Customizing EPD®.Connect

Optegra® Release 6

DOC40191-008

Parametric Technology Corporation

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494-2714
8 January 2001

Table of Contents

Customizing the Interface

Customizing Reports

Sending Commands to Agents Using Opt.pm

Customizing the Client

Destinations, Format, and Commands

Preface

Customizing EPD.Connect describes how to customize an EPD.Connect CMOM client using the supplied Perl modules and extensions.

Related Documents

The following documents may be helpful as you use *Customizing EPD.Connect*:

- *Installing Optegra Applications*
- *Installing EPD.Connect, EPD Roles, and EPD.Visualizer*
- *EPD.Connect User Guide*
- *AccesssWare Function Reference*
- *AccessWare User Guide*

Book Conventions

The following table illustrates and explains conventions used in writing about Optegra applications.

Convention	Example	Explanation
EPD_HOME	`cd` $EPD_HOME`/install` (UNIX) `cd` %EPD_HOME%`\install` (Windows)	Represents the default path where the current version of the product is installed.
Menu selections	Vault > Check Out > Lock	Indicates a command that you can choose from a menu.
Command buttons and options	Mandatory check box, Add button, Description text box	Names selectable items from dialog boxes: options, buttons, toggles, text boxes, and switches.

User input and code	`Wheel_Assy_details`
	
	`-xvf /dev/rst0`
	
	`Enter command>` **`plot_config`**
		Enter the text in a text box or on a command line.
		
		Where system output and user input are mixed, user input is in bold.

System output	`CT_struct.aename`	Indicates system responses.
Parameter and variable names	`tar -cvf /dev/rst0` filename	Supply an appropriate substitute for each parameter or variable; for example, replace filename with an actual file name.
Commands and keywords	The `ciaddobj` command creates an instance of a binder.	Shows command syntax.
Text string	`"SRFGROUPA"` or `'SRFGROUPA'`	Shows text strings. Enclose text strings with single or double quotation marks.
Integer	n	Supply an integer for *n*.
Real number	x	Supply a real number for *x*.
#	`# mkdir /cdrom`	Indicates the root (superuser) prompt on command lines.
%	`%` `rlogin` remote_system_name `-l root`	Indicates the C shell prompt on command lines.
$	`$` `rlogin` remote_system_name `-l root`	Indicates the Bourne shell prompt on command lines.
>	`> copy` filename	Indicates the MS-DOS prompt on command lines.
Keystrokes	`Return` or `Control-g`	Indicates the keys to press on a keyboard.

Online User Documentation

Online documentation for each Optegra book is provided in HTML if the documentation CD-ROM is installed. You can view the online documentation from an HTML browser or from the HELP command.

You can also view the online documentation directly from the CD-ROM without installing it.

From an HTML Browser:

1. Navigate to the directory where the documents are installed. For example,

   $EPD_HOME`/data/html/htmldoc/` (UNIX)

   %EPD_HOME%`\data\html\htmldoc\` (Windows NT)
2. Click `mainmenu.html`. A list of available Optegra documentation appears.
3. Click the book title you want to view.

From the HELP Command:

To view the online documentation for your specific application, click HELP. (Consult the documentation specific to your application for more information.)

From the Documentation CD-ROM:

1. Mount the documentation CD-ROM.
2. Point your browser to:

   CDROM_mount_point`/htmldoc/mainmenu.html` (UNIX)

   CDROM_Drive:`\htmldoc\mainmenu.html` (Windows NT)

Printing Documentation

A PDF (Portable Document Format) file is included on the CD-ROM for each online book. See the first page of each online book for the document number referenced in the PDF file name. Check with your system administrator if you need more information.

You must have Acrobat Reader installed to view and print PDF files.

The default documentation directories are:

- $EPD_HOME`/data/html/pdf/`doc_number`.pdf` (UNIX)
- %EPD_HOME%`\data\html\pdf\`doc_number`.pdf` (Windows NT)

Resources and Services

For resources and services to help you with PTC (Parametric Technology Corporation) software products, see the *PTC Customer Service Guide*. It includes instructions for using the World Wide Web or fax transmissions for customer support.

Documentation Comments

PTC welcomes your suggestions and comments. You can send feedback in the following ways:

- Send comments electronically to `doc-webhelp@ptc.com`.
- Fill out and mail the PTC Documentation Survey located in the *PTC Customer Service Guide.*

Chapter 1

CMOM Overview

This chapter presents an overview of the CMOM communication handler. The following topics are presented:

- What is CMOM
- CMOM Client and Agent
- Customizing with Perl

What is CMOM

The term CMOM stands for Common Message-Oriented Middleware. CMOM is the mechanism by which the desktop user interface (GUI) and the agent applications (nonGUI) communicate. It is the central communication handler for EPD.Connect and all its constituent applications.

When you invoke EPD.Connect, a CMOM server process is started. This process handles intercommunication between the EPD.Connect components.

All EPD.Connect customization is based on using the CMOM message passing mechanism.

Key points to understand about CMOM are:

- The CMOM message server must be running.

  This happens automatically when you invoke EPD.Connect.

- Applications that you have opened from within EPD.Connect communicate within the scope of a CMOM_DOMAIN.

  The `CMOM_DOMAIN` for EPD. Connect is "EPDCONNECT".

- Communication cannot occur across `CMOM_DOMAINs`.

Please note: See "Legal Target Destinations" on page 4-4 to learn where legal CMOM targets are defined.

Required Variables

The variables described in this section are only required if you are running your script outside of the EPD.Connect framework.

UNIX Users

UNIX users can set the following variables:

```
setenv CMOM_DOMAIN "EPDCONNECT"
setenv CMOM_DISPLAY "hostname:0.0"
```

Please note: The `CMOM_DISPLAY` variable should have the same value as the `DISPLAY` variable. See the example Perl code in "Using Aw.pm" on page 2-6.

You can optionally set these variables in your `cvepd.ini` file.

Please note: See *Installing EPD.Connect and EPD Roles, and EPD.Visualizer* for more information about `cvepd.ini`.

Windows Users

On Windows 98, set `CMOM_DOMAIN` in `autoexec.bat` as follows:

```
set CMOM_DOMAIN="EPDCONNECT"
```

On Windows NT, set `CMOM_DOMAIN` using the following:

Control Panel>System>Environment

Please note: See *Installing EPD.Connect and EPD Roles, and EPD. Visualizer* for information about `cvepd.ini`.

Supported Messaging

Two types of messaging are supported. Both are specific to a CMOM client. See “Client” on page 1-4.

- Fire and forget - send
- Fire and wait - request

For information on `Opt.pm` supported Perl calls, see “Opt.pm and CMOM Calls” on page 4-2.

CMOM Client and Agent

There are three classifications of CMOM - agent, client, and server. You can create a custom CMOM client. You cannot create a custom CMOM agent or CMOM server.

Agent

The CMOM agent is a daemon process that monitors CMOM events. When you create custom programs, you typically send instructions to existing agents. Agents enable you to drive an application.

Appendix A, “Destinations, Format, and Commands” lists target destinations.

Client

A CMOM client is an application that is CMOM-enabled. The term “CMOM-enabled” simply means that it has established a connection to the CMOM server.

A client can communicate with any CMOM application in it's `CMOM_DOMAIN`. An example is EPD.Connect itself, or any application invoked from within it - such as CM or CAMU.

A simple CMOM client is a program that sends messages to other applications within its `CMOM_DOMAIN`. Unlike an agent, it does not monitor and act on CMOM events. It tells other CMOM-enabled applications to do something.

A client can also be an agent, for example EPD.Connect. In this case, the client can send and receive messages and listen for CMOM events.

This document describes how to write CMOM client programs that can be used to customize EPD.Connect.

Server

The CMOM server is the single messaging process for a `CMOM_DOMAIN`. It routes instructions to all the applications within a `CMOM_DOMAIN`. The binary `optmsgsrv` (UNIX) and `optmsgsr.exe` (Windows 98 and Window NT) is the CMOM server program.

Customizing with Perl

You must have a working knowledge of Perl in order to customize EPD.Connect.

All the functionality you need to create a simple CMOM client is available using the Perl language. Supplied Perl modules and extensions provide this capability.

Parts of EPD.Connect were implemented using Perl. For example, the Menu Customizer and the CM front end were written in Perl. You can find these Perl scripts in `$EPD_HOME/scripts` or `$EPD_HOME/data`. Files that end with the extension `.pl` are Perl scripts.

A CMOM Perl extension is provided. However, you do not need to use it. You can do all your customization work using the higher level interfaces provided by the `Aw.pm` and the `Opt.pm` Perl modules.

Please note: `Aw.pm` is described in “Using Aw.pm” on page 2-6. `Opt.pm` is described in “Using Opt.pm” on page 4-2.

Decision to Use Perl

Perl was chosen for two reasons.

- It is portable across the UNIX and Windows platforms.
- You do not need to compile or load to perform customization.

Chapter 2

Customizing the Interface

This chapter describes how to use AccessWare to customize the graphical user interface. It assumes that you have a working knowledge of AccessWare. It contains additional informations not available in the *AccessWare Quick Reference*. The following topics are presented:

- Locating Supplied Perl Modules
- Using Awmsg.pm
- Using Aw.pm
- Adding a Menu File

Locating Supplied Perl Modules

This section names and describes the four supplied Perl modules.

