BASEstar Open Device Connectivity

Device Access Software for Allen-Bradley INTERCHANGE User Guide

Order Number: AA-QWJZB-TE

April 1997

This document describes the Device Access Software (DAS) for Allen-Bradley INTERCHANGE which allows BASEstar Open users and DEComni API applications to exchange data with Allen-Bradley processors supporting the Allen-Bradley INTERCHANGE protocol.

Revision/Update Information:	This is a new documentfor the current release.
Operating System and Version:	BASEstar Open is available on a broad range of both hardware and software platforms. Refer to your *Software Product Description* for precise information.
Software Version:	3.1

Digital Equipment Corporation
Maynard, Massachusetts

First printing, February 1996
Revised, July 1996
Revised, April 1997

© Digital Equipment Corporation 1997.

All Rights Reserved.

Possession, use, or copying of the software described in this documentation is authorized only pursuant to a valid written license from DIGITAL or an authorized sublicensor.

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: Alpha AXP, BASEstar Open, DEC, DECmessageQ, DECnet, DECnet-DOS, DECosap, DEComni, DIGITAL, DIGITAL UNIX, FMS, LN03, MicroVAX, NAS, OpenVMS, OpenVMS Alpha, PATHWORKS, PDAS, Rdb/VMS, ReGIS, ThinWire, TK, ULTRIX, VAX, VAXcluster, VAX COBOL, VAX FORTRAN, VAX Pascal, VAX RMS, VMS/ULTRIX Connection, VT, and the DIGITAL logo.

The following are third-party trademarks:

INTERCHANGE is a trademark of Allen-Bradley Corporation.
PLC is a registered trademark of Allen-Bradley Corporation.
MS, Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation.
UNIX is a registered trademark licensed exclusively by X/Open Company Ltd.
Windows and Windows NT are trademarks of Microsoft Corporation.

All other trademarks and registered trademarks are the property of their respective holders.

Table of Contents

Preface..... vii

1 Introduction 1-1

Communicating With Allen-Bradley PLC Processors..... 1-1
Features of the DIGITAL DAS for Allen-Bradley INTERCHANGE 1-2
What's Next?..... 1-2

2 Installing the DAS for Allen-Bradley INTERCHANGE on Your Platform..... 2-1

Installation on DIGITAL UNIX Systems 2-2
Preparing for DAS for Allen-Bradley INTERCHANGE Installation..... 2-2
Installing the Device Access Software for Allen-Bradley INTERCHANGE..... 2-4
Files Installed on Your System..... 2-5
Installation on OpenVMS Systems..... 2-6
Preparing for DAS for Allen-Bradley INTERCHANGE Installation..... 2-6
Installing the Device Access Software for Allen-Bradley INTERCHANGE..... 2-7
Files Installed on Your System..... 2-7

PART I Using BASEstar Open to Communicate With Allen-Bradley INTERCHANGE Processors

3 Overview..... 3–1

BASEstar Open Environment Components..... 3–2
Writing and Reading Device Memory Locations..... 3–4
Receiving Unsolicited Messages 3–5

4 Configuring BASEstar Open 4–1

Registering ODS Entries..... 4–2
Creating Application Services Objects 4–2
Creating Device Services Objects..... 4–2
Creating Protocol_Profiles 4–2
Creating VMDs 4–3
Creating Variables..... 4–4
Creating Data Services Objects 4–6
Creating Device_Data_Points for Read and Write Operations 4–6
Creating Device_Data_Points for Receiving Unsolicited Messages..... 4–7

5 A Complete Example 5-1

Example Overview 5-1
Before Running the Example 5-1
CLI Source 5-1

PART II Using DEComni API to Communicate With Allen-Bradley INTERCHANGE Processors

6 Overview 6-1

DEComni Environment Components 6-2
Writing and Reading Device Memory Locations 6-3
Receiving Unsolicited Messages 6-4

7 Configuring DEComni API 7-1

Registering ODS Entries 7-2
Setting the Application Profile 7-2
Creating VMD Definitions 7-2
Creating Variable Definitions 7-3
Creating Unnamed_Variables for Read and Write Operations 7-3
Creating Unnamed_Variables for Receiving Unsolicited Messages 7-3
Specifying the Maximum Number of Variables 7-4

8 Programming 8-1

DEComni API Procedures for Data Exchange 8-2
Opening and Closing a Connection 8-2
Writing and Reading Device Memory Locations 8-2
Receiving Unsolicited Messages 8-2

PART III Appendices

A ODS Entry Format for the Allen-Bradley INTERCHANGE Protocol A-1

B Mapping BASEstar Open Datatypes B-1

C Mapping DEComni API Types C-1

D DEComni API Network IOSB Messages D-1

Figures

Figure 1–1: Software Components for INTERCHANGE Communication 1-1
Figure 1–2: Communicating With Allen-Bradley INTERCHANGE PLC Processors 1-2
Figure 3–1: BASEstar Open Environment Components 3–2
Figure 3–2: Writing and Reading Device Memory Locations 3–4
Figure 3–3: Receiving Unsolicited Messages 3–5
Figure 6–1: DEComni API Environment Components 6-2
Figure 6–2: Writing and Reading Device Memory Locations 6-3
Figure 6–3: Receiving Unsolicited Messages 6-4

Tables

Table 2–1: Pre-requisite Software for DIGITAL UNIX (BASEstar Open Environment) 2-3
Table 2–2: Pre-requisite Software for DIGITAL UNIX (DEComni API Environment) 2-3
Table 2-3: Pre-requisite Software for OpenVMS (BASEstar Open Environment) 2-6
Table 2-4: Pre-requisite Software for OpenVMS (DEComni API Environment) 2-7
Table 4–1: Device Services Protocol_Profile Attributes 4–2
Table 4–2: Device Services VMD Attributes 4–3
Table 4–3: Device Services Variable Attributes 4–4
Table 4–4: Device_Data_Point Attributes for Read and Write Operations 4–6
Table 4–5: Device_Data_Point Attributes for Unsolicited Messages 4–7
Table 7–1: DEComni VMD Attributes 7-2
Table 7–2: DEComni Variable Attributes 7-3
Table 8-1: DEComni API Procedures for Data Exchange 8-2
Table A–1: Values of ODS Attributes for Allen-Bradley Devices A-1
Table B–1: BASEstar Open Simple Datatype Size B-1
Table B–2: Mapping Allen-Bradley Types to BASEstar Open Datatypes B-2
Table C–1: DEComni Basic MMS Type Sizes C-1
Table C–2: Mapping Allen-Bradley Types to DEComni Types C-2

Examples

Example 4–1: Creating Device Services Protocol_Profiles 4–3
Example 4–2: Creating a Device Services VMD 4–3
Example 4–3: Creating an Unnamed_Variable for Read and Write Operations 4–4
Example 4–4: Creating an Unnamed_Variable for Receiving Unsolicited Messages 4–5
Example 4–5: Creating a Device_Data_Point for Read and Write Operations 4–6
Example 5-1: Creating BASEstar Open Objects 5-2
Example 5-2: Reading and Writing the Memory of a Device 5-4
Example 5-3: Getting the Value Carried by an Unsolicited Write Command 5-4
Example 7-1: Setting the Maximum Number of Variables (DIGITAL UNIX) 7-4
Example 7-2: Setting the Maximum Number of Variables (OpenVMS) 7-4
Example A-1: An ODS Entry for a Device Connected Directly to Ethernet A-1
Example A-2: An ODS Entry for a Device Connected by a DH or DH+ Link A-2

Preface

This document describes how to install and use the DIGITAL Device Access Software (DAS) for Allen-Bradley INTERCHANGE, the software that allows BASEstar Open users and DEComni API applications to exchange data with Allen-Bradley processors supporting the INTERCHANGE software.

Multiplatform Applicability

Most of the information in this manual applies regardless of the operating system platform on which the DAS for Allen-Bradley INTERCHANGE software is installed. Information relating to installation is typically platform-dependent.

Intended Audience

This manual is addressed to:

- System managers who have to install and maintain the DAS for Allen-Bradley INTERCHANGE on any of the supported platforms
- Application developers who have to configure and use BASEstar Open, or the DEComni API (or both) to exchange data with Allen-Bradley processors.

Structure of this Document

This document is organized as follows:

- Chapter 1 introduces you to the DAS for Allen-Bradley INTERCHANGE; Chapter 2 contains the installation procedures.
- Part I describes how Allen-Bradley processors are accessed through BASEstar Open services, how you must configure BASEstar Open for use with the DAS for Allen-Bradley INTERCHANGE, and finally provides a complete code example that you can run in the BASEstar Open environment.
- Part II describes how Allen-Bradley processors are accessed through the DEComni API, how you must configure DEComni for use with the DAS for Allen-Bradley INTERCHANGE, and how you exchange data with the remote processors.
- Part III contains the appendices.

For More Information

If you are using BASEstar Open services you will find related information in the following documents:

- *BASEstar Open Introduction*
- *BASEstar Open Reference Guide*
- *BASEstar Open Command Language Interface*
- *BASEstar Open Application Programming Interface*
- *BASEstar Open Messages*
- Platform-specific management guides

If you are using DEComni API you will find related information in the following documents:

- *DEComni API and DEComni MMS User Guide*
- *DEComni API Guide to Using Omni Directory Services* (for supported platforms)
- *DEComni API Omni Definition Facility* User Guide (for supported platforms)
- *DEComni API Guide to Using OmniView* (for supported platforms)

1
Introduction

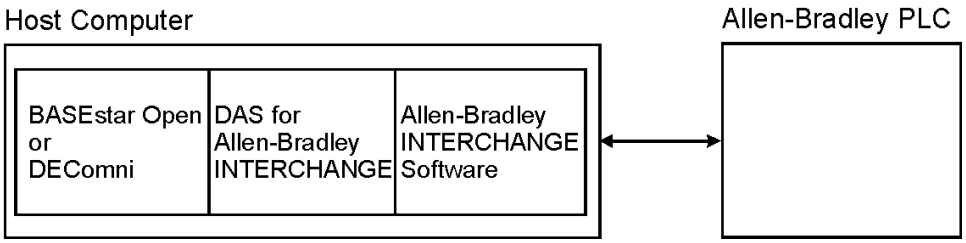
The DIGITAL Device Access Software (DAS) for Allen-Bradley INTERCHANGE allows both BASEstar Open users and DEComni API applications to access data-table memory or shared-data memory on Allen-Bradley PLC processors supporting the INTERCHANGE software.

Communicating With Allen-Bradley PLC Processors

Figure 1–1 shows how a system, equipped with the BASEstar Open or the DEComni API product, can communicate as a host computer with an Allen-Bradley PLC processor using the Allen-Bradley INTERCHANGE software.

