Distributed Computing Environment for Windows NT, Version 2.2

DCE Enhancements

Distributed Computing Environment for Windows NT, Version 2.2

DCE Enhancements

Contents

Chapter 1. DCE for Windows NT Enhancements

The DCE for Windows NT product provides the following value-added features to help you run DCE and develop and deploy DCE applications:

- Slim Client (new in this release)
- Public Key Certificate Login Support (new in this release)
- CDS Preferencing (new to this release)
- Tunable Timeout Values for Configuration (new to this release)
- Tunable Timeout Values for RPC (new to this release)
- Multiple Network Interface Cards (new to this release)
- Modifications to Internationalization (enhanced for this release)
- DCEsetup
- DCE Director
- Visual ACL Editor
- Unattended Configuration
- Auto-Start
- Silent Install
- Integrated Login
- Remote DCE Client Configuration
- Inline CDS Clerk
- CDS Cache Restructuring
- Credential Cache Cleanup
- IBM VisualAge C++ Support
- DCE Control Program (dcecp) Extensions
- Application Debugging with the RPC Event Logger
- IDL Compiler Enhancements
- Additional Example Programs
- Enhanced Online Information
- Simple Network Management Protocol (SMNP)
- Commercial Data Masking Facility (CDMF)
- Event Management Services (EMS)
- MVS DCE Load Balancing Client Support
- DTS Time Providers
- Name Service Interface Daemon
- Uninstall Procedure
- Interoperability and Compatibility

DCE for Windows NT is year 2000 compliant. This means that the user and command line interfaces are enabled to accept the year 2000 as input. Calculations are handled in CCYY format, where CC represents the century, and YY represents the year.

Chapter 2. DCE Administration

The DCE Runtime Services for Windows NT option enables DCE cell administrators to perform many cell administration tasks from a Windows NT system. With a few exceptions that are required by the Intel operating environment, the DCE cell administration tools work exactly the same on Windows NT as they are documented for OSF DCE.

This topic describes those exceptions; it also describes the product's administration enhancements that have been added to the standard OSF DCE functionality. Finally, this topic includes general information to consider when performing cell administration tasks.

DCE Administration Tools on Windows NT

Select from the following topics to obtain information on the various DCE control programs, editors, and administration tools that are provided with the DCE for Windows NT product.

Summary of Control Programs, Editors, and Tools

Running an Administration Tool

Getting Help on Administration Tools

Summary of Control Programs, Editors, and Tools

DCE for Windows NT offers the following control programs, editors, and tools to help you modify your DCE environment.

Icon **Tool**

DCEsetup– Used to perform administration tasks related to configuring and managing DCE clients and servers.

DCE Director– Used to manage a DCE environment.

Visual ACL Editor– Used to set the permissions for all security-relevant objects within DCE, including Registry objects and CDS objects

DCE Control Program (dcecp)– Used to manage DCE services.

CDS Control Program (cdscp)– Used to create and maintain Cell Directory Service objects.

RPC Control Program (rpccp)– Used to create and manage RPC entities.

DTS Control Program (dtscp)– Used to manage the DTS server.

ACL Editor (acl_edit)– Used to edit access control lists (ACLs) in a distributed environment.

Registry Editor (rgy_edit)– Used to display and modify elements in the cell's security registry.

Security Adminstration Program (sec_admin)– Used to manage security registry replica.

The DCE control program, **dcecp**, provides a single command line interface for executing all commands available through **cdscp**, **rpccp**, **dtscp, acl_edit, rgy_edit,**

and sec_admin. While the DCE for Windows NT product continues to support these control programs for the short term, it is recommended that you use **dcecp**.

Note: These older control programs support only English. If you are using non-English data, you must use **dcecp**.

Running an Administration Tool

You can run the following standard DCE administration tools from either a Windows NT MS-DOS command window or the DCE for Windows NT v2.2 Program folder.

The DCE control program, **dcecp** is located in the DCE for Windows NT Program folder. You can run all the other listed tools from the Additional Utilities folder. To display the Additional Utilities folder, click the **Start** menu, select **Programs**, and then click the DCE for Windows NT folder.

To run the DCE Control Program, do one of the following:

- From MS-DOS, type: `dcecp`
- From the DCE for Windows NT Program folder, click on

To run the CDS Control Program, do one of the following:

- From MS-DOS, type: `cdscp`
- From the Additional Utilities folder, click on

To run the RPC Control Program, do one of the following:

- From MS-DOS, type: `rpccp`
- From the Additional Utilities folder, click on

To run the Registry Editor, do one of the following:

- From MS-DOS, type: `rgy_edit`
- From Additional Utilities folder, click on

To run the ACL Editor, do one of the following:

- From MS-DOS, type: `acl_edit`
- From the Additional Utilities folder, click on

To run the Security Adminstration Program, do one of the following:

- From MS-DOS, type: `sec_admin`
- From the Additional Utilities folder, click on

Once the appropriate prompt or command window appears, you can enter any of the commands supported by OSF DCE for that tool. However, certain commands require that the server be resident on the system from which those commands are issued. For a list of these commands, see “Managing the Cell from a Windows NT System” on page 11.

For DCE administration command syntax information, refer to the *OSF DCE Command Reference* online documentation.

For general information on how to administer DCE cells, refer to the *OSF DCE Administration Guide* online documentation.

Getting Help on Administration Tools

Online help is available for all control program commands. To display DCE command reference information, double-click the OSF DCE Command Reference help icon in the DCE for Windows NT Documentation folder.

DCE command reference information is also available in the printed manual *OSF DCE Command Reference*. Help is also available for the windows-based tools (DCEsetup, CDS Director, and the Visual ACL Editor).

DCE Commands Available from the Additional Utilities Folder

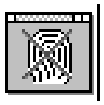
The following commands are available from the Additional Utilities folder. To display the Additional Utilities folder, click the **Start** menu, select **Programs**, select the DCE for Windows NT v2.2,and then click on Additional Utilities.

Icon **Command**

DCE login
Validates a principal's identity and obtains the principal's network credentials.

kinit refreshes the principal ticket-granting ticket.

klist Lists cached tickets.



kdestroy
Destroys a principal's login context and associated credentials.

GUI-Based Administration Tools

DCE for Windows contains the following GUI-based tools to assist you with the administration of you DCE cell.

DCEsetup

DCE Director

Visual ACL Editor

DCEsetup

DCEsetup is a graphical tool for configuring and managing DCE services on a Windows NT system. With DCEsetup, you can configure all of the components on a Windows NT system. The choices you make determine whether your Windows NT system functions as a client or server system, CDS read-only replica server, or Security read-only replica server.

DCEsetup provides a status window that lists all DCE services that have been installed on your system, their configuration status, and the status of the corresponding daemon. It also provides a configuration log window that displays process information as DCEsetup configuration options are performed.

DCEsetup offers these additional capabilities:

Auto-start

Integrated login
Remote client configuration

Using DCEsetup

To run DCEsetup:

Double-click

To get information on using DCEsetup:

Double-click Configuring with DCEsetup in the Documentation folder

Or

Pull down the Help menu in the main window of DCEsetup, select Help Topics and choose Contents. Context-sensitive help is also available for the commands, wizards, and dialog boxes within the tool.

DCE Director

The DCE for Windows NT Runtime Services provides a graphical user interface called DCE Director for managing your DCE environment. DCE Director provides an object-oriented view to the DCE environment. Cell objects consist of users, groups, hosts, CDS directories, and servers, each of which can be created, copied, modified, or deleted.

DCE Director runs on Windows NT. However, it can manage all kinds of DCE hosts because it uses standard DCE protocols.

Using the DCE Director

To run the DCE Director:

Double-click

.

To get information on using the DCE Director:

From the Documentation folder, double-click **Using DCE Director**.

Or

From the DCE Directory window, click the Help menu and then click **Contents**. Context-sensitive help is also available for the commands and dialog boxes within the tool.

Note: DCE Director is based on the function contained in the OSF DCE 1.1 release.

Visual DCE ACL Editor

The Visual DCE ACL Editor makes it easy for you to set the permissions for all security-relevant objects within DCE, including Registry objects and CDS objects.

This tool provides an easy means of managing DCE access control lists (ACLs). It is integrated with DCE Director but can function as a standalone tool.

Using the Visual ACL Editor

To run the Visual ACL Editor:

Double-click

.

To get information on using the Visual ACL Editor:

From the Documentation folder, double-click **Using Visual ACL Editor**.

Or

From the Visual ACL Editor window, click the Help menu, and then click **Contents**. Context-sensitive help is also available for the commands and dialog boxes within the tool.

Note: Visual ACL Editor is based on the function contained in the OSF DCE 1.1 release.

NT Event Logging of DCE Messages

DCE for Windows NT automatically writes fatal and error messages to the Windows NT event log.

A Windows NT enhancement gives administrators the option to edit the **%DCELOC%\dcelocal\var\svc\routing** file and cause select DCE serviceability messages to go to the NT event log as well.

To facilitate this, a new output form, NTLOG, has been added. The text string `"NTLOG:-"` can be inserted in any routing entry in the same way as the STDERR output form. For example:

```
WARNING:STDERR:-;FILE.32.256:e:/opt/dcelocal/var/svc/warning.log
```

becomes...

```
WARNING:NTLOG:-;STDERR:-;FILE.32.256:e:/opt/dcelocal/var/svc/warning.log
```

DCE Credentials on Windows NT

Each time you log into DCE, a fresh set of DCE credentials is obtained. These credentials are available to all of the user's processes in the system. However, the credentials are affected by normal cell and account policies and remain subject to credential expiration. If you remain logged in to Windows NT longer than the credential lifetime, you will need to do one of the following:

- Use **kinit** to refresh existing credentials
- Use **dce_login** to obtain new credentials

Inheriting Machine Credentials from the Administrator Account

When logged into the Windows NT Administrator account, **kdestroy** will operate differently than when logged into other user accounts (even if the account has the administrator privileges). The Administrator account is somewhat analogous to ***root*** on UNIX. As is the case on UNIX, if you have valid DCE credentials and execute **kdestroy**, you will be assigned the local machine credentials (dcecred_ffffff).

On most UNIX DCE implementations, if you are currently assigned the local machine credentials with ***root*** access and perform another **kdestroy** , DCE will deassign the machine credentials and will delete the dcecred_ffffff credential files. This has undesirable side effects for the DCE services running on the system. On DCE for Windows NT, Version 2.2, **kdestroy** will never delete the local machine credentials. A warning message is printed and the request is ignored. If you have Administrator privileges, you will continue to be assigned the local machine credentials.

Be aware that this behavior has the side effect on Windows NT of never allowing unauthenticated RPC access whenever you are logged into the Administrator account. If you have valid DCE credentials, RPC requests will be authenticated. If you **kdestroy** those credentials, you will be assigned the local machine credentials and subsequent RPC requests will continue to be authenticated, both within the cell and across cell boundaries.

If you are logged in as Administrator in an intercell environment, as long as you have valid DCE credentials, the concept of an ″unauthenticated user″ does not exist.

Registering a Cell in X500

DCE for Windows NT allows NT cells to be registered with X500 servers. To do this, follow these steps:

1. On the system where the X500 server is running, create a **dxd_dua.dat** file. Do this with the **dua_configure** utility. The presentation address in the **dxd_dua.dat** file must be in RFC1006 format. To do this, perform these steps:
   a. Also on this system, determine the RFC1006 presentation address string by obtaining the host network address (for example, 11.22.33.44). Use this address as part of the presentation address, as shown in the following address:

```
"DSA"/"DSA"/"DSA"/RFC1006+11.22.33.44,RFC1006
```

b. At your local host, add this address to the dsa's presentation addresses in ncl:

```
ncl> set dsa presentation address  paddr_w_rfc_addr
```

where *paddr_w_rfc_addr* is the presentation address with the RFC address appended to the end. An example of this follows:

```
"DSA/"DSA"/"DSA"/NS+490011AA000021,CLNSRFC1006+11.22.33.44,RFC1006
```

If you now use the **ncl show presentation address** command, the address will appear in a different format.

c. Modify the **dxd_dua.dat** file to include the RFC presentation address. The following is a sample **dxd_dua.dat** file with the RFC presentation address followed by the CLNS address:

```
DUA.KnownDSAs.paddr
="DSA"/"DSA"/"DSA"/NS+006630141054,RFC1006|NS+490004AA000300C71021,CLNS
DUA.KnownDSAs.ae_title = /O=dec/CN=dsa1

#DUA.PreferChaining               =true
#DUA.ChainingProhibited         = false
#DUA.DontUseCopy                 =false
#DUA.DontDereferenceAliases   = false
#DUA.ScopeOfReferral              =DMD
#DUA.TimeLimit                              =.60
#DUA.SizeLimit                               = 30
#DUA.Priority                          = Medium
#DUA.DomainRoot                    = /
#DUA.InitialEntry                      = /
```

2. Copy the **dxd_dua.dat** file to your NT system directory.
3. At your local host, run **X500_addcell.exe** from the command line. The syntax for the call is:

```
X500_addcell -o7 -C cellname -p n
```

where:

-O X500 object class; should always be 7

-C Local NT cell name that is to be registered

-p Either ***n*** or ***y***, depending on whether the user wants to overwrite an existing entry with the same name

Managing the Cell from a Windows NT System

You can use the control programs and tools, provided with the DCE for Windows NT product, to perform many cell administration tasks. However, you must run some commands from the host system for the server that they control.