UNIX Users

The following files are required for customizing EPD.Connect for a UNIX user. They reside at the following location:

```
$EPD_HOME/CVperl/lib/perl5/site_perl
```

Table 2-1 Key Files

File name	Description
`Aw.pm`	AccessWare API package
`Awmsg.pm`	AccessWare Repository API package
`Opt.pm`	Optegra Message API package
`Cmom.pm`	Optegra CMOM API package

Windows Users

The following files are required for customizing EPD.Connect for a Windows user. They reside at the following location:

```
$EPD_HOME/CVperl/lib/
```

Table 2-2 Key Files

File name	Description
`Aw.pm`	AccessWare API package
`Awmsg.pm`	AccessWare Repository API package
`Opt.pm`	Optegra Message API package
`Cmom.pm`	Optegra CMOM API package

Using Awmsg.pm

The `Awmsg.pm` package is a platform independent variable setting and text externalizing module.

Use `Awmsg.pm` to externalize text and variables into initialization files (`.ini` files) for EPD.Connect to use on both the UNIX and Windows platforms. Customized Perl programs can use the same variable settings as EPD.Connect by registering the `.ini` files.

Please note: The `.ini` are files are text files. The line format is:

`KEYWORD`=VALUE

For example, create a file named `foo.ini` file with the following lines:

`MYVALUE`=1

`DEMOSTRING`=Hello

`DEMOSTRING2`=$DEMOSTRING world

The following Perl program registers the file then gets and prints the values.

```
#!/usr/CVperl/bin/perl
#
#
use Awmsg;
# Register the file
  Awmsg::register("foo.ini");
#
# Get the variables defined in foo.ini as well as HOME from the
environment
  $myvalue = Awmsg::get_global('MYVALUE');
  $demostring = Awmsg::get_global('DEMOSTRING');
  $demostring2 = Awmsg::get_global('DEMOSTRING2');
  $home =  Awmsg::get_global('HOME');
  print "HOME=$home MYVALUE=$myvalue DEMOSTRING=$demostring
DEMOSTRING2=$demostring2\n";
#
# End of program
```

Order of Precedence

This section describes the order that Awmsg::get_global uses to find the symbol.

If a name is duplicated in multiple `.ini` files, EPD.Connect uses the first reference found in the `.ini` file register. However, variables defined using environment variables take precedences over `.ini` settings. The supplied `.ini` files are found at `$EPD_HOME/data/reposit` and `$EPD_HOME/data/reposit/$LANG`.

The order in which EPD.Connect finds symbols is:

1. Environment variable - set by the user prior to invoking EPD.Connect
2. `cvepd.ini` - set by or for the user in order to tailor their environment

   Please note: The `cvepd.ini` file is mandatory for Windows users and optional for UNIX users. If used by a UNIX user, it must reside in their home directory. See *Installing EPD.Connect, EPD Roles, and EPD.Visualizer* for full description of `cvepd.ini`.
3. Window manager file - set by an administrator
   - `cfgw98.ini` (Windows)
   - `cfgmotif.ini` (UNIX)
4. Screen Resolution file - set by an administrator
   - `cfg1024.ini`
   - `cfg800.ini`
   - `cfg1280.ini`
5. `epdconn.ini` - set by an administrator

For more detailed information about precedence order and EPD.Connect `.ini` files, see *Installing EPD.Connect, EPD Roles, and EPD.Visualizer.*

Sample Awmsg.pm Perl Code

Customized Perl scripts that are dependent on EPD.Connect globals can use a function in the reporting package to ensure the script registers `.ini` files in this same order. To do so, add the following lines to the Perl script:

```
#
# Adjust @INC path so modules can be found.
use Awmsg;
$incpath = $ENV{"EPD_HOME"}."/data/explorer/library";
unshift( @INC, $incpath);
# Include the module. Use a “require” statement not a “use”
statement
# because we have modified @INC at runtime.
require "expreprt.pl";
# Initialize the globals in the same manner as Connect
explorer::init_globals();
```

Using Aw.pm

To drive an AccessWare application from Perl, the following is needed:

- The AccessWare application is up and running.
- The Perl script uses the same `CMOM_DOMAIN` and `CMOM_DISPLAY` values as the target AccessWare application.
- The script knows the CMOM target destination name of the AccessWare application.

  Please note: The target destination of EPD.Connect for AccessWare messages is "NavigatorScript".
- The script needs to know the QID names of the widgets to manipulate.

Sample Aw.pm Perl Code

The following sample Perl code demonstrates how to use the `Aw.pm` packages to display one of EPD.Connect's windows.

```
#!/usr/CVperl/bin/perl
#
#
# Sample which makes the EPD.Connects “Open Configuration”
# window appear.
# Generally, to drive AccessWare from a perl script you need
# to know the following.
#   1)The CMOM target destination name of the
#     AccessWare application.
#   2)The QID values to the windows/widgets you want
#     to manipulate.

use Awmsg;
use Aw;

#
# Since this script is run outside COnnect environment, set
# required CMOM variables. Needed for UNIX and PC.
$ENV{'CMOM_DOMAIN'} = “EPDCONNECT”;

# Needed for UNIX.
$ENV{'CMOM_DISPLAY'} = $ENV{'DISPLAY'};

# In this simple case, it was not necessary to register any
# .ini file since this script didn't need access to the values
# to do what it needs to do.

# "NavigatorScript" is the CMOM target destination when
# you want to send an AccessWare Message to the EPD.Connect
Aw::set_macro_host("NavigatorScript");
```

```
# Required - performs initialization
Aw::start_macro();

#  Need to know the QID name of the item I want to show
#  Use show_item to make window display
Aw::show_item("CONFIGURATION_PANEL");

# Required - performs cleanup
Aw::exit_macro();
```

Differences between the Perl and C API Calls

The following code fragments illustrate differences between how C and Perl call the same function. In this example, the argument is a QID value.

C code line:

```
aw_show_item("CONFIGURATION_PANEL");
```

Perl code line:

```
Aw::show_item("CONFIGURATION_PANEL");
```

Please note: To determine what you can do with AccessWare widgets, see *AccessWare Function Reference* and *AccessWare User Guide*.

Accessing and Writing Values Using Aw.pm

You can access and write variables (Accessware objects) using their values (QID names) to the window. A sample piece of code is provided below:

```
#!/usr/CVperl/bin/perl
#
#
# Set and get values form an AccessWare window from perl.

use Awmsg;
use Aw;

$ENV{'CMOM_DOMAIN'} = "EPDCONNECT";
$ENV{'CMOM_DISPLAY'} = $ENV{'DISPLAY'};

Aw::set_macro_host("NavigatorScript");

# Required - performs initialization
Aw::start_macro();

#  Need to know the QID name of the item I want to show
```

```
#  Use show_item to make window display
Aw::show_item("CONFIGURATION_PANEL");

#  Set the name of configuration - AW type "QTEXT" QID
#  is "CONF_NAME"
Aw::set_item_value("CONF_NAME", "fred");

#  Set File Mode to Read Only - AW type "QCHOICE" QID
#  is "CA_OPEN_MODE"
Aw::set_item_value("CA_OPEN_MODE", 1);

#  Get the APPLICATION type
$apptype = Aw::get_item_value("CONF_TYPE");

print "$apptype\n";

# Required - performs cleanup
Aw::exit_macro();
```

Adding a Menu File

You can add an entry to an existing menu.

Locating Menu Files

The standard menu files are located at the following:

```
$EPD_HOME/data/menus/$LANG/*.men
```

The custom menu files are located at the following:

```
$EPD_HOME/data/custmenu/$LANG/*.men
```

Other menu sets are as follows:

```
$EPD_HOME/data/camu/menus/$LANG/*.men
$EPD_HOME/data/camu/custmenu/$LANG/*.men
$EPD_HOME/data/cmedit/custmenu/$LANG/*.men
$EPD_HOME/data/ca/menus/$LANG/*.men
```

Adding a Menu File to EPD.Connect

Use the menu format in `$EPD_HOME/data/menus/C/ca_hview.men` as a template. The menu format is shown below.

Option on Pulldown Menu	Action to Perform
Structure Window ...	6903
Structure Overview ...	6902
<SECTION>	
Information Browser ...	SY,$AW_PERL_EXEC $CA_SCRIPTS/InfoBrow.pl
STARTUP&	
Data Browser ...	6817
Data Browser Item Info..	8107
<SECTION>	
Component Status ...	8100
Memory Status ...	8105
Transfer Requests ...	8609
Indicator Key ...	8390
<SECTION>	
Attachment Window ...	6915

Option on Pulldown Menu	Action to Perform
Clipboard ...	7700
<SECTION>	
AW Info Browser Customizer ...	SY,$EPD_HOME/bin/infbcust&
Toolbar Customizer ...	6050
Menu Customizer ...	SY,$AW_PERL_EXEC $CA_REPORT_DIR/menuadd.pl SHOW
Heading Customizer ...	7311

This example shows three types of menu calls:

- Structure Window 6903

  This is the original AccessWare integer action number scheme. When you build a standalone AccessWare application, you must provide a `user_exit.c` function. This function passes the action numbers. Based on the unique action number, the `user_exit` function performs the required tasks.

  The numbers shown reflect the action numbers of the EPD.Connect `user_exit` function.

- AW Info Browser Customizer SY,`$EPD_HOME/bin/infbcust&`

  When you select this menu option, everything after SY is passed to a system call. In this case, the Info Browser program is started in background.

- Menu Customizer SY,$AW_PERL_EXEC $CA_REPORT_DIR/menuadd.pl SHOW

  This call style runs a Perl program. It tells the system to prepare to run a Perl program ($AW_PERL_EXEC), passes the Perl program name ($CA_REPORT_DIR/menuadd.pl) to Perl, and passes an argument (SHOW) to the Perl program.