The host computer is an OpenVMS or DIGITAL UNIX system.

Figure 1–1: Software Components for INTERCHANGE Communication



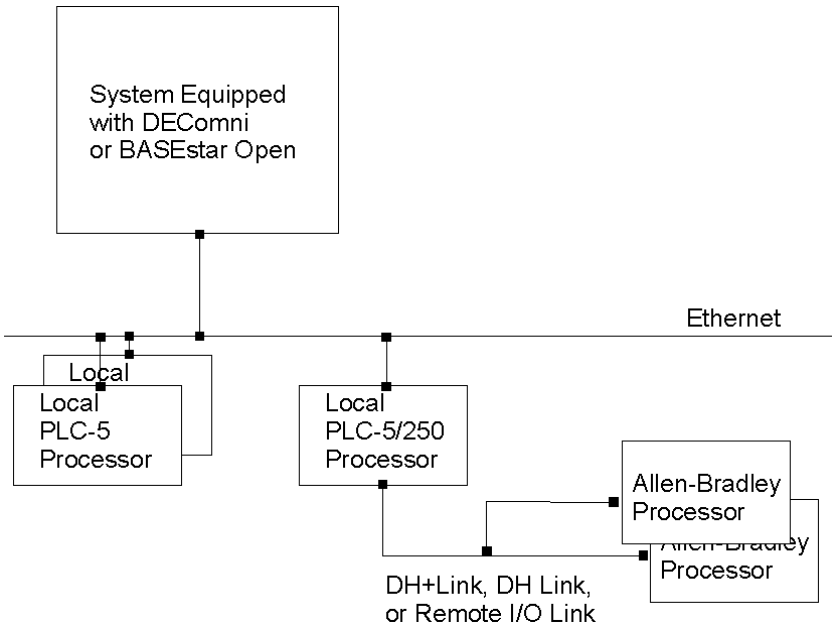
The Allen-Bradley INTERCHANGE software is a library of functions and executable commands that access data-table memory or shared-data memory on Allen-Bradley PLC processors. Allen-Bradley provides INTERCHANGE software versions for the most common platforms on the market, including OpenVMS and DIGITAL UNIX.

The DIGITAL DAS for Allen-Bradley INTERCHANGE is a BASEstar Open device connectivity module that adapts the application programming interface provided by the Allen-Bradley INTERCHANGE software to BASEstar Open or DEComni, thus allowing BASEstar Open users or DEComni applications to exchange data with Allen-Bradley PLC processors.

As illustrated in Figure 1–2, the DIGITAL host computer (and thus BASEstar Open users or DEComni applications), can communicate with:

- PLC-5 and PLC-5/250 stations connected to the Ethernet network
- PLC-2, PLC-3, 1785 PLC-5 and PLC-5/250 stations connected by a DH or DH+ link to a PLC-5/250 station connected to the Ethernet network

Figure 1–2: Communicating With Allen-Bradley INTERCHANGE PLC Processors



Features of the DIGITAL DAS for Allen-Bradley INTERCHANGE

The DIGITAL DAS for Allen-Bradley INTERCHANGE supports the following INTERCHANGE communication functions:

- Sending read and write commands
- Receiving unsolicited messages from the device

What's Next?

In this chapter, you have learned about the features of the DIGITAL DAS for Allen-Bradley INTERCHANGE.

If you intend to use the DAS for Allen-Bradley INTERCHANGE through BASEstar Open services, read Chapter 2 for information about how to install the DAS on your system. Then read Part I of this manual for details of how to configure BASEstar Open and use its services to exchange data with Allen-Bradley processors.

If you intend to use the DAS for Allen-Bradley INTERCHANGE through the DEComni API, read Chapter 2 for information about how to install the DAS on your system. Then read Part II of this manual for details of how to configure the DEComni API and use the API procedures to exchange data with Allen-Bradley processors.

2

Installing the DAS for Allen-Bradley INTERCHANGE on Your Platform

This chapter specifies the requirements and the procedures for installing the DAS for Allen-Bradley INTERCHANGE on the following platform:

- DIGITAL UNIX
- OpenVMS

It also provides the instructions for installing the DAS for Allen-Bradley INTERCHANGE in the BASEstar Open and DEComni API environments.

Installation on DIGITAL UNIX Systems

The installation requirements vary according to whether you are installing the DAS for Allen-Bradley INTERCHANGE for use via BASEstar Open services or via the DEComni API.

Preparing for DAS for Allen-Bradley INTERCHANGE Installation

Before starting the installation procedure, complete the preparation requirements outlined in this section.

License Registration

The Device Access Software for Allen-Bradley INTERCHANGE includes support for the DIGITAL UNIX License Management Facility (LMF). A License Product Authorization Key (License PAK) must be registered in the License Database (LDB) in order to use install the Device Access Software for Allen-Bradley INTERCHANGE on a newly-licensed node. The License PAK may be shipped along with the kit if you ordered the license and media together; otherwise, it is shipped separately to a location based on your license order.

If you are installing the Device Access Software for Allen-Bradley INTERCHANGE as an update on a node already licensed for this software, you have already completed the License PAK registration requirements.

If you are installing pre-requisite or optional software along with the Device Access Software for Allen-Bradley INTERCHANGE , review the PAK status and install the PAKs for any pre-requisite or optional software before you install the Device Access Software for Allen-Bradley INTERCHANGE

To register a license under the DIGITAL UNIX system, first log in as superuser.

You then have a choice of two ways to perform the PAK registration in the License Database (LDB):

- Before installing the Device Access Software for Allen-Bradley INTERCHANGE

  At the superuser prompt, edit an empty PAK template with the `lmf register` command and include all the information on your License PAK as follows:

  ```
  # lmf register
  ```

- After installing the Device Access Software for Allen-Bradley INTERCHANGE

  At the superuser prompt, edit the partially completed PAK template in `/usr/var/adm/lmf/`*`template`* with the `lmf register` command to add your unique License PAK information as follows:

  ```
  # lmf register - < /usr/var/adm/lmf/DASAB-INTER
  ```

  After you register your license, use the following `lmf reset` command to copy the license details from the License Database (LDB) to the kernel cache:

  ```
  # lmf reset
  ```

For complete information on using the DIGITAL UNIX License Management Facility, see the *DIGITAL UNIX Guide to Software Licensing* or the `lmf(8)` reference page.

Software Requirements (BASEstar Open Environment)

If you are installing the DAS for Allen-Bradley INTERCHANGE in the BASEstar Open environment, check that the software products listed in Table 2–1 have been installed on your system.

Table 2–1: Pre-requisite Software for DIGITAL UNIX (BASEstar Open Environment)

Subset Name and Description	Comments
DTLEISW060 Allen-Bradley INTERCHANGE Software Version 6.0 or higher	This is an Allen-Bradley product that must be separately ordered from Allen-Bradley.
BSTR310 BASEstar Open Server for DIGITAL UNIX Version 3.1	Also ensure that the BASEstar Open Device Connectivity (DEComni) has been installed. In particular, check for the presence of the following subsets: • DOUBASE310 • DASTKRUN310 See the *BASEstar Open Server Inst. & Management Guide* for your platform for details.

Software Requirements (DEComni API Environment)

Check that the software products listed in Table 2–2 have been installed and are functioning on your system.

Table 2–2: Pre-requisite Software for DIGITAL UNIX (DEComni API Environment)

Subset Name and Description	Comments
Allen-Bradley INTERCHANGE Software Version 6.0 or higher	This is an Allen-Bradley product that must be separately ordered from Allen-Bradley.
DEComni API for DIGITAL UNIX Version 3.1	In particular, check for the presence of the following subsets: • DASTKRUN310 See the *DEComni API Installation Guide* for your platform for details.

Disk Space Requirements

The following table lists the disk space required to install the DAS for Allen-Bradley INTERCHANGE.

Usage (Kbytes)	(root)	/usr/opt	/var/opt
Installation	0	1103	381
Permanent	0	1103	339

Backing Up Your System Disk

DIGITAL recommends that you back up your system disk before installing any software. For details of how to perform a system disk backup, refer to your DIGITAL UNIX documentation.

Installing the Device Access Software for Allen-Bradley INTERCHANGE

This section describes the installation, deinstallation and IVP procedures for the Device Access Software for Allen-Bradley INTERCHANGE for DIGITAL UNIX. Installing the Device Access Software for Allen-Bradley INTERCHANGE and running the Installation and Verification Procedure (IVP) on your DIGITAL UNIX system takes approximately 5 to 10 minutes.

Installation Procedure

Install the DAS for Allen-Bradley INTERCHANGE as follows:

1. Log onto the system as superuser (root).
2. Mount the distribution media on the desired location (for example, mount `-dr /dev/rz4c /CDROM`).
3. Change your working directory to the kit location (for example, `cd /CDROM/DASDTL310`).
4. Issue the *`setld -l`* command to load the product onto the system, and follow the installation dialogs.

Running the IVP

During the installation procedure, you can run the IVP as part of the installation. If you did not run the IVP at that time or the product does not work correctly, you can run the IVP at any time by issuing the following command:

```
setld -v DASDTL310
```

Deinstallation Procedure

You can deinstall the DAS for Allen-Bradley INTERCHANGE as follows:

1. Log onto the system as superuser (root).
2. Change your working directory to the kit location (for example, `cd /CDROM/DASDTL310`).
3. Issue the *`setld -d DASDTL310`* command.

Files Installed on Your System

This section lists directories and files created by the Device Access Software for Allen-Bradley INTERCHANGE.

./usr/opt/dasdtl_310/doc:

dasdtl_guide.ps

relnotes.ps

relnotes.txt

./usr/opt/dasdtl_310/include:

omni_integrator9_defs_include.h

./usr/opt/dasdtl_310/lib:

libdasdtl.a

./usr/opt/dasdtl_310/shlib:

libdasdtl.so

./usr/var/opt/dasdtl_310/examples:

dasdtl_ivp

dasdtl_ivp.c

runivp_dtl

Installation on OpenVMS Systems

The installation requirements vary according to whether you are installing the DAS for Allen-Bradley INTERCHANGE for use via BASEstar Open services or via the DEComni API.

The information that follows is valid for both Alpha and VAX architectures unless otherwise stated.

Preparing for DAS for Allen-Bradley INTERCHANGE Installation

Installing the Device Access Software for Allen-Bradley on your OpenVMS system takes approximately 3 minutes.

Before starting the installation procedure, complete the preparation requirements outlined in this section.