The following commands must be run on the system where the CDS server is installed:

- create clearinghouse
- delete clearinghouse
- disable server

- show server
- remove clearinghouse

If these commands are run on a system that does not have the CDS server installed, the CDS Control Program returns the following error message:

```
Endpoint not registered
```

Unsupported OSF DCE Function

The differences are grouped into sections by type. Each section is further subdivided into functional categories, which correspond with specific DCE services (such as Configuration, Security, and Cell Directory Services).

Unsupported Services:

- Security:
  - Transitive Trust in a cell hierarchy is not supported in this release.
  - The Public Key Certificate Management API is not supported in this release.
  - The Private Key Storage server is not supported in this release.
  - User-to-User Authentication is not supported in this release.
  - Global Groups are not supported in this release.
- Directory:
  - Hierarchical Cells are not supported in this release.
  - Global Directory Services (GDS) are not provided in this release. However, GDS can exist in the same cell and be used for intercell communications, if it is provided by another product.
- RPC
  - Single-threaded RPC is not supported in this release.

Unsupported Commands:

- Security:

  The **sec_salvage_db**, **rlogin**, **rlogind**, **rsh**, and **rshd** commands supplied by OSF are not supported in this release.
- Distributed Time Service:

  The **dtss-graph** command, which converts synch trace to PostScript, is not supported.

For additional command and API restrictions, see the Internationalization “Unsupported Functions” on page 273, “Enable and Disable Endpoints Restriction” on page 66, and “Unsupported DECthreads Interface Routines” on page 68.

Chapter 3. Installation and Configuration Enhancements

DCE for Windows NT provides improvements in the areas of installation and configuration. These improvements include the following:

Tunable Timeout Values for Configuration

The DCE configuration, start, stop, and unconfiguration programs perform certain tasks that might take some time to complete. These programs wait a specified amount of time for the completion of tasks. Some tasks can take longer on certain machines, causing the program to fail. If these timeouts need to be changed to accommodate a specific machine, you can modify the **usrstime.tcl** file in the **%DCELOC%\dcelocal\etc** directory. This file contains the timeout values for all actions that have timeouts in the configuration, start, stop, and unconfiguration programs.

Note: The **usrstime.tcl** file contains tcl syntax. You must not use variables that have not already been defined. You must also ensure that you do not remove the comment characters (#) from the comment lines or the ″**set**″ from the beginning of the variable lines. It is strongly recommended that you make a backup copy of the **usrstime.tcl** file before modifying it.

All timeouts and intervals are given in seconds. A timeout is the **maximum** time that the code will wait for the specified action to be completed. If the action is completed before the timeout value is reached, the program will continue without waiting the remaining amount of time. An interval is the time waited between checks. For example, if the program is waiting for the CDS namespace, the code loops on a ″**directory list /.:**″. If the timeout is 300 and the interval is 15, the code will run a ″**directory list /.:**″ every 15 seconds for up to 300 seconds until it is completed successfully.

The **usrstime.tcl** file is divided into three sections. The first section contains the default start, stop, and listen timeouts and intervals. These defaults can be used in the specific timeout and interval values by placing a $ before the default text. For example:

```
set default_start_timeout     120
...
set rpc_start_timeout         $default_start_timeout
```

The second section contains the specific start and stop timeouts and intervals for each component. The intervals on the start and stop are how often the code checks the status of the daemon.

```
(comp)_start_timeout   = component start timeout
(comp)_start_init      = component start interval
(comp)_stop_timeout   = component stop timeout
(comp)_stop_init      = component stop interval
```

The components are as follows:

rpc RPC

sec_cl
Security client

sec_svr
Security Master server

sec_rep
Security Replica server

cds_cl
CDS client

cds_svr
Initial Directory server

cds_2nd
Secondary Directory server

dcecm
Integrated Login

dts_local
DTS Local

dts_global
DTS Global

dts_cl DTS client

gda Global Directory Agent

nsid Name Space Interface Daemon

pw_strength
Password Strength server

audit Audit

ems Event Management Service

snmp Simple Network Management Protocol

The third section contains timeouts and intervals for actions other than start and stop. See Table 1 for a listing of the variables, their discriptions, and the error messages that will be displayed if the timeout is reached:

Table 1. Miscellaneous Timeout Variables

Timeout Variable	Description	Error Message
wait_for_cds_root_timeout wait_for_cds_root_int	After starting the CDS client, wait for the CDS advertiser to find the CDS server.	Could not contact the directory server.
wait_for_cds_boot_timeout	This timeout replaces wait_for_cds_root_timeout during autostart.	Could not contact the directory server.

Table 1. Miscellaneous Timeout Variables (continued)

Timeout Variable	Description	Error Message
wait_for_cds_namespace_timeout wait_for_cds_namespace_int	After starting the CDS client, wait for access to the CDS namespace.	Could not access the namespace.
wait_for_acls_timeout wait_for_acls_int	During the CDS initial configuration, wait for setting the clearinghouse ACLs to be successful.	Could not modify the ACLs for: /.:/
wait_for_dced_reg_timeout wait_for_dced_reg_int	After starting the CDS client, wait for a ″test″ server catalog command to work to signify that dced is ready for requests.	Unable to create DCED server objects.
wait_for_dced_reg_boot_timeout	This timeout replaces wait_for_dced_reg_timeout during autostart.	Unable to create DCED server objects.
wait_for_current_dhcp_bindings	Sets the amount of time to wait before attempting to contact dced for the first time in the wait_for_dced_reg_timeout timer loop. This is needed to allow dced time to discard its stale bindings on a DHCP configuration. If you are not running DHCP, this value can be set to zero.	None
wait_for_binding_file_timeout wait_for_binding_file_int	Before starting the security client (secval activate), wait for dced to write out its binding file so that the correct binding information can be gotten from it.	Unable to obtain a binding for the Security client.
wait_for_sec_srv_ready_timeout wait_for_sec_srv_ready_int	After starting the master security server, wait until it is ready to receive commands.	Could not contact the Security Master server.
get_sec_srv_timeout get sec_srv_int	Timeout for autodetecting the master security server.	Unable to determine the Security Master server for the cell: <cellname>
dced_i_stop_timeout dced_i_stop_int	After initializing the dced databases, time to wait for dced to coompletely stop before starting it again with new parpmeters.	Unable to complete creation of DCED databases. DCED -i did not exit.
dir_detete_timeout dir_delete_int	Time to wait for a delete of a CDS directory to be successful.	Deletion of directory <directory_name> failed.

Table 1. Miscellaneous Timeout Variables (continued)

Timeout Variable	Description	Error Message
wait_for_port_135_timeout wait_for_port_135_int	Time to wait for tcp/ip port 135 to be free.	Waited %s minutes for port 135 to clear. DCED could not be started because port 135 is still in use. Try configuration again later.

Example of usrstime.tcl

The following is an example of what a typical **usrstime.tcl** file might look like:

```
###############################################################################
# Default Times to sleep
###############################################################################

# DEFAULT - daemon wait times
  set default_start_timeout 120
  set default_start_int       5

  set default_stop_timeout 120
  set default_stop_int       5

  set default_listen_timeout    120
  set default_listen_int        5
# %Z%%M% %I%  %W%  %G%  %U%

###############################################################################
# Times to sleep
###############################################################################
# Specific Daemon wait times
# RPC
  set rpc_start_timeout $default_start_timeout
  set rpc_start_int $default_start_int
  set rpc_stop_timeout $default_stop_timeout
  set rpc_stop_int $default_stop_int

# SEC_CL
  set sec_cl_start_timeout $default_start_timeout
  set sec_cl_start_int $default_start_int
  set sec_cl_stop_timeout $default_stop_timeout
  set sec_cl_stop_int $default_stop_int

# SEC_SVR
  set sec_svr_start_timeout $default_start_timeout
  set sec_svr_start_int $default_start_int
  set sec_svr_stop_timeout $default_stop_timeout
  set sec_svr_stop_int $default_stop_int

# SEC_REP
  set sec_rep_start_timeout $default_start_timeout
  set sec_rep_start_int $default_start_int
  set sec_rep_stop_timeout $default_stop_timeout
  set sec_rep_stop_int $default_stop_int

# CDS_CL
  set cds_cl_start_timeout $default_start_timeout
  set cds_cl_start_int $default_start_int
  set cds_cl_stop_timeout $default_stop_timeout
  set cds_cl_stop_int $default_stop_int
```

```
# CDS_CVR
  set cds_svr_start_timeout $default_start_timeout
  set cds_svr_start_int $default_start_int
  set cds_svr_stop_timeout $default_stop_timeout
  set cds_svr_stop_int $default_stop_int

# CDS_2ND
  set cds_2nd_start_timeout $default_start_timeout
  set cds_2nd_start_int $default_start_int
  set cds_2nd_stop_timeout $default_stop_timeout
  set cds_2nd_stop_int $default_stop_int

# DTS_LOCAL
  set dts_local_start_timeout 240
  set dts_local_start_int $default_start_int
  set dts_local_stop_timeout $default_stop_timeout
  set dts_local_stop_int $default_stop_int

# DTS_GLOBAL
  set dts_global_start_timeout 240
  set dts_global_start_int $default_start_int
  set dts_global_stop_timeout $default_stop_timeout
  set dts_global_stop_int $default_stop_int

# DTS_CL
  set dts_cl_start_timeout 180
  set dts_cl_start_int       5
  set dts_cl_stop_timeout 120
  set dts_cl_stop_int        5

# AUDIT
  set audit_start_timeout $default_start_timeout
  set audit_start_int $default_start_int
  set audit_stop_timeout $default_stop_timeout
  set audit_stop_int $default_stop_int

# GDA
  set gda_start_timeout $default_start_timeout
  set gda_start_int $default_start_int
  set gda_stop_timeout $default_stop_timeout
  set gda_stop_int $default_stop_int

# EMS
  set ems_start_timeout $default_start_timeout
  set ems_start_int $default_start_int
  set ems_stop_timeout $default_stop_timeout
  set ems_stop_int $default_stop_int

# SNMP
  set snmp_start_timeout $default_start_timeout
  set snmp_start_int $default_start_int
  set snmp_stop_timeout $default_stop_timeout
  set snmp_stop_int $default_stop_int

# PW_STRENGTH_SVR
  set pw_strength_start_timeout $default_start_timeout
  set pw_strength_start_int $default_start_int
  set pw_strength_stop_timeout $default_stop_timeout
  set pw_strength_stop_int $default_stop_int



# NSID
  set nsid_start_timeout        $default_start_timeout
  set nsid_start_int            $default_start_int
  set nsid_stop_timeout         $default_stop_timeout
```

```
  set nsid_stop_int             $default_stop_int

# Misc Timeouts
  set wait_for_cds_root_timeout       3600
  set wait_for_cds_root_boot_timeout  180
  set wait_for_cds_root_int           15

  set wait_for_cds_namespace_timeout  300
  set wait_for_cds_namespace_int      15

  set wait_for_acls_timeout           180
  set wait_for_acls_int               15

  set wait_for_dced_reg_timeout       3600
  set wait_for_dced_reg_boot_timeout  180
  set wait_for_dced_reg_int       15

  set wait_for_current_dhcp_bindings  30

  set wait_for_binding_file_timeout   $default_start_timeout
  set wait_for_binding_file_int       1

  set wait_for_sec_srv_ready_timeout  $default_listen_timeout
  set wait_for_sec_srv_ready_int      $default_listen_int

  set get_sec_srv_timeout       300
  set get_sec_srv_int           10

  set dced_i_stop_timeout       30
  set dced_i_stop_int       5

  set dir_delete_timeout       120
  set dir_delete_int            5

  set wait_for_port_135_timeout       300
  set wait_for_port_135_int           5
```

Tunable Timeout Values for RPC

DCE for Windows NT, Version 2.2 supports the following APIs that allow you to specify through environment variables the amount of time RPC waits to complete a Security server initialization, TCP connection, and a CDS call:

- "RPC_SEC_COM_TIMEOUT"
- "RPC_CN_CONNECTION_TIMEOUT" on page 19
- "RPC_CDS_COM_TIMEOUT" on page 19

RPC_SEC_COM_TIMEOUT

Purpose:

This environment variable resets the communication timeout value for a Security server to initialize.