  It calls whatever Perl program is passed as an argument to the menuadd.pl program. The argument in this example is SHOW.

  This method works on both the UNIX and Windows platforms.

Chapter 3

Customizing Reports

This chapter describes how to create customized reports. The following topics are presented:

- Creating Customized Reports
- Menu Entries and lookup.rep Format
- Creating Interactive Reports
- Customizing Component Names in a Report
- Report Utility Packages Review
- Sample Program

Creating Customized Reports

You can write programs to generate reports based on nodes in the Product Structure window. You can specify none, single, multiple, or all nodes.

When you select a custom report menu, the system uses your custom report program. It generates a file containing node information from the Product Structure window. When the program is finished, it displays the results in the window.

You can optionally output to a file if the file name is specified as a redirect in your custom program. You can optionally structure your program to open a menu for entering a specific report output name.

Locating Required Files

The following file contains a reports table:

```
$EPD_HOME/data/reports/lookup.rep
```

Perl utility packages to help you write custom report Perl programs are located as follows:

```
$EPD_HOME/data/explorer/library/navreprt.pl
$EPD_HOME/data/explorer/library/navrep2.pl
$EPD_HOME/data/explorer/library/expreprt.pl
```

Existing report programs are located as follows:

```
$EPD_HOME/data/explorer/reports/*.pl
```

Menu Entries and lookup.rep Format

Custom report menu entries are identified by action numbers 6500 and 6501. These numbers ensure that an input file to the reporting program is generated and that a results window is displayed. In both cases, lines that begin with a crosshatch sign (#) are comments.

Action Number 6500

The format of the 6500 menu entry is shown below:

```
[Menu text] [Action #] [Report type] [Tree data wanted] [Program]

Menu text   = As in all menus, the text display for the menu pick
Action #    = 6500
Report type = APPLICATION
                 Program called with one extra argument -
                 the filename of the file which has
                 structure information.
            = REPORT,PERLREP
                 Program called with two extra arguments - the
                 filename of the file which has structure
                 information and the filename where the
                 report output should be written.
Tree data   = NONE    No data extracted from structure.
            = SINGLE  Pass information for a single node in
                      the structure.
            = MULTI   Pass information for selected node in
                      the structure.
            = ALL     Pass information for all the nodes in
                      the structure.
            = REPORT  Pass information for the selected node in
                      the Report Viewer
Program     = Filename of program to run with arguments.
```

Action Number 6501

The format of the 6501 menu entry is shown below:

```
[Menu text] [Action #] [Report #]

Menu text   = As in all menus, the text display for the menu pick
Action #    = 6501
Report #    = Report number defined in lookup.rep definition file.
```

Format of lookup.rep

The format of `$EPD_HOME/data/reports/lookup.rep` is shown below:

```
[Report #] [Report type] [Tree data wanted] [Program]

Report #    = Integer report number used for lookup. This should
              be unique.
Report type = APPLICATION
                    Program called with one extra argument -
                    the filename of the file which has
                    structure information.
            = PERL
                     Program called with two extra arguments -
                     the filename of the file that has structure
                     information and the filename where the
                     report output should be written.
Tree data   = NONE    No data extracted from structure.
            = SINGLE  Pass information for a single node in
                      the structure.
            = MULTI   Pass information for selected node in
                      the structure.
            = ALL     Pass information for all the nodes in
                      the structure.
Program     = Filename of program to run with arguments.
```

Creating Interactive Reports

You can create custom programs for reports that, when displayed in the Report Viewer window in a specified format, can interact with the Product Structure window and the EPD.Visualizer window.

To do this,

- Create an index file along with the report file. The index file must have the corresponding component detail for each component displayed in the report.
- Your custom program that generates the report must have a parameter specifying the name of the index file. See “Sample Program” on page 3-9.

Sample Index File

Consider the following custom report:

```
ASSEMBLY : ADJ-COMPLETE1
REVISION : NULL
*****************************************************************
COMPONENT NAME     EDM FILE NAME   PART NUMBER     DESCRIPTION
*****************************************************************

adj-complete1_1   ADJ-COMPLETE1     N/A               N/A
side-plate-sub_1  SIDE-PLATE-SUB    N/A               N/A
rivet_1           ADJ.RIVET         N/A               N/A
rivet_3           ADJ.RIVET         N/A               N/A
side-plate_1      ADJ.SIDE-PLATE    N/A               N/A
gear_1            ADJ.GEAR          N/A               N/A
worm_1            ADJ.WORM          N/A               N/A
arm_1             ADJ.ARM           N/A               N/A
side-plate-sub_2  SIDE-PLATE-SUB    N/A               N/A
rivet_4           ADJ.RIVET         N/A               N/A
rivet_5           ADJ.RIVET         N/A               N/A
rivet_6           ADJ.RIVET         N/A               N/A
side-plate_2      ADJ.SIDE-PLATE    N/A               N/A
shaft_1           ADJ.SHAFT         N/A               N/A


************END************
```

The corresponding index file must be as follows:

```
NULL
NULL
NULL
NULL
NULL
NULL
~adj-complete1_1
adj-complete1_1~side-plate-sub_1
adj-complete1_1~rivet_1
adj-complete1_1~rivet_2
adj-complete1_1~rivet_3
adj-complete1_1~side-plate_1
adj-complete1_1~gear_1
adj-complete1_1~worm_1
adj-complete1_1~arm_1
adj-complete1_1~side-plate-sub_2
adj-complete1_1~rivet_4
adj-complete1_1~rivet_5
adj-complete1_1~rivet_6
adj-complete1_1~side-plate_2
adj-complete1_1~shaft_1
NULL
NULL
NULL
```

Please note: In the index file:

- Leave empty lines that do not require interaction. In the preceding sample, `NULL` indicates such lines that must be empty in the actual index file.
- Define each component in the PARENTASSY~CLASSNAME_INSTANCE format corresponding to each line in the report.
- Separate all multiple component entries on the same line with a tab. The standard BOM report uses this feature.

Customizing Component Names in a Report

To customize component names in a report, set the following variables in the `epdconn.ini` file:

```
CA_REPORT_CUST_COMPONENT='$CLASS-NAME_$INSTANCE'
CA_REPORT_CUST_STRUCTURE='$CLASS-NAME_$INSTANCE'
```

The above settings show the default values.

Please note: Each attribute value separated by any character, must be preceded by $.

Sample Code

The following sample, if added to your custom program, enables customized component names:

```
require "navrep2.pl";

# Unpack input report

    $nitems = &navigator'nav_unpack_report("$ARGV[0]");
for ( $i = 0; $i <= $nitems; $i ++ )
    {
        %item = &navigator'nav_get_item( "$i" );
        $cust_name[$i]  = $item{"CA-REPORT-CUST-NAME"};
    }
```

Report Utility Packages Review

This section describes the three supplied report utility Perl programs.

expreprt.pl

This provides a function to enable a Perl program to use the same global settings as the EPD.Connect program.

```
explorer::init_globals();
```

navreprt.pl

This helps process information contained in a file extracted from node information in the product structure. It does the following:

1. Reads an input file that contains information extracted from the Product Structure window and stores it in arrays.

   ```
   $nitems = navigator::nav_unpack_report($filename);
   ```

2. Returns a hash that contains header (root node) information when called after the `navigator::nav_unpack_report` function.

   ```
   %header = navigator::nav_get_header();
   ```

3. Returns a hash that contains the values for the item number. The hash key matches the keywords in a Product Structure file.

   ```
   %item  =  navigator::nav_get_item($itemno);
   ```

navrep2.pl

This is the same as `navreprt.pl` with a few more capabilities. Its version of `nav_unpack_report` reads more attributes for the input file. It also has an extra set of functions for processing a second report file.

```
$nitems =  navigator::nav_unpack_report($filename);
   %header =  navigator::nav_get_header();
   %item =    navigator::nav_get_item($itemno);
              navigator::create_lookup_table():

   $nitemsc = navigator::nav_unpack_comp_report($filename);
   %item =    navigator::nav_get_comp_item($itemno);
              navigator::create_comp_lookup_table();
```

Sample Program

```
#!/usr/CVperl/bin/perl
#
# Since this is a report program, it will be passed three
# arguments.
# The name of the file which contains information extracted from
# the nodes in the product structure window and the filename to
# write the output which will be displayed in the "REPORT VIEWER"
# window and also the index filename to enable the interaction.
# Adjust @INC path so modules can be found.
use Awmsg;
$incpath = $ENV{"EPD_HOME"}."/data/explorer/library";
unshift( @INC, $incpath);

# Include the module. Use a "require" statement not a "use"
# statement because we have modified @INC at runtime.
require "expreprt.pl";
require "navreprt.pl";
require "navrep2.pl";

# Initialize the globals in the same manner as Connect
explorer::init_globals();

#
# Create report which displays CLASSNAME and INSTANCE

#
# Use navigator function to read the file
$nitems = navigator::nav_unpack_report($ARGV[0]);
 $sep1 = "~";
 $sep2 = "_";

#
# Open output file
open(REPORT,">$ARGV[1]") or die "Can't generate REPORT file";
open(INDEX,">$ARGV[2]") or die "Can't generate INDEX file";
#
print REPORT "This is my custom test report.\n\n";
print INDEX "\n\n";

# Loop thru hash created by nav_unpack_report and write to the
# output file
#

for $i (0 .. $nitems ) {

# nav_get_items returns a hash where the hash key is the keyword.
  %items = navigator::nav_get_item($i);
# Print component names as defined by the CA_REPORT_CUST_COMPONENT
# variable in epdconn.ini
  print REPORT "$items{'CA-REPORT-CUST-NAME'} \n";
```

```
  print INDEX
"$item{'PARENT-ASSY'}$sep1$items{'CLASS-NAME'}$sep2$items{'INSTAN
CE'}\n"

}
print REPORT "End of report\n";
print INDEX "\n";

close(REPORT);
close (INDEX);
```

Chapter 4

Sending Commands to Agents Using Opt.pm

This chapter describes how to send commands to agents using the `Opt.pm` module with Perl scripts. The following topics are presented:

- Using Opt.pm
- Callable Functions from Windows Programs

Using Opt.pm

EPD.Connect and certain agents have command string APIs. They use the `Opt.pm` package to route these strings to the a CMOM target destination. The legal command strings accepted by a destination are destination-specific.