License Registration

You must use the License Management Facility (LMF) to register the Device Access Software for Allen-Bradley INTERCHANGE in accordance with the license agreement signed for your site before you can run either the Installation Verification Procedure (IVP) or the software. License registration information is included in the Product Authorization Key (PAK) that is shipped with the Device Access Software for Allen-Bradley INTERCHANGE. The PAK contains licensing information that should be registered *before* you start the installation.

To register the license, log into the SYSTEM account and set your default to SYS$UPDATE. You can use either of the following two methods to register the license under LMF:

- Invoke the following procedure and enter the data supplied by the PAK:

  ```
  $ SYS$UPDATE:VMSLICENSE.COM
  ```

- Enter the following command, using the qualifiers that specify the data supplied by the PAK :

  ```
  $LICENSE REGISTER DASAB-INTER /qualifier,...
  ```

If you plan to use the Device Access Software for Allen-Bradley INTERCHANGE on more than one node of a VAXcluster system, you must load the license on each of the othe nodes after you have completed this one.

For complete information on using the OpenVMS License Management Facility, refer to the *OpenVMS License Management Utility* Manual.

Software Requirements (BASEstar Open Environment)

If you are installing the DAS for Allen-Bradley INTERCHANGE in the BASEstar Open environment, check that the software products listed in Table 2-3 have been installed correctly on your system.

Table 2-3: Pre-requisite Software for OpenVMS (BASEstar Open Environment)

Subset Name and Description	Comments
DTLNETV2060 Allen-Bradley INTERCHANGE Network DTL V Software Version 6.0	This is an Allen-Bradley product that must be separately ordered from Allen-Bradley.
BASEstar Open Server for OpenVMS Version 3.1	Also ensure that the BASEstar Open Device Connectivity (DEComni API) has been installed. See the *BASEstar Open Server Installation and Management Guide* for your platform for details.
DEComni API for OpenVMS Version 3.1, including DASware	See the *DEComni API Installation Guide* for your platform for details.

Software Requirements (DEComni API Environment)

Check that the software products listed in Table 2-4 have been installed on your system.

Table 2-4: Pre-requisite Software for OpenVMS (DEComni API Environment)

Subset Name and Description	Comments
DTLNETV2060 Allen-Bradley INTERCHANGE Network DTL V Software Version 6.0	This is an Allen-Bradley product that must be separately ordered from Allen-Bradley.
DEComni API for OpenVMS Version 3.1, including DASware	See the *DEComni API Installation Guide* for your platform for details.

Disk Space Requirements

The DAS for Allen-Bradley INTERCHANGE on OpenVMS occupies approximately 500 blocks.

Backing Up Your System Disk

DIGITAL recommends that you back up your system disk before installing any software. For information about how to perform a system disk backup, refer to your OpenVMS documentation.

Installing the Device Access Software for Allen-Bradley INTERCHANGE

Use the following command to install the DAS for Allen-Bradley INTERCHANGE:

```
$ @SYS$UPDATE:VMSINSTALL DASDTLA031 ddcu
```

where *ddcu* is installation directory.

Files Installed on Your System

This section lists the directories and files created by the Device Access Software for Allen-Bradley INTERCHANGE.

SYS$COMMON:[SYSLIB]

- DASDTL_SHR.EXE
- DASDTL_STARTUP.COM
- OMNI_INTEGRATOR9_DEFS_INCLUDE.H
- OMNI_INTEGRATOR9_DEFS_INCLUDE.ADA
- OMNI_INTEGRATOR9_DEFS_INCLUDE.BAS
- OMNI_INTEGRATOR9_DEFS_INCLUDE.PAS
- OMNI_INTEGRATOR9_DEFS_INCLUDE.FOR
- OMNI_INTEGRATOR9_DEFS_INCLUDE.MAR
- OMNI_INTEGRATOR9_DEFS_INCLUDE.R32
- OMNI_INTEGRATOR9_DEFS_INCLUDE.PLI

SYS$COMMON:[SYSMGR]

- DASDTL_STOP.COM

SYS$COMMON:[SYSTEST]

- DASDTL_IVP.COM

DASDTL_IVP.EXE

Getting Help and Reporting Problems

If an error occurs while you are using the DAS for Allen-Bradley INTERCHANGE, and you believe the error is the result of a problem associated with the product, take one of the following actions:

- If you have a basic or DECsupport Software Agreement, call your Customer Support Center (CSC). The CSC provides telephone support for high-level advisory and remedial assistance.
- If you have a Self-Maintenance Software Agreement, you can submit a Software Performance Report (SPR).
- If you purchased the product within the last 90 days and you think the problem is casued by a software error, you can submit an SPR.

If you submit an SPR, please take the following steps:

1. Describe as accurately as possible the circumstances and state of the system when the problem occurred. Include the description and version number of the product. Demonstrate the problem with specific examples.
2. Reduce the problem to as small a size as possible.
3. Remember to include listings of any command files, include files, relevant data files and so forth.
4. Provide a list of the program.
5. If the program is longer than 50 lines, submit a copy of it on machine-readable media (floppy diskette or magnetic tape). If necessary, also submit a copy of the program library used to build the application.
6. Report only one program per SPR. This will facilitate a faster response.
7. Mail the SPR package to DIGITAL.

   Experience shows that many SPRs do not contain enough information to duplicate or identify the problem. Concise, complete information helps DIGITAL give timely and accurate service to software problems.

   If you find an error in the documentation, send an electronic mail message to *manufacturing@digital.com*.

PART I

Using BASEstar Open to Communicate With Allen-Bradley INTERCHANGE Processors

This part describes how to access Allen-Bradley processors via BASEstar Open services and how to configure BASEstar Open for use with the DAS for Allen-Bradley INTERCHANGE. It also provides a complete code example that you can run in the BASEstar Open environment.

The Command Language Interface (CLI) allows you to perform operations on selected BASEstar Open objects. However, you can use either the Application Programming Interface (API) or the Graphical Configuration Utility (GCU) to perform the same tasks.

3
Overview

This chapter describes the BASEstar Open environment components involved when you access Allen-Bradley PLC processors via BASEstar Open services.

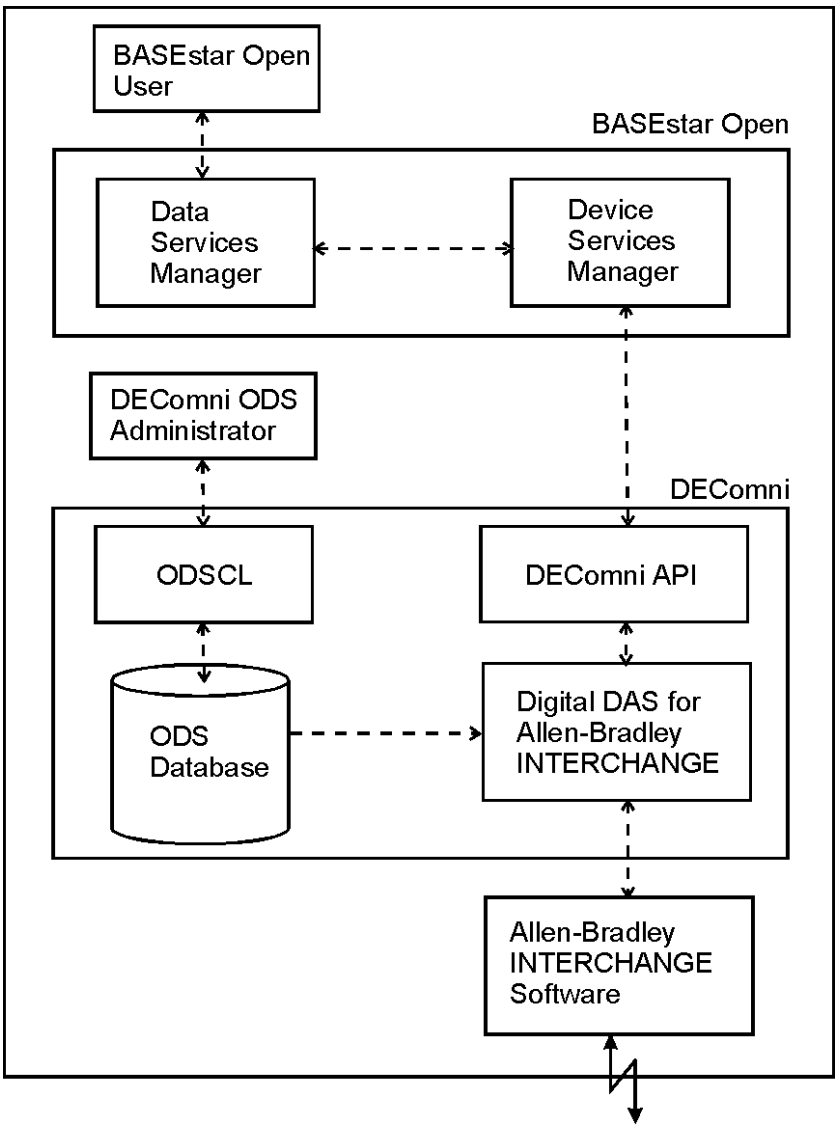
Using the features provided by the DAS for Allen-Bradley INTERCHANGE, a BASEstar Open user can issue the appropriate commands to:

- Perform read and write operations
- Receive unsolicited messages from a device.

BASEstar Open Environment Components

Figure 3–1 shows the environment components that are involved when BASEstar Open users exchange data with Allen-Bradley PLC processors.

Figure 3–1: BASEstar Open Environment Components



Data Services

BASEstar Open users use Device_Data_Points to read and write memory locations of an Allen-Bradley device. When a user writes the value of a Device_Data_Point, BASEstar Open Device Services automatically interacts with the Allen-Bradley INTERCHANGE software which, in turn, sends a message to the appropriate Allen-Bradley PLC processor. Vice versa, BASEstar Open updates the value of a Device_Data_Point each time it receives an unsolicited message from an Allen-Bradley device.

Each Device_Data_Point is linked to a Device Services variable which, in turn, is associated with the VMD definition that models the target PLC.

Device Services

BASEstar Open Device Services provides a unique set of operations to BASEstar Open users for device-independent access to any type of device or network. This device-independent approach to device connectivity is possible because Device Services models a device as a MMS Virtual Manufacturing Device (VMD).

Each Device Services VMD corresponds to an Allen-Bradley PLC processor. A variable associated with this VMD corresponds to a memory location on the same processor.

DEComni API

The DEComni API is the engine that provides BASEstar Open Device Services with MMS functions. DEComni API also allows new DAS modules to be added to the BASEstar Open environment in a standardized way.

The Omni Directory Services (ODS) database is the DEComni component that allows you to define the information required for addressing an Allen-Bradley PLC processor connected to the Ethernet network.