Synopsis:

set RPC_SEC_COM_TIMEOUT=*value*

Description:

The predefined values that can be used are :

0 = 2 sec	6 = 60 sec
1 = 3 sec	7 = 120 sec
2 = 4 sec	8 = 240 sec

3 = 8 sec	9 = 480 sec
4 = 15 sec	10 = infinite
5 = 30 sec (default)	

Example:

In the following example

```
set RPC_SEC_C)M_TIMEOUT=4
```

resets the timeout value to 15 seconds.

RPC_CN_CONNECTION_TIMEOUT

Purpose:

This environment variable sets the value for the amount of time that RPC will wait for TCP connect handshake.

Synopsis:

set RPC_CN_CONNECT_TIMEOUT=*value*

Description:

The timeout value range should be between 1 second and 100 seconds. If the specified value is not within the renge, the timeout value will not be set.

RPC_CDS_COM_TIMEOUT

Purpose:

This environment variable resets the RPC communication timeout value for a call to CDS.

Synopsis:

set RPC_CDS_COM_TIMEOUT=*value*

Description:

The predefined values that can be used are :

0 = 2 sec	6 = 60 sec
1 = 3 sec	7 = 120 sec
2 = 4 sec	8 = 240 sec
3 = 8 sec	9 = 480 sec
4 = 15 sec	10 = infinite
5 = 30 sec (default)	

Unattended Configuration

DCE for NT supports unattended configuration of DCE servers and full DCE clients. Unattended configuration enables you to create a response file containing all the required input for configuration. Invoking this file from a command line configures a server or a full client without requiring your attention.

A response file is generated every time you do a configuration whether you configure through DCEsetup or the command line using **dcecp**. It is written to the following location.

%DCELOC%\dcelocal\etc\rspfiles\dce.rsp

Once you have generated the response file, you will need to edit it to modify the machine-specific parameters so you can use it through the command line to run unattended configurations on other machines.

Remote DCE Client Configuration

With the DCEsetup Remote Client option, configuration of a full client system into a DCE cell is quick and easy. An added benefit to using this configuration method is enhanced security; you can configure full client systems without being given access to the cell administrator account.

During remote full client configuration, information for DCE full client systems is created and stored centrally, in the DCE CDS namespace, by the cell administrator. You can then access this configuration information to configure their full client systems.

The online help file, *Configuring with DCEsetup,* contains a full description of DCEsetup and its features.

Slim Client

The Slim Client provides the same programming environment to RPC-based applications as the full DCE client product, but requires fewer resources than the full client.

Functionality

The following full client APIs and Commands are supported by the Slim Client without change:

Supported APIs:

pthread_*
rpc_*
dns_*
sec_*

Supported Commands:

acl_edit
cadump
catraverse
cdscp
dcecp
dce_login
getcellname
getipaddr
klist
kinit
kdestroy
rgy_edit
rpccp

The same restrictions that apply to a full DCE client use of these commands also apply to the Slim Client.

Migration

The Slim Client does not provide any utilities for migration. The full DCE does not provide any utilities for migration to the DCE for Windows NT Slim Client. If you want to migrate from a previous release you must

1. Completely unconfigure the current configuration of DCE.
2. Uninstall the base.
3. Install the base you want to migrate to.
4. Configure your system using the newly installed base.

Interoperability

The Slim Client is fully OSF DCE-compliant and maintains interoperability with the following existing IBM DCE products.

- DCE for Windows NT, Version 2.0
- DCE for Windows NT, Version 2.2
- DCE for AIX, Version 2.1
- DCE for AIX, Version 2.2

Functions Not Supported

The following functions and commands are not supported by the Slim Client in this release:

Functions	**Commands**
Define a cached server	cdscp define cached server *servername* dcecp -c cdscache create *servername*
Show a cached server	cdscp show cached server *servername* dcecp -c cdscache show *servername* -server
Show a server	cdscp show server dcecp -c server show *servername*
Create a clearinghouse	cdscp create clearinghouse *clearinghouse_name* dcecp -c clearinghouse create *clearinghouse_name_list*
Show a cached clearinghouse	cdscp show cached clearinghouse *clearinghouse_name* dcecp -c cdscache show *server_name* -clearinghouse
Show a clerk	cdscp show clerk
Clear a clearinghouse	cdscp clear clearinghouse *clearinghouse_name*
Delete a clearinghouse	cdscp delete clearinghouse *clearinghouse_name* dcecp -c clearinghouse delete *clearinghouse_name_list*
Clear a cached server	cdscp delete clearinghouse *servername* dcecp -c cdscache delete *servername*
Disable a server	cdscp disable server dcecp -c clearinghouse disable *servername*

Functions	Commands
Disable a clerk	cdscp disable clerk

Uninstall Procedure

DCE for Windows NT provides an uninstall procedure that removes all Windows NT Registry entries related to the DCE kit, stops all running processes, and then deletes the executable files and directory tree where DCE is installed.

To uninstall DCE for Windows NT, do the following:

1. Double-click the My Computer icon.
2. Double-click the Control Panel icon.
3. Double-click the Add/Remove Programs icon.
4. Click on **DCE for Windows NT V2.2**, and then click **Add/Remove**.

DCE Control Program (dcecp) Extensions

DCE for Windows NT provides command line extensions to the DCE control program (**dcecp**).

The **dcecp** control program offers a common command line interface for managing DCE services. The DCE for Windows NT product provides extensions to dcecp, offering additional functionality.

The extensions consist of the following commands:

- "config.dce" on page 23
- "start.dce" on page 36
- "stop.dce" on page 38
- "show.cfg" on page 40
- "clean_up.dce" on page 42
- "mkreg.dce" on page 43
- "rmreg.dce" on page 45
- "unconfig.dce" on page 46

config.dce

Configures the DCE components.

Format

config.dce
[–admin_pwd *password*]
[–autostart {yes | no}]
[–cds_replica_list *"list_of_cds_servers"*]
[–cds_server *cds_server*]
[–cell_admin *cell_admin_id*]
[–cell_name *cell_name*]
[–certificate_based_login {yes | no}]
[–clean_autostart {yes | no}]
[–clr_house *server_id*]
[–config_type {full | local| admin}]
[–courier_role {courier | noncourier | backup}]
[–dce_hostname *dce_hostname*]
[–group_rsp_path *filename*]
[–host_id *machine_identifier*]
[–kdc_profile *kdc_profile*]
[–kdc_ini_file *kdc_ini_file*]
[–kdc_passphrase *kdc_passphrase*]
[–lan_profile *profile*]
[–max_unix_id *max_UNIX_id*]
[–min_group_id *min_group_id*]
[–min_org_id *min_org_id*]
[–min_principal_id *min_principal_id*]
[–no_pesite_update]
[–nsid_pwd *nsid_password*]
[–pesite_update_time *update_time*]
[–protocol {tcp udp}]
[–proxy]
[–pwdstr_arg *command_line_args*]
[–pwdstr_cmd *server_name*]
[–pwdstr_principal *password_strength_principal_id*]
[–pwdstr_protect_level {*pktinteg | cdmf | pktprivacy*]
[–rsp_file *filename*]
[–sec_master *security_server*]
[–sec_server_name *security_server_name*]
[–sync_clocks {yes|no}]
[–time_server *server_id*]
[usage]
[–?]
[help]
[operations]
components

Note: The command can recognize unique abbreviated option strings. For example, **–adm** is recognized as **–admin_pwd**. Ensure that the abbreviated strings are unique, **–min** would not be recognized because there are three options (**–min_group_id**, **–min_org_id**, and **–min_principal_id**) that begin with that string. **–min_g**, **–min_o**, and **–min_p**, however, would be recognized because they are unique.

Configuring Clients

Note: –sec_master and –cds_ server do not need to be specified. The configuration code will detect the information if the machine is on the same LAN as a Directory server. However, when the machine being configured is on a different subnet, –sec_master and –cds_server must be supplied because DCE cannot detect this information outside its immediate network.

To Admin Configure a Full Client:

config.dce –config_type admin –host_id *machine_identifier* [–dce_hostname *dce_hostname*] [–cell_admin *cell_admin_id*] [–admin_pwd *password*] [–lan_profile *profile*] [–protocol {tcp udp}] [–group_rsp_path *filename*] [–rsp_file *filename*] cds_cl sec_cl dts_cl

To Local Configure a Full Client:

config.dce –config_type local [–cell_name *cell_name*] [–dce_hostname *dce_hostname*] [–sec_master *security_server*] [–cds_server *cds_server*] [–no_pesite_update] [–pesite_update_time *update_time*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–proxy] [–sync_clocks {yes |no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] client_components

To Fully Configure a Full Client:

config.dce –config_type full [–cell_name *cell_name*] [–dce_hostname *dce_hostname*] [–cell_admin *cell_admin_id*] [–cell_admin *cell_admin_id*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–no_pesite_update] [–pesite_update_time *update_time*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–proxy] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] client_components

Note: The default configuration type is full. For further information on the different configuration types, see the *DCE Administration Commands Reference*.

Configuring Servers

To Configure a Master Security Server:

config.dce –cell_name *cell_name* [–sec_server_name *security_server*][–cell_admin *cell_admin_id*] [–admin_pwd *admin_password*] [–min_principal_id *min_principal_id*] [–min_group_id *min_group_id*] [–min_org_id *min_org_id*] [–max_unix_id *max_UNIX_id*] [–no_pesite_update] [–pesite_update_time *update_time*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–certificate_based_login {yes | no}] [–kdc_profile *kdc_profile*] [–kdc_ini_file *kdc_ini_file*] [–kdc_passphrase *kdc_passphrase*] [–group_rsp_path *filename*] [–rsp_file *filename*] sec_srv

To Configure a Security Replica:

config.dce [–sec_server_name *security server*] [–cell_name *cell_name*] [–cell_admin *cell_admin_id*] [–admin_pwd *password*] [–sec_master *security_server*] [–cds_server *cds_server*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–certificate_based_login {yes | no}] [–kdc_profile *kdc_profile*] [–kdc_ini_file *kdc_ini_file*] [–kdc_passphrase *kdc_passphrase*] [–group_rsp_path *filename*] [–rsp_file *filename*] sec_rep

Note: The **config.dce** command deliberately replicates the **/.:/subsys/dce/sec** directory when it configures a secondary CDS server. During the configuration of a Security Replica, entries are

created in this directory but they might not be immediately propagated to the CDS secondary servers. Since these entries are referenced during subsequent pieces of the Security Replica configuration, failures can occur. To prevent this type of failure, stop all **cdsd** daemons that are running on secondary CDS servers before configuring a Security Replica into the cell. After the successful configuration of the security replica, restart the **cdsd** daemons.