For example, you can use the "`OpenNavigator`" target to manipulate and retrieve product structure information in the EPD.Connect structure window.

Please note: All the environment restrictions described for the `Aw.pm` in "Order of Precedence" on page 2-4 are applicable for `Opt.pm`. However, for `Opt.pm`, the destination need not be an AccessWare application.

Opt.pm and CMOM Calls

The `Opt.pm` package supports the following two calls from Perl:

- `Opt::ci`

  `Opt::ci` is the request function call that supports fire and wait. It routes the message to the target destination and waits for a reply. It returns a status code and a message string on its function return.

  An example Perl code fragment is:

  ```
  ($status, $message) = Opt::ci($cmom_target, $cmd_string,
  $remote_host);
  ```

- `Opt::send`

  `Opt::send` is the send function call that supports fire and forget. Use it when you want to send non blocking message to CMOM targets and don't care about the result.

  An example Perl code fragment is:

  ```
  Opt::send($cmom_target, $cmd_string, $remote_host);
  ```

The third argument on both calls (`$remote_host`) is optional. It is only used by the PC. Use it when the agent program is not available on the PC.

Sample Opt.pm Perl Code

The following two samples show how to use `Opt.pm` in a Perl script:

Sample Program 1

```
#!/usr/CVperl/bin/perl
#
#
# Sample program which tells EPD.Connect to load a Product
Structure
# file into the Connect "Structure Window".
#   1)The CMOM target destination name of the application.
#   2)Knowledge of the legal commands which the target
#     application accepts.

use Opt;

#
# Since this script is run outside Connect environment, set
# required CMOM variables. Needed for UNIX and PC.
$ENV{'CMOM_DOMAIN'} = "EPDCONNECT";

# Needed for UNIX.
$ENV{'CMOM_DISPLAY'} = $ENV{'DISPLAY'};

# In this simple case, it was not necessary to register
# any .ini files
# since this script didn't need access to the values to do
# what it needs to do.

#
# Command string to open a Product Structure file which
# is on the local disc and in the user's create directory.
$cmd_string="OPEN APPLICATION=PS ITEMNAME=PSTESTFILE REFERENCE=ALL
MODE=1
LOCATION=1 DIRTYPE=1";

# "OpenNavigator" is the CMOM target destination when you
# want to send a command string to EPD.Connect.
($status, $message) = Opt::ci("OpenNavigator", $cmd_string );

# Print the result
print "Error code=$status Error message=$message\n";

# End of program
```

Please note: Appendix A, “Destinations, Format, and Commands” describes all keywords for writing calls to the EPD.Connect Product Structure window (using `"OpenNavigator"`) and Product View window (using `"OpenVisualizer"`).

Sample Program 2

```
#!/usr/CVperl/bin/perl
#
#
# The following script is a sample of how to get help from the
# "OpenNavigator" target.

use Opt;

#
# EPD.Connect needs to be running for this to work.
# Since this script is run outside Connect environment, set
# required CMOM variables. Needed for UNIX and PC.
$ENV{'CMOM_DOMAIN'} = "EPDCONNECT";

# Needed for UNIX.
$ENV{'CMOM_DISPLAY'} = $ENV{'DISPLAY'};

#
# If no argument passed then run HELP to get command list
if ( $ARGV[0] ne "" ) {
   $cmd_string = "HELP COMMAND=$ARGV[0]";
}
else {
   $cmd_string="HELP";
}

# "OpenNavigator" is the CMOM target destination when
# you want to send a command string to EPD.Connect.
($status, $message) = Opt::ci("OpenNavigator", $cmd_string );

# Print the result
print "$message\n";
```

Legal Target Destinations

You can send CMOM messages to any legal destination that is registered with the message server that is running in a specific `CMOM_DOMAIN`.

The legal destinations for the `CMOM_DOMAIN` of EPD.Connect are defined in `$EPD_HOME/data/optegra.msg`.

Please note: See “What is CMOM” on page 1-2 for other information on the `CMOM_DOMAIN` variable.

Put the destination name for the `Opt.pm` function in the first column of each line.

See Appendix A, “Destinations, Format, and Commands” for more information.

Callable Functions from Windows Programs

You can call the `optci` and `optsend` functions from Visual Basic. These function calls are provided in the OPTDM.DLL.

optci

The `optci` function sends a CMOM string - either a command, an error text, or a status of processing information - to the destination application. The function then waits for a reply.

The `optci` function is available in the `optpm.dll` library.

Syntax

```
int optci (char * dest, char * data, char * desthost,
           char * status, char * reply);
```

Parameters

Parameters for the `optci` function are described below.

Parameter	Type	Description
dest	Input	Destination application name. Enter an application name (for example; an Optegra application) up to 39 case-sensitive characters.
data	Input	Data String. Enter a string upto 1024 case-sensitive alphanumeric characters. This could be any processing information.
desthost	Input	Destination machine name. Enter a machine name up to 39 case-sensitive characters. This is the machine where your destination application is running. To send a message while the application is running on the same machine (local), use the `desthost` value as "NOHOST". If you do not enter a valid machine name, an error message appears.
status	Output	Output parameter status. This returns the actual status of the command. Maximum character count is 1024.
reply	Output	Output parameter reply. This returns a brief description of the status using a reply message string. Maximum character count is 1024.

See "Visual Basic Requirements" on page 4-8 for more information.

Requirements and Special Considerations

For the `optci` function, allocate memory for `status` and `reply` to 1024+1 using the following:

```
DIM status As String * 1025
DIM reply As String * 1025
```

Please note: If the memory is not allocated, the application might crash or will give unexpected results later.

Return Values

If the `optci` function succeeds, it returns a value 0. On failure, it will return an error code. For more information, see “Error Codes” on page 4-8.

Example

Given next is a sample Visual Basic code for the `optci` function:

```
Declaration for optpm functions
----------------------------
Declare Function optci Lib "optpm.dll"
(ByVal dest$, ByVal data$, ByVal desthost$, ByVal status$, ByVal
reply$) As Integer

Declare Function optsend Lib "optpm.dll"
(ByVal dest$, ByVal data$, ByVal desthost$) As Integer

Sample code for calling optci
----------------------------

Private Sub cmdRequest_Click()
Dim stat As Integer
Dim ret_stat As String * 1025
Dim ret_reply As String * 1025


Screen.MousePointer = 11
desthost$ = txtHost.Text

pnlStatus = ""
pnlReply = ""

stat = optci("CMAgent", "cisignon userid='optegraUserId'
userpw='optegraUserPassword' dbname='databaseName'", desthost$,
ret_stat, ret_reply);
Screen.MousePointer = 1
If stat = 0 Then
   pnlStatus = ret_stat$
   pnlReply = ret_reply$
```

```
Else
   MsgBox "Request failed  : " + "Error message is " + Str(stat)
End If
End Sub
```

optsend

The `optsend` function sends a CMOM string (either a command, error text, or status of processing information) to the destination application.

The `optsend` function is available in the `optpm.dll` library.

Syntax

```
int optsend (char * dest, char * data, char * desthost);
```

Parameters

Parameters for the `optsend` function are described below.

Parameter	Type	Description
dest	Input	Destination application name. Enter an application name (for example, an Optegra application) up to 39 case-sensitive characters.
data	Input	Data String. Enter a string up to 1024 case-sensitive alphanumeric characters. This could be any processing information.
desthost	Input	Destination machine name. Enter a machine name up to 39 case-sensitive characters. This is the machine where your destination application is running. To send a message while the application is running on the same machine, use desthost value as “NOHOST”. If a valid machine name is not entered an error message appears.

See “Visual Basic Requirements” on page 4-8 for more information.

Error Codes

A table with possible error codes and their explanations follows.

Error codes	Explanations
1	TCP connection failed.
1009	Destination application was not found.
1012	Message server is not running.
1014	Destination application is not running.
1014	Window registration failed.
1015	Agent failed to initialize.
1018	Agent is already running.
1050	Invalid input for parameters.
1107	Client time-out occurred.
1108	System timer is not available.

Visual Basic Requirements

Declare all parameters for the `optci` and `optsend` functions as `String` in Visual Basic.

Allocate 1024+1 bytes for status and 9801+1 bytes for reply, using Visual Basic application, before issuing calls.

You must write the declaration as:

```
DIM status As String * 1025
DIM reply As String * 9802
```

Please note: If not done, the application might crash or give unexpected results.

Chapter 5

Customizing the Client

This chapter describes the functions supported by the `edmosrv` library that are used for programming and customizing the Vault client.