DIGITAL DAS for Allen-Bradley INTERCHANGE

The DIGITAL DAS for Allen-Bradley INTERCHANGE is a BASEstar Open device connectivity module that adapts the application programming interface provided by the Allen-Bradley INTERCHANGE software to BASEstar Open, thus allowing BASEstar Open users to exchange data with Allen-Bradley PLC processors.

Allen-Bradley INTERCHANGE Software

This is an Allen-Bradley library of functions and executable commands for accessing data-table memory or shared-data memory on Allen-Bradley PLC processors.

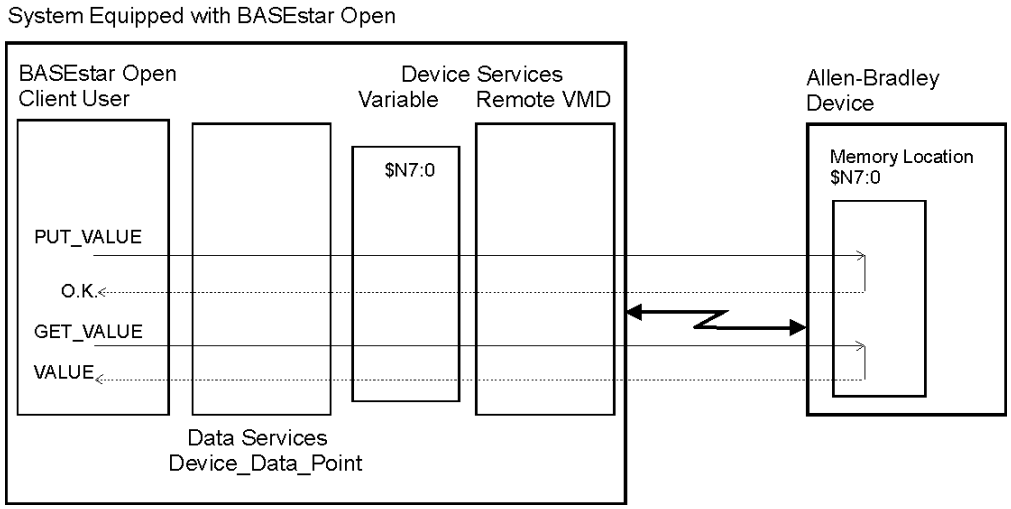
Writing and Reading Device Memory Locations

To write data to (or read data from) a physical memory location of an Allen-Bradley device, a BASEstar Open user simply puts (or gets) the value of a Device_Data_Point. As shown in Figure 3–2, this Device_Data_Point must be linked to an Unnamed_Variable which, in turn, is associated with a VMD object that models the remote device.

An Unnamed_Variable configured for read and write operations maps the remote device memory location whose address is specified as an appropriate attribute of the variable itself.

A GET_VALUE command reads data from the device memory, and returns in the user buffer a number of characters that is equal to the size of the BASEstar Open DataType associated with the variable used for the read operation. A similar consideration applies to a PUT_VALUE command.

Figure 3–2: Writing and Reading Device Memory Locations



To obtain a BASEstar Open configuration that is appropriate for your needs, you must consider the requirements of your overall application. Figure 3–2 illustrates an example in which BASEstar Open operations are synchronous with user's actions; in fact:

- Each time the user issues a PUT_VALUE command, BASEstar Open immediately updates the device memory.
- Each time the user issues a GET_VALUE command, BASEstar Open immediately reads the device memory.

However, you can configure BASEstar Open for asynchronous operations by means of a Polling_Set. In this case, BASEstar Open automatically reads the value of the device memory, and updates the value of the Device_Data_Point asynchronously with respect to the user's actions.

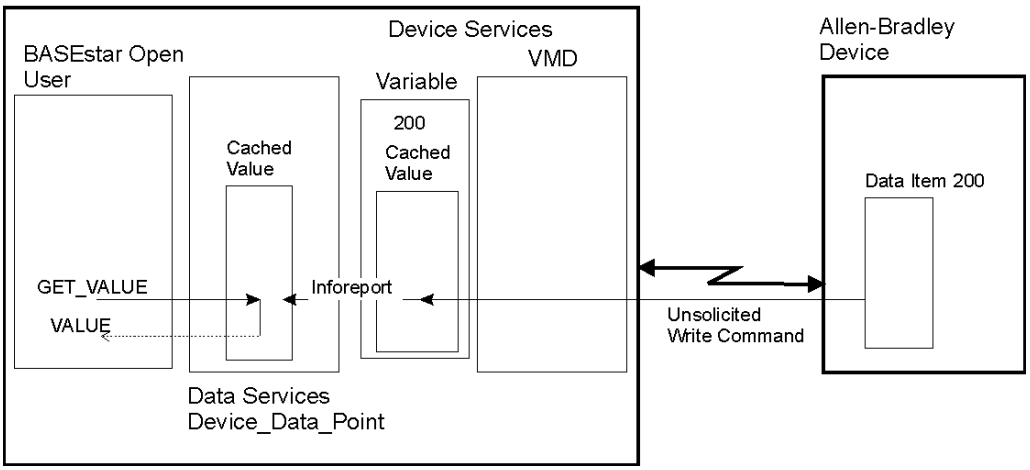
Receiving Unsolicited Messages

Figure 3–3 shows which BASEstar Open objects and commands a BASEstar Open user must use to receive an unsolicited message from a device.

To receive an unsolicited message, a BASEstar Open user must issue a GET_VALUE command on a Device_Data_Point. The Device_Data_Point must have been configured for asynchronous receive operations, and must be linked to a Device Services variable that is associated with a VMD defined for that device. The GET_VALUE command returns to the user the cached value of the Device_Data_Point itself.

BASEstar Open updates the value of the Device_Data_Point in a way that is completely independent of the user's operations: when receiving an unsolicited message from the device, BASEstar Open generates an internal Inforeport which causes the unsolicited updating of the cached value of the Device_Data_Point that is linked to a Device Services Unnamed_Variable.

Figure 3–3: Receiving Unsolicited Messages



An Unnamed_Variable object that is used for receiving unsolicited messages must specify the identifier of an unsolicited data item configured at the addressed Allen-Bradley device.

BASEstar Open interprets the data received from the device in an unsolicited write command according to the datatype associated with the Device Services Unnamed_Variable in use. The size (in bytes) of this datatype must therefore be equal to the size (in bytes) of the device memory location that is associated with the unsolicited data item. If an insufficient number of bytes is received, the value returned in the user buffer is unpredictable; if, instead, the number of the received bytes exceeds that required to meet the datatype, the exceeding bytes are lost.

4
Configuring BASEstar Open

This chapter describes how you must configure BASEstar Open in order to allow users to access an Allen-Bradley processor.

To configure BASEstar Open, you must:

- Register the appropriate entries in the ODS database.
- Create the required Application Services Activity and Program objects.
- Create the required Device Services Protocol_Profile, VMD and Variable objects.
- Create the required Device_Data_Points objects.

Note

For an explanation of BASEstar Open object attributes whose values are not specified here, refer to the *BASEstar Open Command Language Interface* and to the *BASEstar Open Reference Guide*.

Registering ODS Entries

You must create an ODS entry for each Allen-Bradley INTERCHANGE processor, specifying INTERCHANGE protocol-specific information. See Appendix A for details.

Creating Application Services Objects

You can use the DATADEV server or separate Device and Data servers. The DATADEV server makes both Data and Device Services servers available in a single process, thus simplifying configuration operations, optimizing performance, and allowing the same Device_Data_Point to be used for both read and write operations.

Note

The DATADEV server is used throughout this manual, therefore you should modify the information and examples provided in this chapter if you want to use two separate Device and Data Services servers (for example, if your BASEstar Open version does not support DATADEV servers).

See also Chapter 5 for examples showing the commands used to create the required Program and Activity objects. For further details, refer to the *BASEstar Open Command Language Interface* and to the *BASEstar Open Reference Guide*.

Creating Device Services Objects

You must create the following Device Services objects, in the order listed below:

1. Protocol_Profiles
2. VMDs
3. Variables

Creating Protocol_Profiles

Device Services uses Protocol_Profiles to associate a VMD with the appropriate DAS for Allen-Bradley INTERCHANGE.

Table 4–1 lists the Protocol_Profile attributes whose values are specific to the DAS for Allen-Bradley INTERCHANGE.

Table 4–1: Device Services Protocol_Profile Attributes

Attribute Name	Description and Values
-APPLPROFID	The identifier of the DAS to be used. Must be set to 81 to indicate the DAS for Allen-Bradley INTERCHANGE.
-MAXPDUSIZE	The maximum length (in bytes) of the data exchanged with the Allen-Bradley device, as follows: 2048 for PLCs directly connected to the Ethernet network 242 for PLCs connected to a DH or DH+ link.

Example 4–1 shows how to create the Protocol_Profiles for all the supported protocols.

Example 4–1: Creating Device Services Protocol_Profiles

```
BSTR> CREATE PROTOCOL_PROFILE PP_INTERCHANGE_ETH -APPLPROFID 81\
      -MAXPDUSIZE 2048 -LOG

BSTR> CREATE PROTOCOL_PROFILE PP_INTERCHANGE_DH -APPLPROFID 81\
      -MAXPDUSIZE 242 -LOG
```

PP_INTERCHANGE_ETH is a Protocol_Profile valid for PLCs directly connected to the Ethernet network, whereas PP_INTERCHANGE_DH is a Protocol_Profile valid for PLCs connected by a DH or DH+ link to the Ethernet network.

Creating VMDs

A Device Services VMD models an Allen-Bradley device. Table 4–2 lists the mandatory attributes that you must specify for a correct definition of an Allen-Bradley VMD. The same VMD can be used for read and write operations, and for receiving unsolicited messages from the device.

Table 4–2: Device Services VMD Attributes

Attribute Name	Description and Values
-DVM_ACCESS_POINT	A list of one or more access point descriptors, for each access point users use to access the remote device. An access point descriptor has the following format: application_simple_name:protocol_profile_name where: *application_simple_name* must specify an ODS entry valid for Allen-Bradley devices (see Appendix A). *protocol_profile_name* must specify a Protocol_Profile created according to the explanations provided in this chapter.
-MODEL	A string that specifies the type of the PLC processor at the remote station. Allowed values for this attribute are: PLC_5_250, PLC5250 PLC_5_40, PLC540 PLC_3, PLC3 PLC_2, PLC2 SLC_500, SLC500

The VMD_INTERCHANGE VMD created in Example 4–2 is associated with a PLC directly connected to the PLC network.