To Configure an Initial CDS Server:

config.dce [–cell_name*cell_name*] [–cell_admin *cell_admin_id*] [–admin_pwd *password*] [–sec_master *security_server*] [–autostart {yes | no} [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–group_rsp_path *filename*] [–rsp_file *filename*] cds_srv

To Configure an Additional CDS Server:

config.dce [–cell_name *cell_name*] [–cell_admin *cell_admin id*] [–admin_pwd *password*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–clr_house *server_id*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] cds_second

To Configure a DTS Server:

config.dce [–courier_role {courier | noncourier | backup}] [–cell_name *cell_name*] [–cell_admin *cell_admin_id*] [–admin_pwd *password*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] dts_local | dts_global

To Configure a Global Directory Agent:

config.dce [–cell_name *cell_name*] [–cell_admin *cell_admin id*] [–admin_pwd *password*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] gda_srv

To Configure an Event Management Server:

config.dce [–cell_name *cell_name*] [–cell_admin *cell_admin_id*] [–admin_pwd *password*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] ems_srv

To Configure a Simple Network Management Protocol Agent Server:

config.dce [–cell_admin *cell_admin id*] [–admin_pwd *password*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [snmp_srv

To Configure an Audit Server:

config.dce [–cell_name *cell_name*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–autostart {yes | no}] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] audit

To Configure a Password Strength Server:

config.dce [–cell_name *cell_name*] [–cell_admin *cell_admin_id*] [–admin_pwd *password*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–pwdstr_arg *command_line_args*]

[–pwdstr_cmd *server_name*] [–pwdstr_principal *password strength principal id*] [–pwdstr_protect_level {*pktinteg* | *cdmf* | *pktprivact*}] [–autostart {yes | no}][–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] pw_strength_srv

To Configure a Name Space Interface Daemon:

config.dce [–cell_name *cell_name*] [–cell_admin *cell_admin_id*] [–admin_pwd *password*] [–sec_master *security_server*] [–cds_server *cds_server*] [–lan_profile *profile*] [–autostart {yes | no}]] [–clean_autostart {yes | no}] [–protocol {tcp udp}] [–sync_clocks {yes | no}] [–time_server *server_id*] [–group_rsp_path *filename*] [–rsp_file *filename*] nsid

To Configure an Identity Mapping Server:

The Identity Mapping server must be configured on the same machine as either a Security Master server or a Security Replica server. Use the command to configure the appropriate security server, and add the idms_srv component.

Options

–admin_pwd *password*

Specifies the cell administrator password. Caution should be used with this option because of the security risk it poses by making this password accessible to others.

–autostart {yes | no}

Specifies that the configured components should be started at machine boot.

–cds_replica_list *"list_of_cds_servers"*

A quoted list of CDS server IP host names or addresses.

–cds_server *cds_server*

Specifies the TCP/IP host name or the TCP/IP address of a CDS server. If the local machine is separated from all CDS servers by a router or a gateway that does not pass broadcast packets, a CDS server must be specified using the **–cds_server** option or CDS cannot be configured. This option should be used for all components except rpc, the initial sec_srv, snmp_srv, and the initial cds_srv.

–cell_admin *cell_admin_id*

Specifies the name of the cell administrator account. When configuring the master Security server (the sec_srv component), the **config.dce** command gives this account privileges throughout the cell. Otherwise, the account named must have sufficient privilege to perform configuration tasks within the cell. If the **–cell_admin** option is not specified, the account cell_admin will be assumed. The value for cell_admin is used by all components except rpc, snmp_srv, audit, and dce_unixd.

–cell_name *cell_name*

Specifies the name of the DCE cell into which the machine should be configured. If no **–cell_name** option is specified, the **config.dce** command uses the cell name in the file **%DCELOC%\dcelocal\etc\dce\dce_cf.db**. A value for *cell_name* is required by all components except snmp. The value can either be in the form of */.../cellname* or *cellname*.

–certificate_based_login {yes | no}

Enables or disables certificate based login.

–clean_autostart {yes | no}
Specifies whether to run the clean up script before auto-starting DCE.

–clr_house *server_id*
Specifies an additional CDS server clearinghouse name.

–config_type {full | local | admin}
Allows the cell administrator to split configuration by specifying admin, local, or full configuration of full clients within the DCE cell. The **–config_type** option has three available config_types:

admin Indicates the admin portion of full client configuration. This updates the namespace and security registry with information about the new client.

The admin piece of configuring a full client requires the cell administrator to run the **config.dce** command from a machine within the existing cell. It should not be run from the new client machine. The cell administrator does not need root user authority to run the admin portion of configuration.

Note: When **config.dce** is called with –config_type admin, the **–host_id** option is also required. The **–host_id** option can be in the form of a TCP/IP address or host name (with or without the domain). The **–dce_hostname** flag is optional. If both flags are used and the machine identifier (–host_id) is in the form of a TCP/IP host name, **host** is called to get the IP address.

local Indicates the local portion of full client configuration. This creates necessary files on the local machine and starts the daemons for the new client.

If the admin piece of **config.dce** has not yet been run, the local piece will fail when trying to contact the cell. In addition the user must have root authority on the machine, and does not need to have any authority in the DCE cell.

Note: When **config.dce** is called with the –config_type local, the **–dce_hostname** *dce_hostname* option should be used with the same name that the cell administrator specified during the admin configuration. If the option is not used, the dce_hostname will be presumed to be the same as the name of the machine (including the domain, as returned from a call to the **host** command). If the name is not the same as the dce_hostname the cell administrator used when setting up the client, the configuration will fail.

full Indicates full configuration. This is the default. Full configuration includes both admin and local configuration steps. The DCE cell administrator must have root authority on the local machine being configured into the cell. If the **–config_type** option is not used, a full configuration will be assumed.

–courier_role {courier | noncourier | backup}
Specifies the interaction the server should have with the global servers in the cell when configuring a DTS server (dts_local or dts_global component). The **–courier_role** option must have on of the following values:

courier
Always synchronize with one of the global servers.

backup
Synchronize with one of the global servers if no couriers are available on the local area network (LAN).

noncourier
Synchronize with the local server first, and, if not enough, try one of the global servers.

–dce_hostname *dce_hostname*
Specifies the identifying name within the cell of the machine being configured. This can be the same as the TCP/IP host name, but does not have to be. If the **–dce_hostname** is not used, the *dce_hostname* will default to the long TCP/IP host name (hostname.domain) of the local machine. When **config.dce** is called with –config_type local, the **–dce_hostname** *dce_hostname* option should also be used with the same dce_hostname used by the cell administrator when **config.dce** was called with –config_type admin to configure the client machine. Otherwise, the configuration will fail. If the cell administrator does not use the **–dce_hostname** flag for the admin portion of configuration, the client is not required to use it either.

–group_rsp_path *filename*
Specifies a directory path to use when searching for included response files.

–host_id *machine_identifier*
Specifies the TCP/IP host name or the TCP/IP address of the client machine being admin configured. When **config.dce** is called with –config_type admin, the **–host_id** option must also be used. Admin configuration can be used for a machine whose TCP/IP address is not yet registered with a nameserver. In that situation, use the **–dce_hostname** *dce_hostname* option with the **–host_id** *IP_address* option.

Note: The **–host_id** option can only be used with the **–config_type admin** option.

–kdc_profile *profile*
Specifies the full pathname of the Entrust user's profile.

–kdc_ini_file *kdc_ini_file*
Specifies the full pathname of the Entrust initialization file.

–kdc_passphrase *kdc_passphrase*
The password associated with the Entrust profile for the Security Server.

–lan_profile *profile*
Specifies the name of the LAN profile this machine should use. If the profile does not yet exist, it is created. The default is **/.:/lan–profile**.

–max_unix_id *max_UNIX_id*
Specifies the highest UNIX ID that can be assigned to principals, groups. or organizations by the Security service. The default is 2,147,483,647.

–min_group_id *min_group_id*
Specifies the starting point (minimum UNIX ID) for UNIX IDs automatically generated by the Security service when groups are added with the **rgy_edit** command. The default is 100.

–min_org_id *min_org_id*
Specifies the starting point for UNIX IDs automatically generated by the Security service when organizations are added with the **rgy_edit** command. The default is 100.

–min_princ_id *min_principal_id*
Specifies the starting point (minimum UNIX ID) for UNIX IDs automatically generated by the Security service when principals are added with the **rgy_edit** command. The default is 100.

–no_pesite_update
Specifies not to update the pesite file.

–nsid_pwd
The password used to create the keytable entry for nsid. (This is usually the same as the cell_admin password.)

–pesite_update_time *update_time*
Specifies the pesite file update interval.

–protocol {tcp udp}
Specifies which communication protocols to support. Valid values are: **tcp** and **udp**.

–proxy
Specifies that the CDS full client is to act as a CDS proxy.

–pwdstr_arg *command_line_args*
Specifies one or more command line arguments to be passed to the password strength server. If more than one argument is passed, double-quotes should be used.

–pwdstr_cmd *server_name*
Specifies the name of the password strength server daemon. The default name is **pwd_strengthd**. When creating password strength servers, it is important to remember that the server daemon should have sufficient owner and group permissions to perform its tasks. For example, if the password strength server requires read access to **%DCELOC%\dcelocal\etc\security**, then the userid it runs under may need to belong to the security group.

–pwdstr_principal *password strength principal id*
Specifies a principal id for the password strength server to run under. For a DCE principal ID, the password strength server will use the credentials of the principal.

–pwdstr_protect_level *protection level*
Specifies the protection level for the password strength server. Valid values are:

pktprivacy
(highest level of encryption, contains cdmf)
cdmf
(lowest level of encryption, contains pktinteg)
pktinteg
no encryption

Encryption protection levels are selected as installation options. It is not valid to select pktprivacy if only the cdmf package is installed.

–rsp_file *filename*
Specifies the full path name of a response file to use for configuration.

–sec_master *security_server*
Specifies the host id of the master Security server. You can use the TCP/IP host name or the TCP/IP address of the master Security server for this option. If the server is not specified, an attempt will be made to locate the master Security server using the Cell Directory Services(CDS). If the master Security server cannot be located, it must be specified using the **–sec_master** option or security cannot be configured. The **–sec_master** option is also needed when the **–sec_rep** option is used to configure a Security replica.

–sec_server_name *security_server_name*
Specifies the name to be given to the Security replica. The default name **dce_hostname** will be used if a Security replica is configured without specifying a name with the **–sec_master** option. Each Security replica must have a unique name within the cell. Using the default name helps ensure this uniqueness.

–sync_clocks {yes | no}
Specifies that this machine clock should be synchronized with the clock on a time server already in the cell.

–time_server *server_id*
Specifies the TCP/IP host name or the TCP/IP address of a time server to synchronize clocks with. If not specified, an attempt will be made to locate the DTS server using the Cell Directory Services (CDS). If a DTS server cannot be located, a DTS server must be specified using the **–time_server** option or the clocks can not be synchronized.

usage Displays a help message.

–? Displays a help message.

help Displays a brief description for the passed arguments.

operations
Lists all the options and the components.

components
Specifies the components to be stopped.

The **Client Components** are:

all_cl All clients (cds_cl, dts_cl, rpc, and sec_cl).

client Same as all_cl.

cds_cl
CDS client.

core Single-machine cell components including **cds_srv**, **sec_srv**,**cds_cl**,**sec_cl**, and **rpc**.

dcecm
Integrated Login

dts_cl DTS client. This component and **dts_local** and **dts_global** are mutually exclusive.

rpc RPC daemon.

sec_cl
Security client. This component includes rpc.

The **Server Components** are:

audit Audit daemon

cds_second
Secondary CDS server. This component and **cds_srv** are mutually exclusive.

cds_srv
Initial CDS server for the cell. This component and **cds_second** are mutually exclusive.

core_srv
Single-machine cell components This is equivalent to including **cds_srv**, **sec_srv**,**cds_cl**,**sec_cl**, and **rpc**.

dts_global
DTS global server. This component and **dts_local** and **dts_cl** are mutually exclusive.

dts_local
DTS local server. This component and **dts_global** and **dts_cl** are mutually exclusive.

ems_srv
Event Management server

gda_srv
Global Directory Agent

idms_srv
Identity Mapping server

nsid Name Space Interface Daemon

pw_strength_srv
Password Strength server

sec_srv
Security server

sec_rep
Security replica

snmp_srv
SNMP server

Description

The **config.dce** command configures and starts the specified DCE components. This command also configures and starts any prerequisite client components. The **config.dce** command only configures the core DCE components. Use the **config.dfs** command to configure DFS components.

Note: If you configure the DCE cell using an X.500 style name and you are running DFS, you will not be able to access the local cell DFS file space unless GDS is also configured.