- Overview of the edmosrv Library
- edmosrv Functions
- SCRAMBLE Library
- Customizing through the Perl Interface

Overview of the edmosrv Library

The `edmosrv` library is used by the Vault server and the Optegra clients to query the Oracle database directly. It exists as a static library on the IBM AIX operating system. On all the other operating systems it is a shared library, `libcvedmosrv.<ext>` where `ext` is the shared library extension. This is called `edmosrv32.dll` on Windows. The client must be linked to `edmosrv32.lib`.

The files get installed as specified in the table below.

Library	Windows	UNIX
`edmosrv32.dll`	Windows system	-
`edmosrv32.lib`(Connect)	`EPD_HOME/lib`	-
`edmosrv32.lib`(Locator sdk)	`EPD_HOME/sdk/lib`	-
`libcvedmosrv.*`	-	`$EPD_HOME/lib`

The include files associated with the library are as follows:

For Windows:

$EPD_HOME`/data/edmosrv/edmopub.h`.
$EPD_HOME`/data/edmosrv/edmopri.h`
$EPD_HOME`/data/edmosrv/edmcall.h`
$EPD_HOME`/data/edmosrv/sqlprnt.h`

For UNIX:

$EPD_HOME`/include/edmosrv/edmopub.h`.
$EPD_HOME`/include/edmosrv/edmopri.h`
$EPD_HOME`/include/edmosrv/nokernel.h`
$EPD_HOME`/include/edmosrv/sqlprnt.h`

It is necessary to define the variable `NO_CVKERNEL` in the custom programs before including the header files. The following libraries are required by the `edmosrv` library for building custom applications:

- UNIX: `cedmpi.a` , `libcvkernel.a`, `liboptscramble.a`

  This is specific only to the IBM platform. Here, it is necessary to link the executable with the library `liboptscramble.a`.

- Windows: `optscram.lib, edmosrv.lib`

The `edmosrv` client library scans the environment variable ANSPATH to find the `pm.config` file to resolve the full domain name of the server. If the `edmosrv` client and the `edmosrv` server are in different network domains, specify the full name of the node in the `pm.config` file. Set the environment variable `EDMOANS` to 1. On the Windows NT platform, set the EDMOANS variable using Start > Settings > Control Panel > System > Environment.

edmosrv Functions

The following sections discuss the functions and the return codes provided by the `edmosrv` library.

SQL Functions

The syntax of all the SQL functions is as follows:

- `EDM_O_STAT edm_o_connect(const char *server, const char *usrid, const char *paswd, *msg)`

  This function connects to Oracle. The function returns `EDM_O_OK` for a successful connection.

- `EDM_O_STAT edm_o_query(const char *query, EDM_O_HANDLE *handle, char *msg)`

  This function assists in issuing a query to the database. The function supports queries like selecting, inserting, updating, or deleting information.

- `EDM_O_STAT edm_o_bindquery(char *query, char *bindvalue1, char *bindvalue2,....,char *0, EDM_O_HANDLE *hand, char *msg)`

  This function assists in issuing a bind variable query to the database. The function supports queries like selecting information. The char *0 argument marks the end of the bind value arguments issued to the database. The function returns `EDM_O_OK` if the query is executed successfully.

Please note: A maximum of 64 bind variables are supported in the query.

- `EDM_O_STAT edm_o_fetch(EDM_O_HANDLE *handle, char *msg)`

  This function fetches the results of the query, one row at a time. When all the rows are fetched, the function returns `EDM_O_NFOUND`.

- `EDM_O_STAT edm_o_close(EDM_O_HANDLE *handle, char *msg)`

  This function closes the query and removes the memory allocated for the operation.

- `EDM_O_STAT edm_o_disconnect(char *msg)`

  This function assists in disconnecting from the server.

- `EDM_O_STAT edm_o_commit(char *msg)`

  `EDM_O_STAT edm_o_rollback(char *msg)`

  The preceding two functions update the changes made to the database, and retain the results permanently.

Return Codes

The following table shows the possible return codes and their explanations.

Return Code	Explanation
EDM_O_OK	Successfully connected to the server
EDM_O_SON	Already connected to the database
EDM_O_NOUSER	No user name supplied
EDM_O_NOPW	No user password supplied
EDM_O_NOCHAR	Query returned unsupported data type
EDM_O_NSON	Not connected to database
EDM_O_NOCONNECT	Failed to connect to server
EDM_O_NOSOCK	Bad socket (Communications error on server)
EDM_O_NOFREE	No free RPC (Remote Procedure Call) program numbers
EDM_O_EXPIRED	Licence has expired
EDM_O_NFOUND	No more data to fetch

In addition to the return codes mentioned above, the Oracle return codes and messages are also returned.

Procedure for SELECT

For SELECT, use the routines as follows:

1. Connect to the server using `edm_o_connect(server,usrid,paswd,msg)`. The password is scrambled and passed to the server so that it is not visible to malicious users on the network.
2. Issue a query using `edm_o_query(query,handle,msg)`.
3. Fetch the results one row at a time using `edm_o_fetch(handle,msg)`.
4. Close the query using `edm_o_close(handle,msg)`.
5. Repeat steps 2, 3, and 4 till all the queries are completed.
6. Disconnect from the server using `edm_o_disconnect(msg)`.

Procedure for UPDATES

For UPDATES, use the routines as follows:

1. Connect to the server using `edm_o_connect(server,usrid,paswd,msg)`.
2. Issue the query using `edm_o_query(query,handle,msg)` and fetch the results.
3. Commit the changes using `edm_o_commit(msg)`

   Or

   Roll back using `edm_o_rollback(msg)`.
4. Disconnect from the server using `edm_o_disconnect(msg)`.

Other Functions

- `EDM_O_STAT edm_o_file_query(FILE *, EDM_O_HANDLE *handle, char *)`

  This function is similar to `edm_o_query`. This function expects a file pointer of a file having the SQL query for processing.

Please note: Only one query can be active for a connection at a time.

- `int edm_o_numcol(EDM_O_HANDLE *handle, char *msg)`

  This function returns the number of columns.

- `int edm_o_getnthcol(EDM_O_HANDLE *handle, int col, char *str, int maxlen)`

  This function extracts up to `maxlen` characters from the `col` column and copies the characters to `str`. The `maxlen` includes the terminating null character written to the string.

  The function returns the number of characters written, not including the terminating null. Hence, the maximum value returned is (`maxlen` - 1), but not less than 0.

  This function returns 0 if a problem, such as col larger than the number of available columns, occurs.

- `int32_t edm_o_getnthlen(EDM_O_HANDLE *handle, int col)`

  This function gets the length stored in the B array of long `ints` in the handle. The array is valid only after the query, and it describes the length of the columns.

  The function returns 0 if a problem, such as the value of `col` being larger than the number of available columns, occurs.

Example

The following sample program shows how to sign on to the database and use the `edm_o_bindquery` function to query the interface.

```
EDMOSTAT return_val;
char msg[512];
char ret_param[24]
EDM_O_HANDLE hand;

ret_val = edm_o_connect("scott","tiger","emp_database",msg);
 if(return_val == EDM_O_OK) {
    return_val = edm_o_bindquery("select * from emp where emp_name
=:b1,dept_no =:b2","Scott","801",(char*)0, &hand,msg);
  if(return_val !=EDM_O_OK)
/* Execute failure algorithm */
  else {
   while((return_val=edm_o_fetch(&hand, msg))!=
EDM_O_NFOUND)
{
   /* Do all your operations here */
  if(return_value!=EDM_O_OK) {
/* Execute failure algorithm */
    }
  }
edm_o_close(&hand,msg);
edm_o_disconnect(msg);
}
```

SCRAMBLE Library

This section describes the functions and utilities supported by the Scramble library. The scramble library has been implemented as a static library and is also used by the edmosrv library. It is used for scrambling/unscrambling strings.

The scramble library gets installed as per the table below.

Library Name	Windows	Unix (IBM)
optscram.lib(EPD.Connect)	EPD_HOME/lib	-
optscram.lib(Locator SDK)	EPD_HOME/sdk/lib	-
liboptscramble.a(EPD.Connect)	-	$EPD_HOME/lib

- liboptscramble.a

  This is installed with EPD.Connect in UNIX only on the IBM platform. In all the other platforms, it is available through the edmosrv library.

  The include file for this library is located in:

  $EPD_HOME/include/optscram/scramble.h

Functions

The scramble library provides the following functions:

- int scramble (unsigned char* string_A, unsigned char* string_B)

  Where string_A is a string to be scrambled and string_B is the buffer into which the scrambled string is returned.

  - The memory for both the strings must be allocated by the caller of the function.
  - The memory allocated for string_B must be equal to (3 bytes + (2 x length of string_A)).
  - This function supports Japanese characters.
  - This function always returns 1.

- int unscramble (unsigned char* string_A, unsigned char* string_B)

  Where string_A is a scrambled string, and string_B is the buffer in which the unscrambled string is stored.

  - The memory for both the strings must be allocated by the caller of the function.

- Memory allocated for `string_B` must be equal to ((length of `string_A` / 2) - 1 byte).
- The unscrambling algorithm supports Japanese characters.
- The unscramble function always returns 0.

Utilities

In addition to the preceding functions, the following utility converts a text string to a scrambled string. This utility is useful for manually storing the scrambled string in files or tables.