Example 4–2: Creating a Device Services VMD

```
BSTR> CREATE VMD VMD_INTERCHANGE\

      -MODEL "PLC_5"     -VENDOR "BASEstar Open"\
      -DVM_ACCESS_POINTS ("/cn=ODS_INTER003":PP_INTERCHANGE_ETH)\

      -DESCRIPTION "To access the 003 Allen-Bradley device directly
      connected to the Ethernet network"\
      -LOG
```

Creating Variables

Configuration of Device Services Unnamed_Variables change depending on whether you are using them for read and write operations, or for receiving unsolicited messages from the device.

Table 4–3 lists the attributes that you must specify for a correct definition of a Device Services variable.

Table 4–3: Device Services Variable Attributes

Attribute Name	Description and Values
-DATATYPE *datatype_name*	Name of the DataType the DAS for Allen-Bradley INTERCHANGE associates with the data received from the device. *Appendix B* specifies the sizes of the basic BASEstar Open DataTypes.
-SEND *notification*	For variables used in read and write operations, the value assigned to this attribute depends on configuration choices.
	For variables used for receiving unsolicited messages, this attribute must be set to ALWAYS or ON_CHANGE.
-ACCESS *access_type*	For variables used in read and write operations, this attribute must be set to WRITE (which allows bi-directional access to the device memory location).
	For variables used for receiving unsolicited messages, this attribute must be set to READ.
-ADDRTYPE *address_type*	For variables used in read and write operations, this attribute must be set to UNCONSTRAINED.
	For variables used for receiving unsolicited messages, this attribute must be set to NUMERIC.
-ADDRESS *address*	The value that you must provide depends on the type of operation.
	See the following two sections for details.

Creating Unnamed_Variables for Read and Write Operations

To define an Unnamed Variable for use in read and write operations, you must:

1. Set the -ADDRTYPE attribute to UNCONSTRAINED.
2. Set an appropriate Allen-Bradley memory location address in the -ADDRESS attribute. The syntax is as follows:

   ```
   $[<ModuleNumber]<FileType><FileNumber>:<ElementNumber>
   ```

   Refer to the appropriate Allen-Bradley documentation for details.

Example 4–3 shows how to create a Device Services variable that can be used for write and read operations to the memory location $N7:0 of the device identified by the VMD_INTERCHANGE VMD.

Example 4–3: Creating an Unnamed_Variable for Read and Write Operations

```
BSTR> CREATE UNNAMED_VARIABLE VMD_INTERCHANGE.VARSOL_001\
      -DATATYPE INTEGER_32\
      -SEND NEVER  -ACCESS WRITE
      -ADDRTYPE UNCONSTRAINED -ADDRESS "$N7:0"
```

Creating Unnamed_Variables for Receiving Unsolicited Messages

To define an Unnamed Variable for use in receiving an unsolicited message from the device, you must:

1. Set the -ADDRTYPE attribute to NUMERIC.
2. Set the appropriate data item identifier in the -ADDRESS attribute. Remember that this works if the specified data item has been correctly configured on the Allen-Bradley device itself. Allowed values range is from 0 to 77777 octal.

   Refer to the appropriate Allen-Bradley documentation for details.

Example 4–4 shows how to create an Unnamed_Varibale for receiving an unsolicited message generated by the device data item 200. The chosen configuration directs BASEstar Open to update the value of the Device Service variable each time an unsolicited write command is received. Because the -SEND attribute has been set to ALWAYS, BASEstar Open also updates asynchronously the value of any linked Device_Data_Points.

Example 4–4: Creating an Unnamed_Variable for Receiving Unsolicited Messages

```
BSTR> CREATE UNNAMED_VARIABLE VMD_INTERCHANGE.VARUNS_001\
       -DATATYPE INTEGER_32\
       -SEND ALWAYS -ACCESS READ
       -ADDRTYPE NUMERIC -ADDRESS "200"
```

Specifying the Maximum Number of Variables

At initialization time, the DAS for Allen-Bradley INTERCHANGE must know the maximum number of variables that will be used for communication with the device.

The default value is 500. However, you can change this value by defining the DASDTL_DEF_TABLE_SIZE environment in the Parameter File of the Device Services server Program.

Creating Data Services Objects

BASEstar Open users exchange data with devices by means of Device_Data_Points without taking into consideration communication and device characteristics.

Creating Device_Data_Points for Read and Write Operations

A BASEstar Open user uses a Device_Data_Point associated with a VMD to write data to (and read data from) a memory location of an Allen-Bradley device.

Table 4–4 lists the attributes you must set to define a Device_Data_Point that will be used for read and write operations.

Table 4–4: Device_Data_Point Attributes for Read and Write Operations

Attribute Name	Description and Values
-VMD_NAME *vmd_name*	Must be set with the name of a VMD, that is, a VMD that models an Allen-Bradley device.
-VAR_NAME *var_name*	Must be set with the name of a Device Services Unnamed_Variable that is associated with the *vmd_name* VMD and is configured for read and write operations.
-VAR_CLASS *var_class*	Must be set to UNNAMED.
-DATATYPE *datatype_name*	Must be set with the name of the same datatype associated with the *var_name* variable specified by the -VAR_NAME attribute.
-DEVICE_ACCESS *device_access*	You are suggested to set it to RDWR. In this way, users can use the Device_Data_Point for both read and write operations.
-UPDATE_POLICY *update_policy*	Only significant for read operations. You can set any value provided it is consistent with the overall configuration.
-ACCESS_POLICY *access_policy*	Only significant for read operations. You can set any value provided it is consistent with the overall configuration.

Using the Device_Data_Point shown in Example 4–5, a BASEstar Open user can perform read and write operations on the device.

You can change the values of the -UPDATE_POLICY and -ACCESS_POLICY attributes to meet your specific configuration requirements.

Example 4–5: Creating a Device_Data_Point for Read and Write Operations

```
CREATE DEVICE_DATA_POINT DP_SOLICITED\
       -VMD_NAME VMD_INTERCHANGE\
       -VAR_NAME VAR_SOL001 -VAR_CLASS UNNAMED\
       -DATATYPE INTEGER_32 -DEVICE_ACCESS RDWR\
       -UPDATE_POLICY SOLICITED -ACCESS_POLICY ALWAYS
```

Creating Device_Data_Points for Receiving Unsolicited Messages

Table 4–5 lists the attributes you must set to define Device_Data_Points to be used for receiving unsolicited messages.

Table 4–5: Device_Data_Point Attributes for Unsolicited Messages

Attribute Name	Description and Values
-VMD_NAME *vmd_name*	Must be set with the name of an appropriate remote VMD.
-VAR_NAME *var_name*	Must be set with the name of an Unnamed_Variable that is associated with the *vmd_name* VMD and is configured for unsolicited operations.
-VAR_CLASS *var_class*	Must be set to UNNAMED.
-DATATYPE *datatype_name*	Must be set with the name of the same datatype associated with the *var_name* variable specified by the -VAR_NAME attribute.
-DEVICE_ACCESS *device_access*	Must be set to READ to indicate that the linked Device Services variable produces the value of the Device_Data_Point.
-UPDATE_POLICY update_policy	Must be set to UNSOLICITED to indicate that Data Services updates in asynchronous mode the value of the Device_Data_Point according to what was specified in the -SEND attribute of the linked Device Services variable.

5
A Complete Example

This chapter describes through an example how you can configure BASEstar Open to:

- Perform read and write operations
- Receive unsolicited messages from the remote device.

Example Overview

The example contained in this chapter shows you a possible configuration that can be used to perform read and write operations, and to receive unsolicited messages from an Allen-Bradley PLC processor directly connected to the Ethernet network.

BASEstar Open being highly configurable, this example shows only one of many possible modes for implementing these operations.

Read and write operations

It is assumed that read and write operations are executed in synchronous mode. According to configuration, BASEstar Open physically accesses the memory of the addressed device each time a user issues a PUT_VALUE or a GET_VALUE command. The command only terminates and returns control to the user when the operation has been completed.

Reception of unsolicited messages

In the configuration example, an event is generated each time BASEstar Open receives an unsolicited write command from the device referred to by the configured data item. This allows the involved BASEstar Open users to be notified in an immediate and asynchronous mode of any change occurring on the device. For this purpose also a Trigger and the appropriate Event Services objects are defined (these would be useless in different configurations).

Before Running the Example

It is assumed that the Allen-Bradley device has been configured, is functioning, is accessible and can understand incoming requests. Before you can use this example, you must:

- Replace *My_TCPnode* with the TCP/IP name of your the BASEstar Open Node.
- Replace *My_Model* with the model identifier of your Allen-Bradley PLC processor.
- Replace *My_ODS_Interchange* with the name of the ODS entry that complies with the communication characteristics of your Allen-Bradley processor.

CLI Source

Example 5-1 shows the CLI commands used to create BASEstar Open objects and to activate the required servers.