You can configure a machine into a cell in two ways:

full configuration
used by the cell administrator (as root user) to complete all the configuration steps within the cell (updating the CDS namespace and the security registry) and on the local machine (creating files and starting

daemons). Full configuration is specified with the **–config_type full** option. Full configuration is the default. If **–config_type** is not specified, a full configuration is performed.

split configuration
breaks the configuration tasks into two distinct segments, admin and local. Admin configuration is used by the cell administrator from a machine currently configured in the cell to update the CDS namespace and the security registry with necessary information about the client. Local configuration allows the root user of the new client to create files local to the system and to start the DCE client daemons. Split configuration is specified with the **–config_type admin** and **–config_type local** options.

Full configuration must be used for all servers and can be used for clients. Usually the cell administrator does not have administrator access for all the machines that are going to be configured into the cell as client machines. In this situation split configuration is the option to use.

The admin portion must be run before the local portion can be run successfully. When **config.dce** is called with **–config_type admin**, the **–host_id** option is also needed to identify the machine to be configured as a client. The **–host_id** option can be in the form of a TCP/IP address or a TCP/IP host name with or without the domain. The **–dce_hostname** flag is optional if the cell administrator wishes to specify the dce_hostname of the client machine. If the **–host_id** *IP_address* option is used without the **–dce_hostname** option, the dce_hostname will be presumed to be the same as the TCP/IP host name of the machine (including domain as returned from a call to the **host** command).

A cell administrator might want to configure new clients into a cell before actually having the client machines available or before the host name and IP address are registered in the name server. The **config.dce –config_type admin** command, using the **–host_id** *IP_address* **–dce_hostname** *hostname* options will allow the namespace and security registry information to be updated without any calls to the nameserver for a machine not yet registered.

When **config.dce** is called with **–config_type local**, it is important that the client use the same dce_hostname used during the admin configuration. If the dce_hostname is not the same, the configuration will fail.(If the cell administrator did not use the **–dce_hostname** option, it is not necessary to use it for the local configuration.) If the cell name is not provided, a call to **getcellname** will determine if the local machine is already part of a cell. If it is, the assumption is made that additional client components are to be configured on this machine (for example, to add a CDS client to a machine with only a security client). If a cell name is not provided and the host is not already part of a cell, the configuration will fail. When configuring the master Security server (the sec_srv component), **config.dce** will prompt you for the password to be assigned to the initial accounts it creates in the registry database, including that of the cell administrator. When configuring most other components, this command will prompt you for the password of the cell administrator account so it can perform configuration tasks that require DCE authentication. If the environment variable cell_admin_pw is set, **config.dce** uses its value for the cell administrator password without prompting you. This feature can be useful when automating configuration tasks, but should be used sparingly because of the security risk it poses by making this password accessible to others. The cell administrator password should be changed after the tasks are completed and the cell_admin_pw value is unset in order to limit the security risk. If a requested component is already configured, the **config.dce** command reports this

and continues configuring other components. After the command has completed running, the configured components are listed on the screen. If a requested component is already partially configured, use the **unconfig.dce** command to clean it up before using the **config.dce** command to configure that component. To reconfigure a component with different parameters, use the **unconfig.dce** command to remove the existing configuration before running the **config.dce** command to set up the new configuration.

If a machine has a component configured, and additional components are to be configured, you do not have to respecify values for the **–cell_name**, **–sec_master**, **–cds_server**, and **–lan_profile** options. For example, if you have already configured the Security client on a machine, by specifying the name of the cell (**–cell_name**) and the Master Security server (**–sec_master**), you do not need to specify values for **–cell_name** and **–sec_master** again when you configure other DCE components on that machine.

Before configuring a machine into a cell, ensure that the machine clock is within five minutes of the cell master Security server clock. If the machine clock is skewed more than five minutes, the **config.dce** command may report authentication errors, and the configuration may fail. The **–sync_clocks** and the **–time_server** *server_id* options can be used to synchronize the machine clock to the specified time server.

The **–dce_hostname** option is used to specify the dce_hostname for a machine configured into a cell. The dce_hostname is completely independent of the TCP/IP host name of the machine. If the **–dce_hostname** option is not specified, the dce_hostname will default to the TCP/IP host name (including the domain; for example, **jas.austin.ibm.com**). The default clearinghouse for any cds_second servers will be {dce_hostname}_ch. A Security Replica name will also default to the dce_hostname if the **–sec_server_name** option is not used. The recommended usage is to accept the default name.

Only one security server (either a Master Security Server or a Security Replica) can run on a machine. The sec_srv component is used for the Master Security Server and sec_rep is used for the Security Replica. The **config.dce** command will ensure that the security client (**dced**) and, when configuring a Security Replica, the CDS client (**cds_cl**) are running on the machine before starting the security daemon (**secd**). When configuring a Security Replica the **–sec_master** option can be used to locate the Master Security Server.

The **config.dce** command deliberately replicates the **/.:/subsys/dce/sec** directory when it configures a secondary CDS server. During the configuration of a security replica, entries are created in this directory but they might not be propagated immediately to the CDS secondary servers. Since these entries are referenced during subsequent pieces of the Security replica configuration, failures can occur. To prevent this type of failure, stop all **cdsd** daemons that are running on secondary CDS servers before configuring a security replica into the cell. After the successful configuration of the Security replica, restart the **cdsd** daemons.

Examples

When configuring a DCE cell, first configure and start the master Security server:

```
dcecp config.dce -cell_name /.../comp.sci.cell.uw.edu
sec_srv
```

This command establishes the cell name as **/.../comp.sci.cell.uw.edu**, the name specified with the **–cell_name** option.

It creates the master Security server using the default name (cell_admin) for the cell administrator account. It also configures and starts the RPC daemon and a Security client on the same machine as the master Security server.

If you have UNIX systems in a cell with the Master Security server on a Windows NT or a Windows 95 sever, to avoid UNIX ID conflicts in the DCE registry, use the **–min_princ_id**, **–min_group_id**, **–min_org_id**, and **–max_unix_id** options to specify the starting point and maximum values for UNIX IDs assigned to principals. groups. and organizations when configuring the master Security server. The **–dce_hostname** option is used to designate the dce_hostname of the machine.

```
dcecp config.dce -cell_name /.../comp.sci.cell.uw.edu -min_princ_id\
 2000 -min_group_id 2000 -min_org_id 2000 -max_unix_id\
 45000 dce_hostname csadmin sec_srv
```

After configuring and starting the master Security server on a machine with the TCP/IP short hostname of **deptchair**, configure and start the initial CDS

```
dcecp config.dce –cell_name /.../comp.sce.cell.uw.edu -sec_master\
 deptchair cds_srv
```

Because no **–cell_admin** option was specified, this command assumes that the name of the cell administrator account is “cell_admin”. This command also configures and starts the RPC daemon, a Security client, and a CDS client on the same machine as the initial CDS server.

To run the initial Security and CDS servers on the same machine, the previous examples can be combined into one command:

```
dcecp config.dce -cell_name /.../comp.sci.cell.uw.edu -dce_hostname\
 csadmin sec_srv cds_srv
```

To configure another machine as a DTS global courier server (in a different LAN than the initial CDS server, which is a client to all other DCE services, type the following:

```
dcecp config.dce -cell_name /.../comp.sci.cell.uw.edu -dce_hostname\
 timemachine -courier_role courier -sec_master deptchair -cds_server deptchair\
-lan_profile /.:/lan-prof-2 dts_global cds_cl
```

The **–lan_profile** option was used to specify a user-defined LAN profile rather than the default profile.

To specify the admin portion of configuration for a new client in the comp.sci.cell.uw.edu cell (requires cell administrator’s password only) type:

```
dcecp config.dce -config_type admin -host_id 129.35.6.1 all_cl
```

If the TCP/IP hostname of the machine identified with the **–host_id** flag is **jas.austin.ibm.com**, the dce_hostname will default to **jas.austin.ibm.com**. If the lookup at the nameserver fails, the dce_hostname will be **129.35.6.1**.

```
dcecp config.dce -config_type admin -host_id chc cds_cl
```

The dce_hostname will default to **chc.austin.ibm.com**.

```
dcecp config.dce -config_type adm -host_id\
 pal401.pals.austin.ibm.com -dce_hostname mikep all_cl
```

The dce_hostname is **mikep**. Note that it has no relationship to the TCP/IP host name. Admin configuration updates the CDS namespace and security registry with information about the new client being configured. The local piece of configuration must subsequently be completed on the client machine.

To specify the local portion of configuration for a new client (requires root authority only) type:

```
dcecp config.dce -config_type local\
 -cell_name /.../comp.sci.cell.uw.edu -sec_master deptchair\
 [-cds_server deptchair] all_cl
```

The dce_hostname of this client is **jas.austin.ibm.com**, the same as its TCP/IP host name.

```
dcecp config.dce -config_type local -cell_name /.../comp.sci.cell.uw.edu\
 cds_cl
```

If done on an existing security client, it is not necessary to use the **–sec_master** or **–cds_server** options.

```
dcecp config.dce -config_type local -cell_name /.../comp.sci.cell.uw.edu\
-sec_master deptchair -dce_hostname mikep all_cl
```

The dce_hostname entered is the same one the cell administrator used during admin configuration.

Local configuration must be run after the admin configuration has been completed. To specify full configuration of a client into an existing cell (requires root authority and cell administrator password).

```
dcecp config.dce [-config_type full]\
-cell_name /.../comp.sci.cell.uw.edu [-dce_hostname mjs]\
-sec_master deptchair [-cds_server deptchair] _cl
```

If the **dce_hostname** option is not used, the dce_hostname and the client TCP/IP address are determined by **config.dce** through a call to **hostname**.

Related Information

Commands: **unconfig.dce**.

start.dce

Starts the DCE daemons configured on the local machine.

Format

start.dce
[all]
[usage]
[–?]
[help]
[operations]
components

Options

all Starts the configured DCE components on the local machine.

usage Displays a help message.

–? Displays a help message.

help Displays a brief description for the passed arguments.

operations
Lists all the options and the components.

components
Specifies the components to be stopped.

The **Client Components** are:

all All configured components (client and server)

core All configured DCE components (client and server)

all_cl All clients (cds_cl, dts_cl, rpc, and sec_cl)

client Same as all_cl

cds_cl
CDS clerk

dcecm
Integrated login

dts_cl DTS client

rpc RPC daemon (rpcd)

sec_cl
Security client

The **Server Components** are:

all_srv
All servers (cds_second, cds_srv, dts_global, dts_local, gda, sec_srv, ems_srv, pw_strength_srv, sec_rep, snmp_srv)

core_srv
All core servers (rpc, dced, sec_srv,cds_cl, cds_srv)

audit Audit daemon

cds_second
Additional CDS servers

cds_srv
Initial CDS server for the cell

dts_global
DTS global server

dts_local
DTS local server

ems_srv
Event Management server

gda Global Directory Agent

idms_srv
Identiry Mapping server

nsid Name Space Interface Daemon

pw_strength_srv
Password Strength server

sec_srv
Security server

sec_rep
Security replica

snmp_srv
SNMP server

Description

The **start.dce** command starts the currently fully configured component daemons on the local machine.

Related Information

Commands: **config.dce**, **stop.dce**

stop.dce

Stops the DCE daemons configured on the local machine.

Format

stop.dce
[all]
[usage]
[–?]
[help]
[operations]
components

Options

all Stops the DCE components configured on the local machine.

usage Displays a help message.

–? Displays a help message.

help Displays a brief description for the passed arguments.

operations
Lists all the options and the components.

components
Specifies the components to be stopped.

The **Client Components** are:

all All configured components (client and server)

core All configured DCE components (client and server)

all_cl All clients (cds_cl, dts_cl, rpc, and sec_cl)

client Same as all_cl

cds_cl
CDS clerk

dcecm
Integrated login

dts_cl DTS client

rpc RPC daemon

sec_cl
Security client

The **Server Components** are:

all_srv
All servers (cds_second, cds_srv, dts_global, dts_local, gda, sec_srv, ems_srv, pw_strength_srv, sec_rep, snmp_srv)

core_srv
All core servers (rpc, dced, sec_srv,cds_cl, cds_srv)

audit Audit daemon

cds_second
Additional CDS servers

cds_srv
Initial CDS server for the cell

dts_global
DTS global server

dts_local
DTS local server

ems_srv
Event Management server

gda Global Directory Agent

idms_srv
Identiry Mapping server

nsid Name Space Interface Daemon

pw_strength_srv
Password Strength server

sec_srv
Security server

sec_rep
Security replica

snmp_srv
SNMP server

Description

The **stop.dce** command stops the currently fully or partially configured component daemons on the local machine.