- `scramexe` (On an operating system based on Unix)
- `scramexe.exe` (Windows)

Usage: `scramexe` string

This utility displays the following output:

```
The password encryption utility for Optegra.
Tiger scrambled to MFHFAKFBCHFK.
```

The client can store the displayed string in a scrambled form.

This utility is being shipped with EPD.Connect and Vault in the $EPD_HOME`/bin` directory. The password in the `epdconn.ini` file is also scrambled using this utility.

Customizing through the Perl Interface

You can use the `edmosrv` library also through the perl interface to customize the client.

The following sample program shows how you can connect to Oracle and print the employee name from the employee file:

```
#! /usr/CVperl/bin/perl

use Edmosrv;

$ip = "hathi";

$ret = Edmosrv'connect($ip, "scott", "tiger");
if ( $ret == 0 ) {
    print "connected to vault $ip\n";
    $ret = Edmosrv'query("SELECT ENAME from EMP");
    if ( $ret != 0 ) {
        print ("Failed to query\n");
        exit(1);
    }
    while ( !Edmosrv'fetch()) {
        $col = Edmosrv'no_columns();
        for ( $i=0; $i<$col; $i++ ) {
            print "output = ", Edmosrv'column_data($i), "\n";
        }
    }
    $ret = Edmosrv'close();
    if ( $ret != 0 ) {
        print "Could not close the query\n";
    }
    $ret = Edmosrv'disconnect();
    print "disconnected from vault $ip\n";
}
else {
    print "could not connect to vault $ip\n";
}
```

Perl Functions for the edmosrv Library

The syntax of all the Perl functions supported by `edmosrv` is given below:

- `debug_on()`

  This function enables the logging of error messages.

- `debug_off()`

  This function disables the logging of error messages.

- `get_debug()`

  This function displays whether the DEBUG mode is ON or OFF. The function returns an integer value.

- `msgtext()`

  This function returns the last error message, as a string of characters.

- `connect(domain, userid, passwd)`

  This function connects to the Oracle database. It returns 0 on success.

- `query(query)`

  This function assists in issuing a query to the database. The function supports queries like selecting, inserting, updating, or deleting information. It returns an integer value.

- `bindquery(query, bindvalue1, bindvalue2,...)`

  This function assists in issuing a query to the database with bind variables. The function supports queries like selecting information. It returns an integer value.

- `commit()`

  `rollback()`

  The preceding two functions update the changes made to the database, and retain the results permanently. These functions return integer values.

- `fetch()`

  This function fetches the results of the query, one row at a time. The function returns 0 on success.

- `close()`

  This function closes the query and removes the memory allocated for the operation. The function returns 0 on success.

- `no_columns()`

  This function returns the number of columns.

- `column_data(col)`

  This function extracts characters from the `col` column and copies the characters to a string.

  This function returns 0 if a problem, such as col larger than the number of available columns, occurs.

- `disconnect()`

  This function disconnects the connection to the database.

- `is_connected()`

  This function returns the status of connection to the database.

Perl Functions for the Scramble Library

The `Scramble` library through the Perl interface provides the following functions:

- `scramble(string)`

  This function returns a scrambled string.

- `unscramble(string)`

  This function accepts a scrambled string and returns an unscrambled string.

For details of the above functions refer to the section “Functions” on page 5-8.

Appendix A

Destinations, Format, and Commands

This appendix describes legal targets, configuration file format, and command syntax. The following topics are presented:

- Legal Target Destinations in the EPD.Connect CMOM_DOMAIN
- CMOM Server Configuration File Format
- OpenNavigator Commands
- EPD.Visualizer and 3D Viewer Commands
- Environment Variables

Legal Target Destinations in the EPD.Connect CMOM_DOMAIN

When you start EPD.Connect, it runs the CMOM server using `$EPD_HOME/data/optegra.msg` as the configuration file.

The configuration file defines the legal target destination for the specific CMOM_DOMAIN serviced by a CMOM server.

To be able to deliver a CMOM message to a target destination, the following conditions must be present:

1. The target destination must be defined in the CMOM server configuration file.
2. The target destination program must be running and be registered with the CMOM server that is servicing the specific CMOM_DOMAIN.

The EPD.Connect startup script starts all required target destinations.

Please note: There is currently no auto-start capability for starting a target destination that is running, but is defined in the configuration file.

CMOM Server Configuration File Format

The CMOM server configuration file is `$EPD_HOME/data/optegra.msg`. Comments are denoted by a pound sign (#).

The data line format is:

Application<TAB>Class<TAB>Name<TAB>Message Name

Data line components are described below:

- Application

  This is the target destination name used by the application code.

- Class and Name

  Class and Name are used by agents when they connect to the CMOM server. They are used by the CMOM server to qualify the target name and are required arguments when an agent establishes a connection to the CMOM server.

  See "CMOM Client and Agent" on page 1-4 for information on these terms.

- Message Type

  The message type defines what kind of message a target destination can receive. It is primarily a tagging mechanism for the message. Only two of message types contain any special meaning.

  - _AW_SCRIPT_MESSAGE

    This message type should be used for any AccessWare application that wants to receive messages using the `Aw.pm` package, see "Using Aw.pm" on page 2-6.

  - _OPTEGRA_CLIENT_MESSAGE

    This is the generic Optegra message type. It is used by agents receiving messages from the `Opt.pm` package, see "Using Opt.pm" on page 4-2.

OpenNavigator Commands

This section lists the commands that can be sent to the OpenNavigator target. The "OpenNavigator" target corresponds to the Product Structure window and EPD.Connect toolbar. By default, errors from these commands are recorded in the EPD.Connect Audit/Transaction Log window.

- `OPEN`
- `OPENNEW`
- `OPENREF`
- `LOAD`
- `LOAD_REF`
- `MEMORIZE`
- `CLOSE`
- `CLOSEREF`
- `ADD`
- `CHANGE`
- `CUT`
- `ATTREDIT`
- `ATTRCUT`
- `REFRESH_ATTR`
- `REORDER`
- `UNDO`
- `HIGHLIGHT`
- `DEHIGHLIGHT`
- `SELECT`
- `DESELECT`
- `SHOW`
- `HIDE`
- `ACCESS`
- `SCROLLTO`
- `REPORT`
- `ACTION`
- `LOCK`

- UNLOCK
- LOCKLINK
- UNLOCKLINK
- CLEAR_HIDDEN
- CLEAR_HIGHLIGHTED
- CLEAR_MEMORY
- CLEAR_SELECTED
- CLEAR_ALL
- CLEAR_ACCESS
- ACCESS_KEY
- CLEAR_ACCESS_KEY
- IS_TREE_LOADED
- IS_LIST_LOADED
- TREE_STATS
- GET_NODE
- REFRESH_NODE
- LIST_STATS
- GET_LIST
- SET_LIST
- REFRESH_LIST
- SET_COMPONENT_TEXT
- SET_TOOLBAR
- MESSAGE
- PUTENV
- SET_DEFAULT
- SIGNON_STATS
- HELP

The following sections provide a quick summary of each command and its keywords.

Notes

1. Keywords in square brackets [] are optional. The use of the optional parameter OBJECTID for the some of the commands is to identify a child component of the given node based on the attribute value. In such cases, the command is executed on the child component that matches with the attribute name of the OBJECTID parameter.
2. CMOM treats any occurence of '=' as a special character if it is not preceded by a backslash (\). Hence, if any external command has a '=' as a part of its value for any argument, the '=' should be preceded by a backslash (\).

OPEN

This opens an existing object of type APPLICATION and displays it in the Product Structure window. The format is:

```
OPEN APPLICATION= ITEMNAME= ITEMREV= [REFERENCE=ALL/no]
[MODE=1-RO,2-RW] [SHOWACCESS=ALL|LOCAL|DATABASE] LOCATION=1-Local
2-DB [DIRTYPE=1-Create, 2-Read, 3-Other] [DIRECTORY=]
[ACTION=GUIONLY]
```

OPENNEW

This opens a new Product Structure of type `APPLICATION`. The format is:

```
OPENNEW APPLICATION= ITEMNAME=
```

OPENREF

This opens a reference assembly. The format is:

```
OPENREF [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [REFERENCE=ALL/no] [MODE=1-RO,2-RW]
[SHOWACCESS=ALL|LOCAL|DATABASE] LOCATION=1-Local 2-DB
[DIRTYPE=1-Create, 2-Read, 3-Other] [DIRECTORY=]
```

LOAD

This forces the system to load a product structure without obeying Product Structure naming rules. The format is:

```
LOAD FILENAME= [SHOWACCESS=ALL|LOCAL|DATABASE]
```

LOAD_REF

This forces the system to load a reference assembly without obeying Product Structure naming rules. The format is:

```
LOAD_REF [PARENTASSY=] [CLASSNAME=] [INSTANCE=] FILENAME=
[SHOWACCESS=ALL|LOCAL|DATABASE] [OPTION=ADD|ADDKIDS]
```

MEMORIZE

This saves the active Product Structure into memory so that another can be loaded. It is primarily used for comparison purposes.