Example 5-1: Creating BASEstar Open Objects

```
! ----------------------------------------------------------------
! The example is set up to work with the VOLATILE database,
! but you can easily adapt it to work with the PERMANENT database.
! ----------------------------------------------------------------
SET SCOPE VOLATILE
! ----------------------------------------------------------------
! Create the Common Services objects: Nodes, Actors, Domains,
| Protocol_Profiles and VMDs
! ----------------------------------------------------------------
CREATE NODE NODE1 -PHYSICAL_NAME "My_TCPnode"

CREATE DOMAIN /DOM_DATA

CREATE ACTOR /ACT_SERVERS

CREATE PROTOCOL_PROFILE PP_INTERCHANGE -APPLPROFID 81\
       -MAXPDUSIZE 2048 -LOG

CREATE VMD VMD_INTERCHANGE\
       -MODEL "My_Model" -VENDOR "Allen-Bradley"\
       -DVM_ACCESS_POINTS ODS_INTERCHANGE:PP_INTERCHANGE)\
       -DESCRIPTION "My_Description"\
       -LOG
! ----------------------------------------------------------------
! Create the objects needed to define Data Services and
! Device Services servers.
! ----------------------------------------------------------------
CREATE PROGRAM /ACT_SERVERS/PRG_DATA -PROGRAM_KIND DATADEV

CREATE ACTIVITY /ACT_SERVERS/ACY_DATA\
       -PROGRAM /ACT_SERVERS/PRG_DATA -NODES (NODE1)\
       -DOMAINS (/DOM_DATA)\
       -VMDS (/VMD_INTERCHANGE)\
       -STARTUP_TIMEOUT 120 -RECOVERY_POLICY NECESSARY
! ----------------------------------------------------------------
! Note that an Event Services server Activity is used in this
! example, but using Events and Trigger is only a matter of
! configuration choice.
! ----------------------------------------------------------------
CREATE PROGRAM /ACT_SERVERS/PRG_EVENT -PROGRAM_KIND EVENT

CREATE ACTIVITY /ACT_SERVERS/ACY_EVENT\
       -PROGRAM /ACT_SERVERS/PRG_EVENT -NODES (NODE1)\
       -DOMAINS (/DOM_DATA)\
       -STARTUP_TIMEOUT 120 -RECOVERY_POLICY NECESSARY


! ----------------------------------------------------------------
! Execute the ACTOR to activate required servers and set the context.
! ----------------------------------------------------------------
```

```
EXECUTE ACTOR /ACT_SERVERS

SET CONTEXT /DOM_DATA

! ------------------------------------------------------------------

! Create the Device Services variables.

!The following variable provides for solicited access to
!the $N7:0 device memory location. To access another memory location
!change the value of the -ADDRESS attribute.

! ------------------------------------------------------------------

CREATE UNNAMED_VAR VMD_INTERCHANGE.VAR_SOLICITED \
       -DATATYPE INTEGER_32\
       -SEND NEVER -ACCESSMODE WRITE
       -ADDRTYPE UNCONSTRAINED -ADDRESS "$N7:0"

! ------------------------------------------------------------------

! The following variable allows reception of an unsolicited message.
! To receive values associated with other data items change the
! value of the -ADDRESS attribute.

! ------------------------------------------------------------------

CREATE UNNAMED_VAR VMD_INTERCHANGE.VAR_UNSOLICITED\
       -DATATYPE INTEGER_32 -ACCESSMODE RED -SEND ALWAYS
       -ADDRTYPE NUMERIC -ADDRESS "200"

! ------------------------------------------------------------------

! Create the Data Services objects.

! ------------------------------------------------------------------

CREATE DEVICE_DATA_POINT DP_SOLICITED\
       -VMD_NAME VMD_INTERCHANGE -VAR_CLASS UNNAMED\
       -VAR_NAME VAR_SOLICITED\
       -DATATYPE INTEGER_32
       -DEVICE_ACCESS RDWR\
       -UPDATE_POLICY SOLICITED -ACCESS_POLICY ALWAYS

CREATE DEVICE_DATA_POINT DP_UNSOLICITED\
       -VMD_NAME VMD_INTERCHANGE -VAR_NAME VAR_UNSOLICITED\
       -VAR_CLASS UNNAMED\
       -DATATYPE INTEGER_32\
       -DEVICE_ACCESS READ\
       -UPDATE_POLICY UNSOLICITED

! ------------------------------------------------------------------

! Create the Trigger that allows a user to be notified by the
! EV_INTERCHANGE event of any change in the value of the
! DP_UNSOLICITED Device_Data_Point.

! ------------------------------------------------------------------

CREATE EVENT EV_INTERCHANGE

CREATE ENBOX EN_INTERCHANGE\
       -MAX_NOTIFICATIONS 5

ADD SUBSCRIPTION EVENT EV_INTERCHANGE EN_INTERCHANGE

CREATE TRIGGER TR_INTERCHANGE\
       -DATA_POINT DP_UNSOLICITED
        -EVENT EV_INTERCHANGE
```

Example 5-2 shows a possible sequence that allows a user to read and write the configured memory location. On successful completion of the following command sequence, you have first read the current value of the $N7:0 address directly from the memory of the device, then

you have written a new value, and finally you have checked the written value by issuing another read command.

Example 5-2: Reading and Writing the Memory of a Device

```
BSTR> GET VALUE DEVICE_DATA_POINT DP_SOLICITED
12
BSTR>
BSTR> PUT VALUE DEVICE_DATA_POINT DP_SOLICITED
(Integer 32) : 345
BSTR>
BSTR> GET VALUE DEVICE_DATA_POINT DP_SOLICITED
345
```

To get the value carried by a received unsolicited write command generated by the data item 200, a BASEstar Open user can issue the sequence of commands shown in Example 5-3 (remember that RECEIVE NOTIFICATION is a suspensive command which only returns control to the user when at least an Event has been generated for the specified Enbox).

Example 5-3: Getting the Value Carried by an Unsolicited Write Command

```
BSTR>
BSTR> CONNECT ENBOX EN_INTERCHANGE
BSTR> RECEIVE NOTIFICATION ENBOX EN_INTERCHANGE
BSTR> ...
BSTR> GET VALUE DEVICE_DATA_POINT DP_UNSOLICITED
12
```

PART II

Using the DEComni API to Communicate With Allen-Bradley INTERCHANGE Processors

This part explains how to prepare DEComni API applications to communicate with Allen-Bradley devices using the features of the DAS for Allen-Bradley INTERCHANGE.

6
Overview

This chapter describes the DEComni API environment components involved when you access Allen-Bradley PLC processors via the DEComni API.

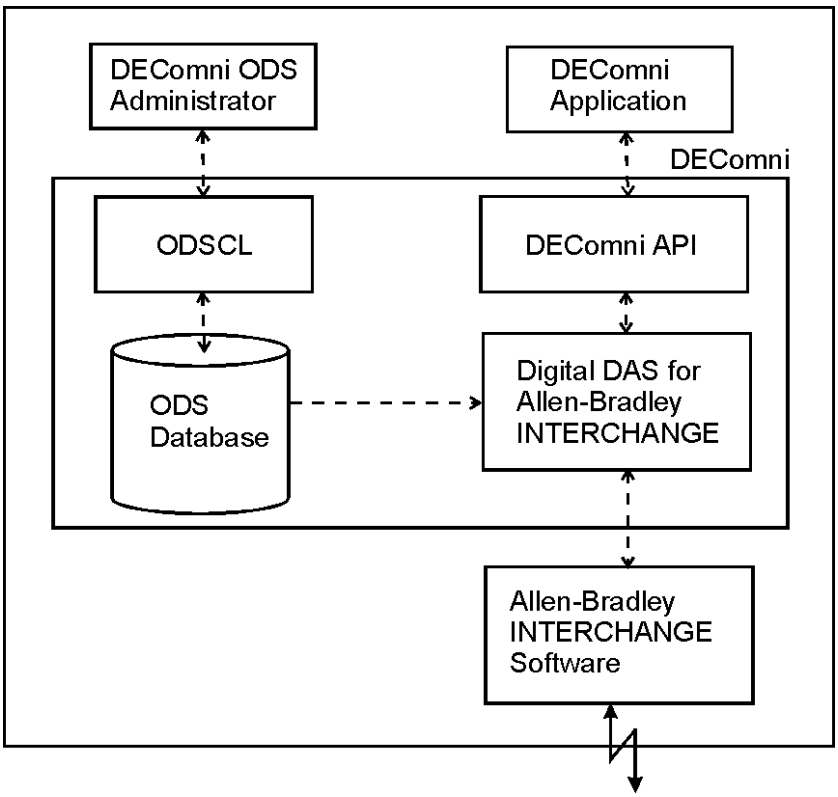
It also provides an overview of the operations a DEComni API application can perform using the features provided by the DAS for Allen-Bradley INTERCHANGE. Calling the appropriate procedures, a DEComni API application can:

- Perform read and write operations
- Receive unsolicited messages from the remote device.

DEComni Environment Components

Figure 6–1 shows the components involved when DEComni API applications exchange data with Allen-Bradley PLC processors.

Figure 6–1: DEComni API Environment Components



DEComni API

The DEComni API provides applications with a unique set of operations for device-independent access to any type of device or network. This device-independent approach to device connectivity is possible because each device is modeled as an MMS Virtual Manufacturing Device (VMD).

A VMD is associated with an Allen-Bradley INTERCHANGE processor. A variable object corresponds to data stored in the device memory. The DEComni API also provides a standardized means of adding new DAS modules.

The Omni Directory Services (ODS) database is the DEComni API component that allows you to define the information required for addressing an Allen-Bradley PLC processor connected to the Ethernet network.

DIGITAL DAS for Allen-Bradley INTERCHANGE

The DIGITAL DAS for Allen-Bradley INTERCHANGE is a BASEstar Open device connectivity module that adapts the application programming interface provided by the Allen-Bradley INTERCHANGE software to BASEstar Open, thus allowing BASEstar Open users to exchange data with Allen-Bradley PLC processors.

Allen-Bradley INTERCHANGE Software

This is an Allen-Bradley library of functions and executable commands for accessing data-table memory or shared-data memory on Allen-Bradley PLC processors.

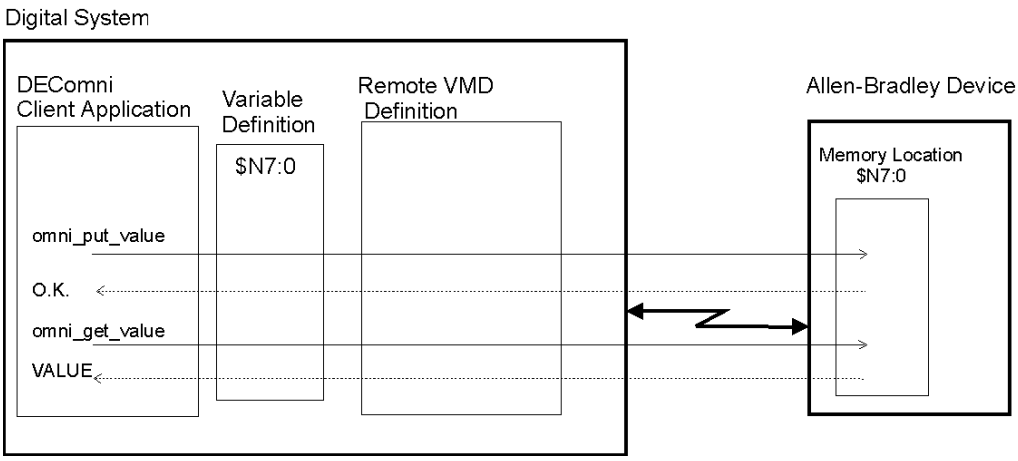
Writing and Reading Device Memory Locations

The DAS for Allen-Bradley INTERCHANGE allows a DEComni API application to read and write the value of memory locations of a remote Allen-Bradley device.

To perform these operations, a DEComni API application uses VMD and Unnamed Variables definitions. DEComni API variables can map any type of addressable memory locations on the Allen-Bradley device. Each Unnamed Variable that is used in a read or write operation must specify the address of a valid memory location on the addressed Allen-Bradley device.

As Figure 6–2 illustrates, a DEComni API application calls the *omni_put_value* (or the *omni_get_value*) procedure, and specifies the handle of an Unnamed_Variable associated with a remote VMD, to write a value to (or read value from) a memory location of the device.

Figure 6–2: Writing and Reading Device Memory Locations



An *omni_get_value* procedure reads data from the device memory, and returns in the user buffer a number of characters that is equal to the size of the MMS Type associated with the variable used for the read operation. A similar consideration applies to an *omni_put_value* procedure.