Related Information

Commands: **config.dce**, **start.dce**

show.cfg

Displays the DCE components configured on the local machine.

Format

show.cfg
[all]
[dce]
[usage]
[-?]
[help]
[-no_daemon_check]
[operations]

Options

all Lists all the DCE components configured on the local machine.

dce Displays the configured DCE components. This option is the default.

usage Displays a help message.

–? Displays a help message.

help Displays a brief description for the passed arguments.

–no_daemon_check
Specifies that the daemon running states should not be determined or displayed.

operations
Lists all the options and the components.

Description

The **show.cfg** command displays the currently configured components on the local machine.

The valid configuration states are:

Configured
The component was successfully configured.

Partial
The component failed to configure successfully.

The valid running states are:

Running
The daemon is running and listening.

Not Running
The daemon is not currently running.

Available
The component functions are available (there is no daemon).

Not Available
The daemon is running, but is not currently listening.

Unknown
The running state of the component could not be determined.

Examples

The following is an example of a component summary:

```
Component Summary for Host: xxxxxxxx.xxxxx.xxx.xxx
Component                          Configuration State     Running State
Security Master server                 Configured              Running
Security client                        Configured              Running
RPC                                    Configured              Running
Identity Mapping server                  Partial             Not Running
Initial Directory server               Configured              Running
Directory client                       Configured              Running
Password strength server               Configured              Running
     pwd_strengthd
Audit server                           Configured              Running
Integrated Login (dcecm)               Configured             Available

The component summary is complete.
```

Related Information

Commands: **config.dce**, **unconfig.dce**.

clean_up.dce

Cleans the DCE databases, sockets, and cache files, creates backup log files, and removes DCE-generated core files.

Format

clean_up.dce
[–core]
[–truncate_log]
[usage]
[–?]
[help]
[operations]

Options

–core Specifies that DCE-generated core files are to be removed.

–truncate_log
Specifies that backup DCE-generated log files should be created.

usage Displays a help message.

–? Displays a help message.

help Displays a brief description for the passed arguments.

operations
Lists all the options and the components.

Description

The **clean_up.dce** command cleans DCE databases, sockets, and cache files, creates backup log files, and removes DCE-generated core files. If DCE problems are encountered, the **clean_up.dce** command can be used to remove possibly corrupted files. All of the files that are removed will be recreated.

Without any options, the **clean_up.dce** command removes DCE databases, cache files, and socket files.

With the **–core** option, **clean.dce** removes core files.

When the **–truncate_log** option is used, backup files for the DCE serviceability log files are created in **%DCELOC%\dcelocal\var\svc**.

Related Information

None.

mkreg.dce

Adds information about a DCE cell into the domain namespace.

Format

mkreg.dce
[–input_file *input_file*]
[–named_data_file *named_data_file*]
[usage]
[–?]
[help]
[operations]

Options

–input_file *input_file*
Specifies the name of a file containing information about the cell you want to register. The default is **%DCELOC%\dcelocal\etc\input.file**.

–named_data_file *named_data_file*
Specifies the name of the file that contains data for the Microsoft DNS server, when registering a DNS-style cell name. The default is **%DCELOC%\dcelocal\etc\named.data**.

usage Displays a help message.

–? Displays a help message.

help Displays a brief description for the passed arguments.

operations
Lists all the options and the components.

Description

The **mkreg.dce** command enters information about your DCE cell into the database maintained by your domain Microsoft DNS server.

This command cannot be used to register X.500–style cell names.

If the Microsoft DNS server machine is a member of the DCE cell you want to register, **mkreg.dec** will update the **named** data file you specify with the **–named_data_file** option and if the Microsoft DNS server is running, refresh it.

If the name server machine is not part of the DCE cell you want to register or is not configured with DCE at all, do one of the following:

- Generate the **mkreg.dce** input for the name server. On a machine that is part of the DCE cell you want to register, run the following two commands:

  ```
  cdscp show cell /.: as dns>input.file
  cdscp show clearinghouse/.:/*>>input.file
  ```

  Take the resulting file to your DOMAIN name server and run **mkreg.dce**, using the **–input_file** option to specify the name of the input file.
- Generate the **mkreg.dce** output to add to the named data file on the name server.

It is necessary to create a temporary data file on a machine within the cell, with the information to append to the permanent data file on the nameserver. To do this, run the following two commands to create the file and add the relevant cell data to it:

```
dcecp mkreg.dce –named_data_file output.file
```

Take the resulting output.file to your DOMAIN name server, add the contents of this file to the Microsoft DNS server data file, and have the server read the new data.

Note: When configuring with a DCE host name, be sure to add the DCE host name and the proper IP address of the machine associated with the DCE host name to the list of host names. Remember that the DCE host name is case sensitive.

For example, if you configure a cell with the cell name **/.:/hulacell.austin.ibm.com** on the machine named **mustang1** and set the DCE host name to be **hula.austin.ibm.com**, the following entry needs to be added to the Microsoft DNS server data file on the DNS name server so that the machine name, or in this case the DCE host name, can be resolved to a TCP/IP address.

```
cdsaix1.austin.ibm.com IN A 129.35.66.4
mustang1.austin.ibm.com IN A 129.35.69.52
hula.austin.ibm.com IN A 129.35.69.52
```

Examples

To register a cell when the name server is configured as a CDS client of the cell, type:

```
dcecp mkreg.dce
```

To register a cell when the information about the cell and its CDS clearinghouse is contained in the file **/tmp/cell.info**, type:

```
dcecp mkreg.dce -input_file /tmp/cell.info
```

Related Information

Commands: **rmreg.dce**.

rmreg.dce

Removes information about a DCE cell into the domain namespace.

Format

rmreg.dce
[–dns_cell_name *dns_cell_name*]
[–named_data_file *named_data_file*]
[usage]
[–?]
[help]
[operations]

Options

–dns_cell_name *dns_cell_name*
Specifies the cell name to be unregistered. If no **–dns_cell_name** option is specified, the **rmreg.dce** command uses the cell name in the **%DCELOC%\dcelocal\etc\dce\dce_cf.db** file.

–named_data_file *named_data_file*
Specifies the name of the file on the domain name server that contains the data for the Microsoft DNS server. The default is **%DCELOC%\dcelocal\etc\named.data**.

usage Displays a help message.

? Displays a help message.

help Displays a brief description for the passed arguments.

operations
Lists all the options and the components.

Description

The **rmreg.dce** command removes entries from the database maintained by your domain Microsoft DNS server that were added by the **mkreg.dce** command.

This command cannot be used to register X.500–style cell names.

This command must be run on the name server with which the cell is registered. Use the **–named_data_file** option to specify the name of the data file used by the Microsoft DNS server. The cell information is removed from the specified file.

If the primary name server machine is not part of the DCE cell, the **–dns_cell_name** option must be used.

Examples

To unregister a cell named **/.../comp.sci.cell**, type:

```
dcecp rmreg.dce –dns_cell_name /.../comp.sci.cell
```

Related Information

Commands: **mkreg.dce**.

unconfig.dce

Removes configuration of the DCE components.

Format

unconfig.dce
[–admin_pwd *password*]
[–cell_admin *cell_admin_id*]
[–config_type {full | local | admin}]
[–dce_hostname *dce_hostname*]
[–dependents]
[–force]
[–group_rsp_path *filename*]
[–host_id *machine_identifier*]
[–pwdstr_principal *password_strength_principal_id*]
[–rsp_file *filename*]
[all]
[usage]
[–?]
[help]
[operations] components

Note: The **unconfig.dce** command can be used to unconfigure a full DCE Client. A Slim Client uses a separate tool for unconfiguration.

To Remove Admin Configuration:

unconfig.dce –config_type admin –dce_hostname *dce_hostname* [–cell_admin *cell_admin id*] [–host_id *machine_identifier*] [–dependents] [–force] [–pwdstr_principal *password_strength_principal_id*]components

To Remove Local Configuration:

unconfig.dce –config_type local [–dependents] [–force] [–pwdstr_principal *password_strength_principal_id*] components

To Remove Full Configuration:

unconfig.dce –config_type full [–cell_admin *cell_admin_id*] [–dependents] [–force] [–pwdstr_principal *password_strength_principal_id*] components

Options

–admin_pwd *password*

Specifies the cell administrator password. Caution should be used with this option because of the security risk it poses by making this password accessible to others.

–cell_admin *cell_admin_id*

Specifies the name of the cell administrator account. If the **–cell_admin** option is not specified, the account cell_admin will be assumed.

–config_type {full | local | admin}

Used to specify what type of unconfiguration is to be done. The **–config_type** option has three available unconfig_types:

admin

Specifies that the admin portion of unconfiguration will be completed for the dce_host indicated by the **–dce_hostname** flag. This cleans up the CDS namespace and security registry. The user must have cell administrator authority within the cell.

local Specifies that the local portion of unconfiguration will be completed for the local machine. This stops the daemons and removes the appropriate files. The user must have root authority on the local machine.

Local unconfiguration must be selected when unconfiguring a CDS server whose clearinghouse contains a master replica of a directory.

full Specifies full unconfiguration on the local machine. This is the default unconfig_type. When doing a full unconfiguration on the local host, the user must be the DCE cell administrator and have administrator authority on the local machine. Full unconfiguration is the equivalent of admin unconfiguration and local unconfiguration combined. If the **–unconfig_type** option is not used, a full unconfiguration will be assumed.

–dce_hostname *dce_hostname*
Used with the **–config_type** option to identify the dce_host to unconfigure. Use **–dce_hostname** only when doing the admin portion of unconfiguration.

–dependents
Unconfigures dependent components. Specifies that any components that depend on those listed on the command line should also be unconfigured. For example, on a machine with sec_cl, cds_cl, and rpc, **unconfig.dce –dependents sec_cl** will also unconfigure the cds_cl.

–force Forces unconfiguration of components named on the command line, even if other components depend on their presence. Use this option in clean-up situations. Use this option with extreme caution because the cell can be put into an unstable state.

–group_rsp_path
Specifies the directory path for searching included response files.

–host_id *machine_identifier*
Specifies the TCP/IP host name or the TCP/IP address of the client machine being admin unconfigured. When **unconfig.dce** is called with –config_type admin, the **–host_id** option must also be used. Admin unconfiguration can be used for a machine whose TCP/IP address is not yet registered with a nameserver. In that situation, use the **–dce_hostname** *dce_hostname* option with the **–host_id** *IP_address* option.

Note: The **–host_id** option can only be used with the **–config_type admin** option.

–pwdstr_principal *password strength principal id*
Specifies a principal id for the password strength server. Since more than one password strength server can be configured, the principal id is used to identify a specific server.

all Unconfigures all configured components on the local machine.

–rsp_file *filename*
Specifies the full path name of a response file.

usage Displays a help message.

–? Displays a help message.

help Displays a brief description for the passed arguments.

operations
Lists all the options and the components.

components

Specifies the components to be unconfigured.

The **Client Components** are:

all All configured components (client and server).

all_cl All clients (cds_cl, dts_cl, rpc, and sec_cl)

client Same as all_cl

cds_cl
CDS client

dcecm
Integrated login

dts_cl DTS client. This component and **dts_local** and **dts_global** are mutually exclusive.

rpc RPC daemon

sec_cl
Security client. This client includes rpc.

The **Server Components** are:

all_srv
All servers (cds_second,cds_srv, dts_global, dts_local, gda, sec_srv, ems_srv, pw_strength_srv, sec_rep, snmp_srv)

audit Audit daemon

cds_second
Secondary CDS server. This component and **cds_srv** are mutually exclusive.

cds_srv
Initial CDS server for the cell. This component and **cds_second** are mutually exclusive.

core_srv
Single-machine cell components This is equivalent to including **cds_srv**, **sec_srv**, **cds_cl**, **sec_cl**, and **rpc**.

dts_global
DTS global server. This component and **dts_local** and **dts_cl** are mutually exclusive.

dts_local
DTS local server. This component and **dts_global** and **dts_cl** are mutually exclusive.

ems_srv
Event Management server

gda Global Directory Agent

idms_srv
Identity Mapping server

nsid Name Space Interface Daemon

pw_strength_srv
Password Strength server

sec_srv
Security server

sec_rep
Security replica

snmp_srv
SNMP server

Description

The **unconfig.dce** command stops the specified components and removes their configuration and database files. The **unconfig.dce** command unconfigures only the core DCE components. Use the **unconfig.dfs** command to configure DFS components. If you are removing all DCE and DFS components, use the **unconfig.dfs** command before you use the **unconfig.dce** command.