CLOSE

This closes the open Product Structure. The format is:

```
CLOSE [OPTION=FORCE]
```

CLOSEREF

This closes open references. The format is:

```
CLOSEREF [PARENTASSY=] [CLASSNAME=] [INSTANCE=]
[OPTION=STRUCTURE_ONLY]
```

ADD

This adds a node to the existing Product Structure. The format is:

```
ADD PARENT= [PARENTASSY=] CLASSNAME= [INSTANCE=] APPLICATION=
[ITEMNAME=] [ITEMREV=] [NOFF=] [TRIGGER=0/1]
[ORIENTATION=0:0:0:0:0:0] [GLOBAL_ORIENT=0:0:0:0:0:0]
```

CHANGE

This renames an existing node in the tree. The format is:

```
CHANGE ORIG_CLASSNAME= ORIG_INSTANCE= [PARENT=] [PARENTASSY=]
CLASSNAME= [INSTANCE=] [APPLICATION=] [ITEMNAME=] [ITEMREV=]
[NOFF=] [TRIGGER=0|1] [ORIENTATION=0:0:0:0:0:0]
[GLOBAL_ORIENT=0:0:0:0:0:0] [SYMMETRY=0|1|2] [TIM-FLAG=0|1]
[OPTION=ONLY]
```

CUT

This cuts a node from the tree and moves it to the clipboard. The format is:

```
CUT [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [CLIPBOARD=NOSHOW]
[OBJECTID=]
```

ATTREDIT

This changes the attribute value on the node. The format is:

```
ATTREDIT [PARENTASSY=] CLASSNAME= [INSTANCE=] ATTRTYPE=1-Instance
2-Class ATTRNAME= ATTRVALUE= [ATTRLINK=] [TYPE=] [READONLY=0/1]
[VISIBLE=0/1] [SAVED=0/1][OBJECTID=]
```

ATTRCUT

This deletes an attribute from the node. The format is:

```
ATTRCUT [PARENTASSY=] CLASSNAME= [INSTANCE=] ATTRTYPE=1=Instance
2=Class ATTRNAME= [OBJECTID=]
```

REFRESH_ATTR

This refresh attribute information for the node from the database. The format is:

```
REFRESH_ATTR CLASSNAME= INSTANCE=
```

REORDER

This repositions a node in the tree. The format is:

```
REORDER CLASSNAME= INSTANCE= [PARENTASSYABOVE=] [BELOW=]
```

UNDO

This undoes the last clipboard operation.

HIGHLIGHT

This highlights nodes in the tree. The default is all nodes. The format is:

```
HIGHLIGHT [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [OBJECTID=]
```

DEHIGHLIGHT

This de-highlights nodes in the tree. The default is all nodes. The format is:

```
DEHIGHLIGHT [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=]
```

SELECT

This selects nodes in the tree. The default is all nodes. The format is:

```
SELECT [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [OBJECTID=][COLOR=]
```

DESELECT

This de-selects nodes in the tree. The default is all nodes. The format is:

```
DESELECT [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [OBJECTID=]
```

SHOW

This displays the node or sub-tree. It is the opposite of HIDE. The default is all nodes. The format is:

```
SHOW [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=] [ITEMREV=]
[SUBASSY=] [OBJECTID=]
```

HIDE

This hides the node or sub-tree. It is the opposite of SHOW. The default is all nodes. The format is:

```
HIDE [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=] [ITEMREV=]
[SUBASSY=] [CHILDREN_ONLY=0/1] [OBJECTID=]
```

ACCESS

This sets the indicator color on the tree node. Color is an integer number 1-16 as defined in the Access Rules color palette. The format is:

```
ACCESS [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [COLOR=] [OBJECTID=]
```

SCROLLTO

This centers the Product Structure display on the node specified. The format is:

```
SCROLLTO [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [OBJECTID=]
```

REPORT

This runs a predefined `REPORT` program. The format is:

```
REPORT NUMBER=
```

CREATE_CGM

This creates a binary CGM file for the AccessWare tree. The format is:

```
CREATE_CGM TREEQID=CA_TREELIST FILENAME= PAPERSIZE=A4
PLOTSCALE=1.0 ORIENTATION=1-PORTRAIT, 2-LANDSCAPE
ALL_TIERS=1-Yes, 0-No TIER_FROM=0 TIER_TO=2 MULTIPAGE=1-Many
pages, 0-Single large page that contains the complete tree
```

PLOT_CGM

This plots the AccessWare tree using the CGM file. The format is:

```
PLOT_CGM CGMFILENAME= SCALE_TYPE=0-NOAUTOSCALE, 1-AUTOSCALE
[PAPERSIZE=A|B|C|D|E|A0|A1|A2|A3|A4] COPIES=1 PLOTTERNAME=
ORIENTATION=0|90|180|270|360
```

ACTION

This runs code associated with an `ACTION NUMBER` (user exit function). The format is:

```
ACTION NUMBER=
```

LOCK

This locks the node. The tree node displays graphically in a locked state. The format is:

```
LOCK [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=] [ITEMREV=]
[SUBASSY=] [OBJECTID=]
```

UNLOCK

This unlocks a node. The format is:

```
UNLOCK [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [OBJECTID=]
```

LOCKLINK

This locks the relationship between parent and child nodes. The format is:

```
LOCKLINK [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [OBJECTID=]
```

UNLOCKLINK

This unlocks the relationship between parent and child nodes. The format is:

```
UNLOCKLINK [PARENTASSY=] [CLASSNAME=] [INSTANCE=] [ITEMNAME=]
[ITEMREV=] [SUBASSY=] [OBJECTID=]
```

CLEAR_HIDDEN

This unhides (shows) all hidden nodes.

CLEAR_HIGHLIGHTED

This unhighlights all highlit nodes.

CLEAR_MEMORY

A Product Structure can be held in memory for comparison purposes. This clears the Product Structure in memory.

CLEAR_SELECTED

This unselects all selected nodes.

CLEAR_ALL

This clears all aspects of the tree. This includes selected, highlighted, and hidden nodes and access state information.

CLEAR_ACCESS

This clears access state values that were set by running Access rules.

ACCESS_KEY

This associates a keyname with an Access rule color palette number. The format is:

```
ACCESS_KEY COLOR= KEY=
```

CLEAR_ACCESS_KEY

This clears all Access rule key settings.

IS_TREE_LOADED

This queries whether a tree is loaded. It returns a value of 1 (true) if a tree is loaded or a value of 0 (false) if a tree is not loaded.

IS_LIST_LOADED

This queries whether a list is loaded in the Data Browser window. It returns 1 (true) if a list is loaded or 0 (false) if a list is not loaded.

TREE_STATS

This returns information about the currently loaded Product Structure tree. The format is:

```
TREE_STATS KEYWORD=NODES
TREE_STATS KEYWORD=REFERENCE
TREE_STATS KEYWORD=HIGHLIGHT
TREE_STATS KEYWORD=HIDDEN
TREE_STATS KEYWORD=NAME
TREE_STATS KEYWORD=REVISION
TREE_STATS KEYWORD=TYPE
TREE_STATS KEYWORD=LOCATION
TREE_STATS KEYWORD=DISPLAYED
TREE_STATS KEYWORD=SELECT
```

GET_NODE

This obtains information about the node, identifying the node by its unique node identification number.

Please note: Every node has a unique integer number (NODE=#) when it is displayed in the Product Structure window

The system returns a message for the specified KEYWORD. The format is:

```
GET_NODE NODE=# KEYWORD=ITEM_NAME
GET_NODE NODE=# KEYWORD=ITEM_REV
GET_NODE NODE=# KEYWORD=SELECT
GET_NODE NODE=# KEYWORD=APPLICATION
GET_NODE NODE=# KEYWORD=USERID
GET_NODE NODE=# KEYWORD=NODENAME
GET_NODE NODE=# KEYWORD=CLASS
GET_NODE NODE=# KEYWORD=OWNER
GET_NODE NODE=# KEYWORD=STATUS_CODE
GET_NODE NODE=# KEYWORD=SYSTEM_CODE
GET_NODE NODE=# KEYWORD=SYSTEM_TYPE
GET_NODE NODE=# KEYWORD=USER_TYPE
GET_NODE NODE=# KEYWORD=PART_NUMBER
GET_NODE NODE=# KEYWORD=DESCRIPTION
GET_NODE NODE=# KEYWORD=GTCODE
GET_NODE NODE=# KEYWORD=CREATE_USER
GET_NODE NODE=# KEYWORD=CREATE_DATE
GET_NODE NODE=# KEYWORD=CREATE_TIME
GET_NODE NODE=# KEYWORD=UPDATE_USER
GET_NODE NODE=# KEYWORD=UPDATE_DATE
GET_NODE NODE=# KEYWORD=UPDATE_TIME
GET_NODE NODE=# KEYWORD=SOURCE_DATE
GET_NODE NODE=# KEYWORD=SOURCE_TIME
GET_NODE NODE=# KEYWORD=ACTION_USER
GET_NODE NODE=# KEYWORD=ACTION_DATE
GET_NODE NODE=# KEYWORD=ACTION_TIME
GET_NODE NODE=# KEYWORD=VAULT
```

REFRESH_NODE

This refreshes attribute information for the node from the database. The format is:

```
REFRESH_NODE NODE=#
```

LIST_STATS

This list status information about the Data Browser window. The format is:

```
LIST_STATS KEYWORD=LOCATION 0 - DB, 1 - LOCAL
```

```
LIST_STATS KEYWORD=DIRECTORY
LIST_STATS KEYWORD=LINES
```