See Appendix C for information on how to choose the appropriate DEComni API data type that maps the appropriate Allen-Bradley memory location type.

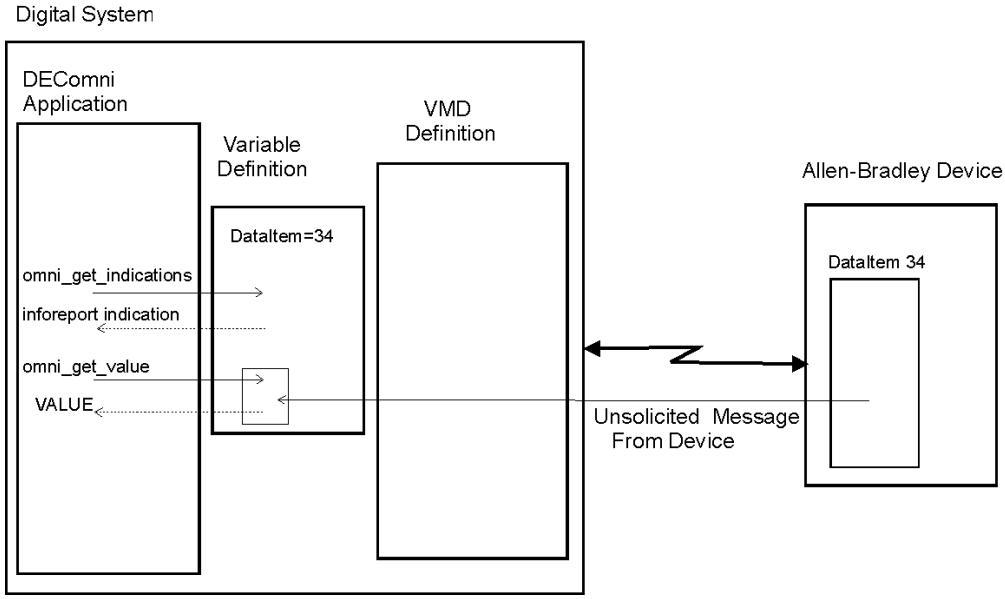
Receiving Unsolicited Messages

Figure 6–3 shows the DEComni API definitions and procedures an application must use to receive an unsolicited message from an Allen-Bradley device.

To receive an unsolicited message, a DEComni API application must call the *omni_get_indications* procedure. When receiving an unsolicited write command from the remote device, the DAS for Allen-Bradley INTERCHANGE generates an Information report indication on the remote VMD definition. This causes the completion of the *omni_get_indications* procedure.

The returned context refers to the variable in which the DAS for Allen-Bradley INTERCHANGE returns the value associated with the unsolicited write command.

Figure 6–3: Receiving Unsolicited Messages



An Unnamed_Variable definition that is used for receiving unsolicited messages must specify the identifier of an unsolicited data item configured at the addressed Allen-Bradley device.

The DAS for Allen-Bradley INTERCHANGE interprets the data received from the device in an unsolicited write command according to the MMS data type associated with the Unnamed_Variable in use. If an insufficient number of bytes is received, the value returned in the user buffer is unpredictable; if, instead, the number of the received bytes exceeds that required to meet the data type, the exceeding bytes are lost.

7
Configuring DEComni API

You must configure DEComni API in order to allow DEComni API applications to exchange data with an Allen-Bradley device. Configuring DEComni API means creating the required definitions using the following API procedures:

- *omni_define*
- *omni_get_definition*
- *omni_modify_definition*

Note

This chapter describes which values you must specify for DEComni API definition attributes. For an explanation of DEComni API attributes whose values are not specified here, refer to the *DEComni API and DEComni MMS User Guide* for your platform.

To configure DEComni API, you must:

- Register the appropriate entries in the ODS database
- Set the required application profile
- Create the required DEComni VMD and Variable definitions.

Registering ODS Entries

For each Allen-Bradley INTERCHANGE processor you must create an ODS entry which specifies INTERCHANGE protocol-specific information. See Appendix A for details.

Setting the Application Profile

An application must invoke the *omni_set_application_profile* procedure to specify the appropriate application profile. This ensures that the definitions created after this procedure is called are appropriate for use with the DAS for Allen-Bradley INTERCHANGE.

The following example shows how an application can invoke the *omni_set_application_profile* procedure to set the required application profile.

```
status=omni_set_application_profile (omni_c_app_profile_dtl)
```

Creating VMD Definitions

A DEComni API application must create a VMD definition for each Allen-Bradley device with which the application wishes to communicate. The same VMD definition can be used for read and write operations, and for receiving unsolicited messages from the device.

Table 7–1 lists the mandatory attributes that you must specify when defining a VMD.

Table 7–1: DEComni VMD Attributes

Attribute Name	Description and Values
omni_c_attr_name	The name of the VMD definition.
omni_c_attr_appl_simple_name	The name of an ODS entry valid to address an Allen-Bradley device. See Appendix A for a description of ODS entries attributes.
omni_c_vmd_max_segment	The maximum length in bytes of the data exchanged with the Allen-Bradley device, as follows: 2048 for PLCs that are directly connected to the Ethernet network. 242 for PLCs that are connected by a DH or DH+ link to the Ethernet network.
omni_c_attr_model	A string that specifies the type of the PLC processor at the remote station. Valid values for this attribute are: PLC_5_250 PLC_5_5 PLC_3 PLC_2 SLC_500

Creating Variable Definitions

For each device memory location it wants to access, a DEComni API application must create an Unnamed_Variable definition. Creation of Unnamed_Variables change depending on whether you are using them for read and write operations, or for receiving unsolicited messages from the device.

Table 7–2 lists all the mandatory attributes that you must specify for a correct definition of a variable.

Table 7–2: DEComni Variable Attributes

Attribute Name	Description and Values
omni_c_attr_app_type_descr	Name of the application type. Appendix C specifies the size of the basic DEComni MMS types and the mapping between DEComni MMS types and Allen-Bradley memory location types. For variables used for receiving unsolicited messages, the used DEComni MMS type must be appropriate and must have a size equal to the size of the device memory location associated with the unsolicited data item. For variables used in read and write operations, the used DEComni MMS type must be appropriate and its size must correspond to the actual length of the data to be read or written.
omni_c_attr_address_type	For variables used in read and write operations, it must be set to omni_c_address_unconstrained. For variables used for receiving unsolicited messages, it must be set to omni_c_address_numeric.
omni_c_attr_address_string	The value that you must provide depends on the type of operation. See the following two sections for details.

Creating Unnamed_Variables for Read and Write Operations

For a correct definition of an Unnamed Variable to be used for read and write operations, a DEComni API application must:

1. Set the omni_c_attr_address_type attribute to omni_c_address_unconstrained.
2. Set an appropriate Allen-Bradley memory location address in the omni_c_attr_address_string attribute. The syntax is as follows:

   ```
   $[<ModuleNumber]<FileType><FileNumber>:<ElementNumber>
   ```

   Refer to the appropriate Allen-Bradley documentation for details.

Creating Unnamed_Variables for Receiving Unsolicited Messages

To define an Unnamed Variable for receiving an unsolicited message from the device, a DEComni API application must:

1. Set the omni_c_attr_address_type attribute to omni_c_address_numeric.
2. Set the appropriate data item identifier in the omni_c_attr_address_string attribute. Remember that this works if the specified data item has been correctly configured on the Allen-Bradley device itself. Allowed values range is from 0 to 77777 octal.

   Refer to the appropriate Allen-Bradley documentation for details.

Specifying the Maximum Number of Variables

At initialization time, the DAS for Allen-Bradley INTERCHANGE must know the maximum number of variables that will be used for communication with the device.

The default value is 500. However, you can change this value by defining the DASDTL_DEF_TABLE_SIZE environment variable before activating DEComni API, as shown in Example 7-1 and Example 7-2.

Example 7-1: Setting the Maximum Number of Variables (DIGITAL UNIX)

```
# setenv DASDTL_DEF_TABLE_SIZE 1000
```

Example 7-2: Setting the Maximum Number of Variables (OpenVMS)

```
$ define DASDTL_DEF_TABLE_SIZE 1000
```

8
Programming

This chapter describes how you can use the DEComni API procedures to write a DEComni API application that is capable of exchanging data with an Allen-Bradley device. Bear in mind the following considerations when using the API procedures to write your application:

- The general syntax and programming rules, described in the *DEComni API and DEComni MMS User Guide* for your platform, are also valid ifthe API procedures are used in the DAS for Allen-Bradley INTERCHANGE context.
- When using the *DEComni API and DEComni MMS User Guide* for your platform, ignore any descriptions that refer to services and functions not supported by the DAS for Allen-Bradley INTERCHANGE.

DEComni API Procedures for Data Exchange

Table 8-1 lists the DEComni API procedures that a DEComni API application can invoke to manage the communication with Allen-Bradley devices.

Table 8-1: DEComni API Procedures for Data Exchange

Procedure Name	Description
omni_abort	Causes the forced closure of a connection with an Allen-Bradley device (local operation only).
omni_conclude	Causes the forced closure of a connection with an Allen-Bradley device. It behaves in the same way as the *omni_abort* procedure (local operation only).
omni_connect	Opens a connection with an Allen-Bradley device (local operation only).
omni_get_indications	Returns connection-related and Inforeport indications (for example, the Inforeport indication notifies the reception of the value carried by an unsolicited write command).
omni_get_value	When specifying a variable defined for read and write operations, it allows a DEComni API application to read the value of the specified memory address directly from the Allen-Bradley device memory. When specifying a context returned by an Inforeport indication, it allows a DEComni API application to get the value of an unsolicited message.
omni_put_value	When specifying a variable defined for read and write operations, it allows a DEComni API application to write the value into a memory location of an Allen-Bradley device.

Opening and Closing a Connection

To open a connection with a remote Allen-Bradley device, a DEComni API application must call the *omni_connect* procedure, and specify the VMD definition associated with the device as remote VMD.

A DEComni API application must call the *omni_get_indications* procedure to monitor an incoming omni_c_ind_abort indication which notifies that the connection has been terminated for any reason.

To close a connection, an application must call the *omni_conclude* or the *omni_abort* procedure. These procedures perform the same operations, have only local effect, and cause a forced closure of the connection.

Writing and Reading Device Memory Locations

To write (or read) the value of a memory location of a remote Allen-Bradley device, a DEComni API application must call the *omni_put_value* (or *omni_get_value*) procedure. The Unnamed_Variable referred to by the specified handle must be associated with the VMD that models the device, and must have been configured for read and write operations on the device memory location.

Receiving Unsolicited Messages

A DEComni API application awaiting unsolicited messages from the device must call the *omni_get_indications* procedure, and specify the handle of the remote VMD associated with that device.