You can unconfigure a machine from a cell in two ways:

full configuration
used by the cell administrator (as root user) to complete all the necessary steps within the cell (updating the CDS namespace and the security registry) and on the local machine (stopping daemons and deleting files). Full unconfiguration is specified with the **–config_type full** option. The **unconfig.dce** command also defaults to full unconfiguration if the **–config_type** option is not used.

If the cell administrator does not have root user access to the machine that is going to be unconfigured split configuration is the option to use.

Note: If you unconfigure the initial CDS server (with the master copy of the**/.:/directory**) or master Security server in a cell, you will have to unconfigure and reconfigure the entire cell.

split configuration
breaks the unconfiguration tasks into two distinct segments, admin and local. Admin unconfiguration is used by the cell administrator on any machine within the cell to update the CDS namespace and the security registry about changes in the cell. Local configuration is used by the root user on the machine being unconfigured to stop the daemons and delete the appropriate files. Split unconfiguration is specified with the **–config_type admin** and **–config_type local** options.

If the cell for which a machine is configured is inaccessible and you need to unconfigure the machine for any reason, use the **–config_type local** option. This option limits the **unconfig.dce** command to remove only the local pieces of a DCE configuration; it does not remove entries from the namespace or registry database. To remove the entries from the namespace and registry database, the cell administrator should use the **–config_type admin –dce_hostname** option from a machine within the cell.

If the environment variable *cell_admin_pw* is set, **unconfig.dce** uses its value for the cell administrator password without prompting you. This feature can be useful when automating unconfiguration tasks. Be aware, however, that this use of this feature should be limited because of the security risk it poses by making this password accessible to others. The cell administrator password should be changed after the tasks are completed and the *cell_admin_pw* value is unset in order to limit the security risk.

While the **config.dce** command automatically configures any client components required by the specified components, the **unconfig.dce** command will fail if configured components depend on the presence of those requested to be unconfigured. To unconfigure exactly those components specified on the command line, use the **–force** option. This option should be used with caution. In some cases, unconfiguring one component will disable other components that are dependent upon it.

To unconfigure those components specified on the command line and all components that depend on them, use the **–dependents** option. When a cds server (either **cds_srv** or **cds_second**) is requested to be unconfigured, **unconfig.dce** checks for several conditions. If a full configuration is specified, **unconfig.dce** checks to ensure that none of the clearinghouses on the server machine contain a master replica of any directory. If they do not, the unconfiguration continues. If they do, configuration exits with a message explaining what must be done before the server can be unconfigured. If it is necessary to unconfigure a CDS server with a clearinghouse that contains a master replica of a directory, **unconfig.dce** can be run using the **–config_type local** option. After that, **unconfig.dce –config_type admin** should be run from a machine within the cell.

Once a cds server has been unconfigured, it might not be possible to reconfigure it using the same **dce_hostname** until all updates have taken place in the cell. Either use a different **dce_hostname** or wait overnight before reconfiguring.

If you unconfigure the Master Security server in a cell, you will have to unconfigure and reconfigure the entire cell.

Security Replication adds more opportunities for the **pe_site** file to change; therefore, the **pe_site** file will be updated whenever **unconfig.dce** is run (if the **cdsadv** daemon is running).

Examples

To remove the DTS clerk configuration from a machine when the cell administrator account name is **ca**, type:

```
dcecp unconfig.dce –cell_admin ca dts_cl
```

To remove all DCE configuration files and databases from a machine, the administrator types:

```
dcecp unconfig.dce –config_type local all
```

The **unconfig.dce –config_type local** option limits the **unconfig.dce** command to removing only the local pieces of a DCE configuration; it does not remove entries from the namespace or registry database. Use this command when unconfiguring the last machine in a DCE cell or when removing a CDS server whose clearinghouse contains a master replica of any directory.

To specify the admin portion of unconfiguration for a client in an existing cell, the cell administrator types:

```
dcecp unconfig.dce –config_type admin –dce_hostname chc all_cl
dcecp unconfig.dce –config_type admin –dce_hostname jas.austin.ibm.com\
cds_second cds_cl sec_cl dts_cl
```

To specify the local portion of unconfiguration for the local machine, the administrator (with no DCE authority required) types:

```
dcecp unconfig.dce –config_type local all_cl
dcecp unconfig.dce –config_type local –dependents sec_srv
```

To perform full unconfiguration on a client in an existing cell, the cell administrator with administrator authority types:

```
dcecp unconfig.dce
all_cl
dcecp unconfig.dce –config_type full all_cl
```

Related Information

Commands: **config.dce**.

Silent Install

Silent Install enables automated electronic software distribution. Silent Install eliminates the need for the users to monitor their installation process and the need to provide input.

For more information, see:

- Starting Silent Install
- The SETUP.ISS Response File
- The Server CD Response File
- The Client CD Response File
- The SETUP.LOG File

Starting Silent Install

Note: Silent Install can be used with a full DCE Server or a full DCE Client. It does not support the DCE Slim Client.

To start silent install , run SETUP.EXE with the *-s* option. For example, from a DOS command prompt type:

```
setup -s
```

You can also use the *-f1* and *-f2* options to change the name and location of the response file.

For example, from the DOS command prompt type:

```
setup -s -f1c:\mydir\mydir.iss -f2c:\mydir\mydir.log
```

The previous example starts Silent Install, uses the MYDIR.ISS file from the C:\MYDIR directory, and then generates the MYDIR.LOG log file in the same directory.

If you do not use the **-f1** option when running Silent Install, setup looks for the response file SETUP.ISS in the same directory as SETUP.EXE.

When Silent Install runs, a log file is created in the same directory as the response file. The log file has the the default name of SETUP.LOG if the **-f2** switch is not provided along with the**-f1** switch.

The SETUP.ISS Response File

DCE Runtime Services provide a default SETUP.ISS response file.

You can also create your own. You can use the **-r** option in order to select installation options and to automatically record the Silent Install response file.

When you are recording the InstallShield response file, make sure that all prerequisites (for example, Microsoft Windows NT Service Packs, TCP/IP protocol, and NETBIOS Service) have been met.

DCE for Windows NT provides a default SETUP.ISS response file.

The Server CD Response File

The server response file assumes that:

- The system has Microsoft Windows 4.0 ServicePack 3 installed.
- The system has NetBIOS service and TCP/IP Protocol installed.
- If there is an existing DCE, Silent Install defaults to **Preserve the DCE Configuration**. It will not prompt you to choose the **Preserve DCE Configuration**, or **Delete DCE Configuration**.
- The components you can install include:
  - DCE Runtime Services
  - DCE Application Development Kit (ADK)
  - DCE Cell Directory Services (CDS)
  - DCE Security Services (SS)
  - Event Management Services (EMS)
  - Simple Network Management Protocol (SNMP)
- The default install directory is C:\Program Files\DCE.
- The default preferred cultural convention is ENUS437 English in US OEM CP.
- The response is ″No″ to display the readme file.
- The response is ″No″ to restart the system.

The following is an example server response file:

```
[InstallShield Silent]
Version=v3.00.000
File=Response File
[D1gOrder]
Dig0=SdWelcome-0
Count=6
D1g1=SdComponentDialog-0
D1g2=SdAskOptionsList-0
D1g3=SdStartCopy-0
D1g4=AskYesNo-0
D1g5=SdFinishReboot-0
[SdWelcome-0]
Result=1
[SdComponentDialog-0]
szDir=C:\Program Files\DCE
Component-type=string
Component-count=6
Component-0=DCE Runtime Services
Component-1=DCE Application Development Kit (ADK)
Component-2=DCE Cell Directory Services (CDS)
Component-3=DCE Security Services (SS)
Component-4=Event Management Services (EMS)
Component-5=Simple Network Management Protocol (SNMP)
Result=1
[SdAskOptionsList-0]
Component-type=string
Component-count=1
Component-0=ENUS437      English in US OEM CP
Result=1
[SdStartCopy-0]
Result=1
[Application]
Name=Dce for Windows NT
Version=2.2
Company=IBM
[AskYesNo-0]
```

```
Result=0
[SdFinishReboot-0]
Result=1
BootOption=0
```

The Client CD Response File

The client response file assumes that:

- The system has Microsoft Windows 4.0 ServicePack 3 installed.
- The system has NetBIOS service and TCP/IP Protocol installed.
- If there is an existing DCE, Silent Install defaults to **Preserve DCE Configuration**. It will not prompt you to choose the **Preserve DCE Configuration**, or **Delete DCE Configuration**.
- The components you can install include:
  - DCE Runtime Services
  - DCE Application Development Kit (ADK)
  - Event Management Services (EMS)
- The default install directory is C:\Program Files\DCE
- The default preferred cultural convention is ENUS437 English in US OEM CP
- The response is ″No″ to read the release note.
- The response is ″No″ to reboot the system.

The following is an example client response file:

```
[InstallShield Silent]
Version=v3.00.000
File=Response File
[D1gOrder]
Dig0=SdWelcome-0
Count=6
D1g1=SdComponentDialog-0
D1g2=SdAskOptionsList-0
D1g3=SdStartCopy-0
D1g4=AskYesNo-0
D1g5=SdFinishReboot-0
[SdWelcome-0]
Result=1
[SdComponentDialog-0]
szDir=C:\Program Files\DCE
Component-type=string
Component-count=3
Component-0=DCE Runtime Services
Component-1=DCE Application Development Kit (ADK)
Component-2=Event Management Services (EMS)
Result=1
[SdAskOptionsList-0]
Component-type=string
Component-count=1
Component-0=ENUS437      English in US OEM CP
Result=1
[SdStartCopy-0]
Result=1
[Application]
Name-DCE for Windows NT
Version=2.2
Company=IBM
[AskYesNo-0]
Result=0
[SdFinishReboot-0]
Result=1
Bootoption=0
```

The SETUP.LOG File

When you run an installation in silent mode, none of the messages are displayed on the screen. Instead, the SETUP.LOG file captures the installation information. You must view the SETUP.LOG file to determine if the installation was successful.

The SETUP.LOG file contains the following three sections:

- **[InstallShield Silent]** identifies the version of Silent Install that was used in the silent installation. It also identifies the file as a log file.
- **[Application]** identifies the name of the installed application, the version, and the name of the company.
- **[ResponseResult]** contains the return code indicating if the silent installation was successful. One of the following integer value is assigned to the *ResultCode* keyname:

  0 Success

  —1 General error

  —3 Required data not found in the **SETUP.ISS** file

  —4 Not enough memory available

  —5 File does not exist

  —6 Cannot write to the response file

  —9 Not a valid list type (string or number)

  —10 Data type is invalid

  —11 Unknown error during setup

  —12 Dialogs are out of order

SETUP.LOG is the default name for the silent install log file and it is located in the same directory where the .INS file resides. To specify a different name and location for the SETUP.LOG file, run SETUP.EXE using the *-f1*and *-f2* options.

For example, from the DOS command prompt type:

```
setup -s -f1c:\mydir\mydir.iss -f2c:\mydir\mydir.log
```

The above example, starts Silent Install, uses the MYDIR.ISS file from the C:\MYDIR directory, and then generates the MYDIR.LOG log file in the same directory.

When an installation is successful, the SETUP.LOG file contains the following:

```
[InstallShield Silent]
Version=v3.00.000
File=Log File
[Application]
Name=DCE for Windows NT
Version=2.2
Company=IBM
[ResponseResult]
ResultCode=0
```

Chapter 4. Application Development Notes and Considerations

The DCE for Windows NT Application Development Kit option provides universal command interfaces, as well as directory structures, filenames, and application development environments resembling those available from DCE implementations on many UNIX systems. In general, this allows users to read any standard OSF DCE documentation, such as that provided with this release, and to create DCE applications on Windows NT systems.

The DCE for Windows NT product provides extensions and enhancements to the standard OSF DCE services. The following topics describe these extensions and enhancements and provide general information to consider during application development. Differences in writing, compiling, and linking applications between UNIX-based implementations and Windows NT are also described.