GET_LIST

This extracts a column that maps to the keyword from a row in the Data Browser window. The format is:

```
GET_LIST LINE=# KEYWORD=ITEM_NAME
GET_LIST LINE=# KEYWORD=ITEM_REV
GET_LIST LINE=# KEYWORD=SELECT
GET_LIST LINE=# KEYWORD=APPLICATION
GET_LIST LINE=# KEYWORD=USERID
GET_LIST LINE=# KEYWORD=NODENAME
GET_LIST LINE=# KEYWORD=CLASS
GET_LIST LINE=# KEYWORD=OWNER
GET_LIST LINE=# KEYWORD=STATUS_CODE
GET_LIST LINE=# KEYWORD=SYSTEM_CODE
GET_LIST LINE=# KEYWORD=SYSTEM_TYPE
GET_LIST LINE=# KEYWORD=USER_TYPE
GET_LIST LINE=# KEYWORD=PART_NUMBER
GET_LIST LINE=# KEYWORD=DESCRIPTION
GET_LIST LINE=# KEYWORD=GTCODE
GET_LIST LINE=# KEYWORD=CREATE_USER
GET_LIST LINE=# KEYWORD=CREATE_DATE
GET_LIST LINE=# KEYWORD=CREATE_TIME
GET_LIST LINE=# KEYWORD=UPDATE_USER
GET_LIST LINE=# KEYWORD=UPDATE_DATE
GET_LIST LINE=# KEYWORD=UPDATE_TIME
GET_LIST LINE=# KEYWORD=SOURCE_DATE
GET_LIST LINE=# KEYWORD=SOURCE_TIME
GET_LIST LINE=# KEYWORD=ACTION_USER
GET_LIST LINE=# KEYWORD=ACTION_DATE
GET_LIST LINE=# KEYWORD=ACTION_TIME
GET_LIST LINE=# KEYWORD=VAULT
```

SET_LIST

The format is:

```
SET_LIST LNE=# KEYWORD=SELECT VALUE=
```

REFRESH_LIST

This refreshes the data presented in the Data Browser window. The format is:

```
REFRESH_LIST LOCATION=1(local) NAME= [REVISION=] APPLICATION=
DIRLOC=1(create) 2(read) 3(other) [DIRECTORY=]
```

```
REFRESH_LIST LOCATION=2(DB) NAME= [REVISION=] [APPLICATION=]
[CLASS=1(ANY) 2(Public) 3(Private) 4(Project) PROJECT=] [STATUS=]
[SYSTEM_CODE=] [USEDBY=] [USEDNODE=] [USER_TYPE=] [SYSTEM_TYPE=]
[PART_NUMBER=] [DESCRIPTION=] [GTCODE=] [UPDATE_USER=
UPDATE_CHOICE= UPDATE_DATE=] [STORE_USER= STORE_CHOICE=
STORE_DATE=] [ACTION_USER= ACTION_CHOICE= ACTION_DATE=] [ATTRNAME=
ATTRVALUE=]
```

SET_COMPONENT_TEXT

The format is:

```
SET_COMPONENT_TEXT [STRUCTURE=] [COMPONENT=]
```

SET_TOOLBAR

This changes the EPD.Connect toolbar based on the menu file. The format is:

```
SET_TOOLBAR FILE=
```

MESSAGE

This outputs the message to the EPD.Connect message result field. Messages can optionally be included in the EPD.Connect AUDIT log. The format is:

```
MESSAGE TEXT= [LOG=ONLY|BOTH]
```

PUTENV

This enables you to set any environment in the scope of the target process. The format is:

```
PUTENV VARIABLE= VALUE=
```

SIGNON_STATS

This returns information on whether EPD.Connect is signed on to the Vault or not. The format is:

```
SIGNON_STATS [KEYWORD=USERID] [KEYWORD=DATABASE]
```

HELP

This returns legal `COMMAND` names and options for specified commands.

EPD.Visualizer and 3D Viewer Commands

This section lists the commands that can be sent to the 3DViewer and Visualizer target destinations.

SELECT

This adds the given component name to the selection list. The format is:

`SELECT` component_name

DESELECT

This removes the given component name from the selection list. The format is:

`DESELECT` component_name

CLEAR_SELECTED

This removes all component names from the selection list.

HIDE

This hides the named component. The format is:

`HIDE` component_name

HIDE_ALL

This hides all components.

SHOW

This displays the named component. The format is:

`SHOW` component_name

SHOW ALL

This displays all components.

COLOR

This applies a color to the named component. The format is:

`COLOR` value component_name

where,

value = integer value between 0 and 15 inclusive

component_name = name of component to apply color

DECOLOR

This clears all color attributes, including transparency, for the selected component. The format is:

`DECOLOR` component_name

CLEAR_COLOR

This clears all color attributes, including transparency, for all components.

TRANSP

This assigns a transparency value to the named component. The format is:

`TRANSP` value component_name

where,

value = value between 0 and 1 inclusive where 0 = solid and 1 = invisible
component_name =name of component to apply transparency to

POSITION

This sets the position of a component using given x,y,z coordinates for rotation and translation. The format is:

`POSITION x1:y1:z1:x2:y2:z2` component_name

where,

x1: y1: z1 = translate using x, y, z coordinates

x2, y2, z2 = rotate using x, y, z coordinates

component_name = name of component to be rotated

TRANSLATE

This translates the named component using the given x,y,z coordinates. The format is:

`TRANSLATE x1:y1:z1` component_name

where,

x1:y1:z1 = translate using x, y, z coordinates

component_name = name of component to be translated

ROTATE

This rotates the named component using the given x,y,z coordinates. The format is:

`ROTATE x2:y2:z2:x1:y1:z1` component_name

where,

x1:y1:z1 = rotate about the x, y, z coordinates

x2:y2:z2 = use for offsetting rotation

component_name = name of component to be rotated

CAMERA_POSITION

This sets the camera position at the given coordinates. The format is:

```
camera = 1 = use main view
         2 = use top view (For Visualizer ONLY)
         3 = use bottom view (For Visualizer ONLY)
       <= 0 = use main view
        > 3 = use main view
relative = 0 = set camera position point to valueX, valueY, valueZ
           1 = set camera position relative to original camera
position
valueX, valueY, valueZ = x,y,z coordinate
```

CAMERA_TARGET

This sets the camera orbit point at the given coordinates. The format is:

```
CAMERA_TARGET camera:relative:valueX:valueY:valueZ
camera = 1 = use main view
         2 = use top view (For Visualizer ONLY)
         3 = use bottom view (For Visualizer ONLY)
         <= 0 = use main view
```

```
          > 3 = use main view
relative = 0 = set camera orbit point to valueX, valueY, valueZ
           1 = set camera orbit point relative to original camera
target
valueX, valueY, valueZ = x,y,z coordinate
```

CAMERA_MOVE

This moves the camera using the given input values. The format is:

```
CAMERA_MOVE camera:action:value1
camera = 1 = use main view
         2 = use top view (For Visualizer ONLY)
         3 = use bottom view (For Visualizer ONLY)
       <= 0 = use main view
        > 3 = use main view
action = 1 = Move camera forward (zoom)
         2 = Move camera backward
         3 = Move camera left
         4 = Move camera right
         5 = Move camera up
         6 = Move camera down
value1 = distance to move
```

CAMERA_ORBIT

This orbits the camera using the given input value. The format is:

```
CAMERA_ORBIT camera:action:value1
camera = 1 = use main view
         2 = use top view (For Visualizer ONLY)
         3 = use bottom view (For Visualizer ONLY)
       <= 0 = use main view
        > 3 = use main view
action = 1 = Orbit camera left about the vertical axis
         2 = Orbit camera right about the vertical axis
         3 = Orbit camera up about the vertical axis
         4 = Orbit camera down about the vertical axis
         5 = level off to the horizontal model plane
         7 = camera axis counter-clockwise
         8 = camera axis clockwise
value1 = orbit angle (degrees)
```

LOAD_OPTION

This specifies a rendering option for shading. The options are:

LOAD_OPTION flat

This displays a shaded solid view of the 3D scene. The edges of the facets between the polygons are distinguishable.

LOAD_OPTION shell

This displays a smooth-shaded solid view of the 3D scene. For polygons that are defined as shells in the graphic files, shared edges are not distinguishable.

LOAD

This loads a list of commands. The format is:

`LOAD` directory/listname

where,

directory = location of listname and

listname = file that contains the command list

DELETE

This delete a component name. The format is:

`DELETE` value component_name

where,

value = `1` = initialize the session if this is the last component to be removed

value = `0` = don't initialize the session

component_name = name of component to delete

DELETE_ALL

This deletes all the components loaded in Visualizer or the 3DViewer. The format is:

`DELETE_ALL` value

where,

value = `1` = initialize the session if this is the last component to be removed

value = `0` = don't initialize the session

UPDATE_DISPLAY

This updates the graphics window.

SET_NO_INTERUPT

This allows no interrupts during display update.

SET_INTERUPT

This allows interrupts during display update.

Environment Variables

The environment variables exclusive to Vault and AccessWare are provided in this section.

Vault

Variables specific to Vault are provided in the next table.

Table A-1 Vault Environment Variables

Variable	Description	Default Value	
		UNIX	Windows
EDM_HOME	Vault Home Directory	/opt/epd/dm/v60	\EPD\dm\v60
NLSPATH	Native Language Support Directory	$EDM_HOME/data/reposit/$LANG/%N.cat	%EDM_HOME%\data\reposit\%LAN%\%N.cat
DATA_DIRECTORY	Database Directory	/usr/apl/edm/data	

AccessWare

Variables specific to AccessWare are provided in the next table.

Table A-2 AccessWare Environment Variables

Variable	Description	Default Value	
		UNIX	Windows
ACCESSDIR	AccessWare Home Directory	$EPD_HOME	%EPD_HOME%
CMOM_DOMAIN	Application Domain	EPD.Connect	

Index

Q

R

S

U

V

W