Each time the DAS for Allen-Bradley INTERCHANGE receives data from the remote device, it generates an Inforeport indication that causes the completion of the *omni_get_indications* procedure.

On completion, the *omni_get_indications* procedure returns to the application a context which refers to the variable definition that was configured for the specified data item. To obtain the value carried by the unsolicited write command, the application must issue an *omni_get_value* procedure call, and specify this context.

PART III

Appendices

A

ODS Entry Format for the Allen-Bradley INTERCHANGE Protocol

This appendix describes the ODS attributes (and their associated values) that allow you to create the ODS entries required to access Allen-Bradley devices. These ODS entries must be specified in any VMD definitions used to access Allen-Bradley devices.

For details of how to use the ODS CLI, refer to the DEComni documentation included in your documentation kit.

Table A–1 provides the name and an explanation for each ODS attribute.

Table A–1: Values of ODS Attributes for Allen-Bradley Devices

Attribute Name	Description and Values
\oc	Identifier of the ODS object class. Must be set to DTL.
netaddress	IP Address of the EI station.
portid	Name of the DH/DH+ port through which the server communicates across the DH/DH+ link to the remote station. See the appropriate Allen-Bradley INTERCHANGE documentation.
stationid	DH/DH+ station number of the remote station. See the appropriate Allen-Bradley INTERCHANGE documentation.
niid	Network Interface Identifier number corresponding to the INTERCHANGE server. See the appropriate Allen-Bradley INTERCHANGE documentation.
lev7conn	Specifies if the connection is handled with messages at application level. Must be set to 0 because the Allen-Bradley INTERCHANGE protocol does not exchange messages for connection at application level.
rsptimeout	Timeout (in seconds) for the reception of the response to a solicited function request, after which the service is terminated with failure.

Example A-1 shows a configuration file command that can be used to create an ODS registration valid to address an Allen-Bradley device (station) local to its EI network interface module.

Example A-1: An ODS Entry for a Device Connected Directly to Ethernet

```
register directory name "/cn=DTL_EI1"
       attributes "/oc=DTL
       /netaddress=16.192.80.139
       /niid=1
       /rsptimeout=5
```

Example A-2 shows a configuration file command that can be used to create an ODS registration valid to address an Allen-Bradley device (station) remote in respect of the specified EI network interface module.

Example A-2: An ODS Entry for a Device Connected by a DH or DH+ Link

```
register directory name "/cn=DTL_RM1"
       attributes "/oc=DTL
       /netaddress=16.192.80.139
       /niid=1
       /rsptimeout=5
       /portid=ORM:2
       /stationid=001
```

B
Mapping BASEstar Open Datatypes

The number of bytes involved in a write or read operation depends on the datatype associated with the Device Services variable in use. This appendix provides you with:

- The size of the basic BASEstar Open datatypes
- The suggested mappings of Allen-Bradley types with BASEstar Open datatypes.

Table B–1 lists the size (in bytes) of the BASEstar Open basic datatypes: this allows you to define a datatype that has a size appropriate for the data to be exchanged with an Allen-Bradley device.

Table B–1: BASEstar Open Simple Datatype Size

Data Type	Exchanged as
BOOLEAN	1 byte
TIME	Not supported
VISIBLE_CHAR	1 byte
INTEGER_8	1 byte
INTEGER_16	2 bytes
INTEGER_32	4 bytes
UNSIGNED_8	1 byte
UNSIGNED_16	2 bytes
UNSIGNED_32	4 bytes
F_FLOAT	4 bytes (IEEE float format)
D_FLOAT	Not supported
BCD1, BCD2, BCD3, BCD4, BCD5, BCD6, BCD7, BCD8	Not supported
OCTET	1 byte
BIT	1 byte
OBJECT_STATUS	Not supported
OBJECT_NAME	Not supported
ARRAY_DATATYPE	As many times as the number of elements it contains. A variable length array is treated as an array having a fixed length; the maximum length is assumed
STRUCTURE_DATATYPE	Packing of the Simple_Datatypes without padding

Table B–2 lists the suggested mapping between Allen-Bradley types and DEComni API types (where not otherwise specified, "integer" means INTEGER_8, INTEGER_16 or INTEGER_32 depending on specific needs).

Table B–2: Mapping Allen-Bradley Types to BASEstar Open Datatypes

Allen-Bradley File Type	File Type Description	BASEstar Open Datatype
O	Output Image	INTEGER, an Array_Datatype of INTEGERs
I	Input Image	INTEGER, an Array_Datatype of INTEGERs
S	Status	INTEGER, an Array_Datatype of INTEGERs
B	Bit (binary)	BOOLEAN, INTEGER, an Array_Datatype of INTEGERs or BOOLEANs
T	Timer	An Array_Datatype of 6 OCTETs, an Array_Datatype of 3 INTEGER_16s a Structure_Datatype
C	Counter	An Array_Datatype of 6 OCTETs, an Array_Datatype of 3 INTEGER_16s a Structure_Datatype
R	Control	An Array_Datatype of 6 OCTETs, an Array_Datatype of 3 INTEGER_16s a Structure_Datatype
N	Integer	INTEGER, an Array_Datatype of INTEGERs
F	Floating-Point	F_FLOAT, an Array_Datatype of F_FLOATs
A	ASCII	An Array_Datatype of VISIBLE_CHARs or OCTETs
D	BCD	Not supported
BT	Block Transfer	An Array_Datatype of 12c OCTETs, a Structure_Datatype
MG,MSG	Message	An Array_Datatype of 112c OCTETs, a Structure_Datatype
PD	PID	An Array_Datatype of 164c OCTETs, a Structure_Datatype
SC	SFC Status	An Array_Datatype of 6c OCTETs, a Structure_Datatype
ST	ASCII String	An Array_Datatype of VISIBLE_CHARs

C
Mapping DEComni API Types

The number of bytes involved in a write or read operation depends on the MMS type associated with the DEComni variable in use. This appendix contains:

- The size of the basic DEComni MMS types
- The suggested mappings of Allen-Bradley types to DEComni types.

Table C–1 lists the size (in bytes) of the basic DEComni MMS types. This allows you to define a DEComni API data type that has a size appropriate for the data to be exchanged with an Allen-Bradley device.

Table C–1: DEComni Basic MMS Type Sizes

Data Type	Exchanged as
omni_c_mmstype_boolean	1 byte
omni_c_mmstype_bit_str	(*n* + 7)/8) bytes (*n* is the number of bits in the string)
omni_c_mmstype_integer	1, 2 or 4 bytes, depending on omni_c_symbol_size, that specify the size of the variable (8, 16, or 32 bits)
omni_c_mmstype_unsigned	1, 2 or 4 bytes, depending on omni_c_symbol_size, that specify the size of the variable (8, 16, or 32 bits)
omni_c_mmstype_floating_point	4 bytes (IEEE float format)
omni_c_mmstype_real	Not supported
omni_c_mmstype_octet_str	*n* bytes (as returned by sizeof())
omni_c_mmstype_visible_str	*n* bytes (as returned by strlen() if varying, or as returned by sizeof() otherwise)
omni_c_mmstype_generalized_time	Not supported
omni_c_mmstype_binary_time	Not supported

Table C–2 lists the suggested mappings between Allen-Bradley types and DEComni types (where not otherwise specified, "integer" means integer 8, integer 16 or integer 32, depending the specific needs).

Table C–2: Mapping Allen-Bradley Types to DEComni Types

Allen-Bradley File Type	File Type Description	DEComni MMS Type	DEComni Application Type
O	Output Image	Integer, Bit String, Array[n] of Int/BS	Integer, Bit String, Array[n] of Int/BS
I	Input Image	Integer, Bit String, Array[n] of Int/BS	Integer, Bit String, Array[n] of Int/BS
S	Status	Integer, Bit String, Array[n] of Int/BS	Integer, Bit String, Array[n] of Int/BS
B	Bit (binary)	Boolean, Integer, Bit String, Array[n] of Int/BS/Boolean	Boolean, Integer, Bit String, Array[n] of Int/BS/Boolean
T	Timer	6 bytes long Octet String, Array[3] of Integer 16, Structure	Word Counted String, Array[3] of Integer 16, Structure
C	Counter	6 bytes long Octet String, Array[3] of Integer 16, Structure	Word Counted String, Array[3] of Integer 16, Structure
R	Control	6 bytes long Octet String, Array[3] of Integer 16, Structure	Word Counted String, Array[3] of Integer 16, Structure
N	Integer	Integer, Array [n] of Integer	Integer, Array [n] of Integer
F	Floating-Point	Floating Point, Array [n] of Float	Floating Point, Array [n] of Float
A	ASCII	Visible String, Octet String	Word counted String
D	BCD	Not supported	Not supported
BT	Block Transfer	12 bytes long Octet String, Structure	Word-counted String, Structure
BT	Block Transfer	12 bytes long Octet String, Structure	Word-counted String, Structure
MG,MSG	Message	112 bytes long Octet String, Structure	Word-counted String, Structure
PD	PID	164 bytes long Octet String, Structure	Word-counted String, Structure
SC	SFC Status	6 bytes long Octet String, Structure	Word-counted String, Structure
ST	ASCII String	Visible String	Null-terminated string

D

DEComni API Network IOSB Messages

This appendix lists the DEComni API messages returned by the DAS for Allen-Bradley INTERCHANGE in the Network IOSB.

dastk_s_success 80125953

Explanation: das toolkit service completed successfully.

User action: none

dastk_s_asoabort 80125962

Explanation: the association has been aborted by the peer.

User action: none.

dastk_s_asoalrdyexist 80125970

Explanation: an association with the same parameters already exists.

User action: change vmd addressing paramters.

dastk_s_nolicense 80125978

Explanation: no valid license have been loaded.

User action: install and load a valid pak for this product.

dastk_s_nosuchasn 80125986

Explanation: the specified application simple name has not been found in ODS.

User action: check ods configuration.

dastk_s_nosuchaso 80125994

Explanation: the specified association does not exist.

User action: check if you refer to the right association.

dastk_s_reqabort 80126002

Explanation: the specified request has been aborted.

User action: none.

dastk_s_reqtimeout 80126010

Explanation: the timer for the specified request expired.

User action: none.

dastk_s_connerr 80126018

Explanation: a generic error occured on the communication layer.

User action: none.

dastk_s_insfmem 80126026

Explanation: insufficient virtual memory.

User action: none.

dastk_s_lentoolrg 80126034

Explanation: pdu length exceeds the maximum allowed value.

User action: redesign user application to use smaller object values.

dastk_s_generr 80126042

Explanation: generic error.

User action: none.