OMNI$BYTEs
OMNI$ARRAY_8B	Array 8 of OMNI$BYTEs
OMNI$ARRAY_4B	Array 4 of OMNI$BYTEs
OMNI$ARRAY_2B	Array 2 of OMNI$BYTEs
OMNI$TIME_OF_DAY4	4 octet string
OMNI$TIME_OF_DAY6	6 octet string