The following topics also describe application development formats and rules on Windows NT systems that may differ from those described in the *OSF DCE Application Development Guide*.

Building Applications

Although the DCE for Windows NT product is designed to minimize differences from DCE as it is installed on UNIX systems, there are reasons to conform to Windows NT standards and conventions first.

Primarily, users encounter the differences between the Windows NT and UNIX platforms when they compile and link programs. However, running compiled programs can require setup procedures specific to Windows NT or this DCE kit.

This topic describes command formats and considerations for compiling and linking applications on DCE for Windows NT. For general information about compiling and linking DCE applications, refer also to the *OSF DCE Application Development Guide*.

For more information, see:

- Including Files
- Including the DCE RPC Header File
- Including POSIX Threads Header File
- Using Compiler and Linker Flags with Visual C++
- Linking DCE Applications
- Structure Alignment with C Compilers Restriction
- MFC Classes in IDL Files Restriction
- TZ Environment Variables with DTS Routines Restriction
- Structure of passwd

Including Files

If the DCE for Windows NT ADK has been installed, the DCE installation process sets up your include environment variable so that your compiler can find the required DCE include files in a path such as this:

%DCELOC%\dcelocal\include

Where *%DCELOC%* is the environment variable pointing to the root where DCE is installed.

Many DCE include files are located in the following folder:

%DCELOC%\dcelocal\include\dce

Note: Users must be sure to reboot their system after installing the ADK in order to have the environment variable set properly.

Including the DCE RPC Header File

Applications should include files from this directory by specifying the path **dce/** in the file specification of the # include statement in the application source code. When you include the RPC header file, **rpc.h**, in a DCE application, do not specify simply <rpc.h>. Instead, include the file as shown in the following example:

#include <dce/rpc.h>

This prevents the accidental inclusion of the Microsoft RPC header file, **rpc.h**, when the DCE RPC header file is needed.

Including POSIX Threads Header File

Every module of a DCE application program should include the file **pthread.h**, as shown in the following example:

```
#include <dce/pthread.h>
```

Include this file before any other file included by the module that could possibly contain system or library calls for which there might be thread jacket routines (such as **stdio.h**). In general, it is best to put **pthread.h** first in the include list for each module. If this file does not precede other files in the include list that contains call declarations for which there are jacket routines, the compiler might generate warning or error messages about the prototypes for these routines.

Using Compiler and Linker Flags with Visual C++

The following example shows the compiler flags that should be used when using Visual C++ to create an application.

Programs that call serviceability functions must also use the **/MD** option.

The **cvarsdll**, **/debug**, and other symbols assume you are compiling or linking the programs from inside a makefile. The **makefile** must include the MS makefile called **win32.mak**. The following example assumes that all application header files are in the same directory as the source. If this is not the case, the **-I** option will need to include the other directories as well.

```
cl -I . $(cvarsdll) -DM_I86 -WINNT -Od -Gz
```

where:

-Od disables optimization

-Gz indicates _**stdcall** convention

The following example shows the compiler flags for a debuggable application:

```
cl -I . $(cvarsdll) -DM_I86 -WINNT -Zi -Od -Gz
```

where:

-Zi indicates debugging information should be generated

The following example shows a link line for a GUI application. (A console application would use **$(conlflags)**.)

```
link $(ldebug) $(guilflags) -machine:$(CPU)
```

For information on using the VisualAge C++ compiler, see "Using IBM VisualAge C++" on page 62.

Linking DCE Applications

When linking a DCE application, you must link with the following libraries:

- **libdce.lib**
- **pthreads.lib**

The following command format is an example of how to link with shared libraries:

```
C:\>  link /out:myprg.exe -subsystem:console msvcrt.lib libdce.lib pthreads.lib-entry:mainCRTSta
```

For general information on compiling and linking DCE applications, see the *OSF DCE Application Development Guide* and the documentation for your Windows NT application development environment.

Structure Alignment with C Compilers Restriction

On Windows NT systems, DCE stub and library code assumes the native, nonaligned form for structures. Do not use the C preprocessor pragma to enable structure member alignment (the **-Zp** switch when using the Microsoft C compiler) in your DCE applications for Windows NT as this kit does not support user applications built with the **Zp** option.

MFC Classes in IDL Files Restriction

Due to changes in Visual C++ 4.x and 5.0 include files, DCE for Windows NT does not support use of MFC classes in IDL files. Therefore, compilation of IDL files that include MFC classes will fail.

TZ Environment Variables with DTS Routines Restrictions

Some of the Distributed Time Service (DTS) routines documented in the *OSF DCE Application Development Reference* allow control over time zones by reading the **TZ** environment variable. The DCE for Windows NT product has been designed so as to not require a **TZ** definition. Do not use this variable to set time zones for DCE applications, because the results will be unpredictable. Set your system time zone according to standard Windows NT procedures (using the **Date/Time** applet in the Control Panel).

Structure of passwd

The particular structure of **passwd** depends on the underlying system. DCE for Windows NT uses a structure like that supported by 4.4BSD. The structure is:

```
struct  passwd  {
char              *pw_name;              /*  user  name  */
char              *pw_passwd;        /*  encrypted  password  */
int                     pw_uid;                    /*  user  uid  */
int                     pw_gid;                    /*  user  gid  */
time_t           pw_change;          /*  password  change  time  */
char               *pw_class;             /*  user  access  class  */
cchar              *pw_dir;                 /*  home  directory  */
char                *pw_shell;             /*  default  shell  */
time_t             pw_expire;           /*  account  expiration  */
};
```

Differences Between OSF DCE and Windows NT Examples

The *OSF DCE Application Development Guide* refers to files that do not exist on Windows NT systems, and illustrates commands and command syntax that do not work in a Windows NT environment. The example command line in Building Applications illustrates some of the differences from OSF DCE documentation compile examples when you compile DCE code on Windows NT. Note the following differences for writing applications on Windows NT systems:

- Object format files created by the IDL and C compilers have the file extension .OBJ instead of **.o**.
- The UNIX feature **fork** is available on Windows NT, but the recommended mechanism is to use the Win32 **CreateProcess** instead.
- DCE filenames and locations are similar but different on Windows NT. See DCE Directory Names for more information.
- Various UNIX commands used in the OSF DCE documentation do not work directly on Windows NT systems. Use the Windows NT equivalents; for example:

Command	**UNIX Name**	**Windows NT Name/Procedure**
List files (directory)	**ls**	**dir**
List processes	**ps**	Use the Task Manager, accessible throug **Ctrl-Alt-Del** key sequence or use **win32_** have the Microsoft Windows 32 SDK, you PView in the Win32 SDK Tools program g
Makefile	**make**	**nmake**
Set environment variable	% **setenv** *name "value "*	C:\> **set *name* =** *value*
Stop Process	**kill**	Use the Task Manager, accessible throug **Ctrl-Alt-Del** key sequence or use **win32_**
Output file to screen	**cat**	**type**

DCE Directory Names

During Windows NT installations, the typical DCE directories and subdirectories are created. Occasionally, a Windows NT subdirectory name or its path name may be slightly different from those on UNIX systems.

In addition, the naming conventions for Windows NT differ from the UNIX conventions in the following ways:

%DCELOC% replaces /opt. DCELOC is the environment variable containing the drive letter and directory you chose as the destination of your DCE software when you installed the kit.

Backslashes (\) replace slashes (/).

Examples:

UNIX Name
Windows NT Name

/opt/dcelocal/share/include/dce
%DCELOC%\dcelocal\include\dce

/opt/dcelocal/var/adm/time
%DCELOC%\dcelocal\var\adm\time

DCE Function Prototypes for C++ Applications

If your application encounters errors at link time, the errors could be caused by DCE prototype declarations that need to be enclosed in an extern ″C″ {} construction. For example, if the linker cannot resolve **dce_error_inq_text** , check the file that includes the prototype for that function. In this case, it is **dce_error.h** . Change the code as follows:

Change:

```
extern void IDL_STD_STDCALL_dce_error_inq_text (
            unsigned long          /* status_to_convert */,
            unsigned char*       /* buffer */t,
            int*                          /*status */
  ) ;
```

To:

```
#ifdef __cplusplus
extern ″C″ {
#endif
  extern void IDL_STD_STDCALL_dce_error_inq_text (
            unsigned long          /* status_to_convert */,
            unsigned  char*      /* buffer */,
            int*                          /*status */
  ) ;
  #ifdef __cplusplus
  }
  #endif
```

Memory Allocation in DCE Applications

DCE applications on Windows NT can be built with C-runtime libraries other than the C-runtime that DCE is built against. DCE applications can also be built with different versions of the Microsoft Visual C++ used by DCE. When this happens, use of standard C-runtime routine **free()** is not guaranteed to return memory

properly. If memory is not deallocated properly, the application may encounter access violations or events logged to the Application Event Log as well as memory growth.

A new function, **dce_free()**, has been provided that ensures memory allocated from DCE is properly deallocated. Use this routine whenever the DCE API documentation says to deallocate memory by calling **free()**. The following prototype describes **dce_free** and is defined by the header file **dce_free.h** contained in the DCE for Windows NT Application Developer's Kit option:

```
void dce_free ( void *ptr );
```

Using IBM VisualAge C++

DCE for Windows NT provides continuing support for the Microsoft Visual C++ compiler and adds support for the IBM VisualAge C++ compiler. The DCE for Windows NT Application Developer's Kit (ADK) option works with both the Microsoft and IBM compilers. IBM VisualAge C++ provides a consistent set of tools, compiler technology, and class libraries that enable the portability of C and C++ source code across multiple, heterogeneous environments.

For more information see:

- New Flags for Makefile
- IDL Compiler Flags
- Compiler Flags
- Linker Flags

New Flags for Makefile

To use the IBM VisualAge C++ compiler, you must generate a makefile. This makefile requires new idl compiler flags, compiler flags, and linker flags. The Application Developers Kit (ADK) contains example makefiles (named <**example**.mak> where **example** is the name of the DCE application) for each DCE application.

The example makefiles are dependent on the MSDEVDIR environment variable. If a machine has both Visual C++ and VisualAge C++ compilers installed, verify that the MSDEVDIR environment variable is not set in the session invoking the VACPP compiler.

For more information see:

- IDL Compiler Flags
- Compiler Flags
- Linker Flags

IDL Compiler Flags

The VisualAge C++ makefile provided with the DCE for Windows NT examples uses the following idl compiler flags to specify the compiler and preprocessor:

```
IDL = idl <options> -cc_cmd icc -cc_opt       \
          "-q -Ms -Gd+ -Gm+ -D_X86_=1 -DWIN32"          \
           -cpp_cmd icc -cpp_opt "-q -Pd+"
```

For more information see:

Compiler Flags

The compiler flags used in the DCE for Windows NT examples, including a possible set of includes, are:

```
INCLUDES = -I.  -I$(CPPMAIN)\bindings         \
                     -I$(CPPMAIN)\include                \
                     -I$(CPPMAIN)\sdk\winh              \
                     -I$(CPPMAIN)\sdk\winh\winnt      \
                     -I$(CPPMAIN)\sdk\winh\win95      \
CFLAGS     = -DWIN32 -Q -Ms -Gd+ -Gm+ -Ti+ -Ss -Su4 -C \
                      -D_X86=1 -DM_I86 $(INCLUDES)
CC              = icc
```

For more information see:

Linker Flags

Linker Flags

The linker flags used in the VisualAge C++ makefile provided with the DCE for Windows NT examples are the following:

```
LD             = ilink
LDFLAGS = -nologo -noe -map -debug
LIBS          = libdce.lib pthreads.lib wsock32.lib
```

Compiler Warnings from Function Assignments

The idl runtime has been modified to support both **_Optlink** and **__stdcall (_System)** and **__cdecl** calling conventions for memory management functions. For example, if you call **rpc_ss_swap_client_alloc_free(mallor,free)**, and **malloc()** and **free()** are **_Optlink** VisualAge C++ C-runtime functions, the idl runtime will call them both, putting arguments in registers for **_Optlink** and pushing arguments on the stack for **__stdcall (_System)** and **__cdecl**.

Although both linkage conventions are supported, the compiler will generate warnings from the stubs, which call **rpc_ss_set_client_alloc_free()**. The warnings will be similar to the following:

```
warning EDC0280: Function argument assignment between types "void*
(*__dcecl) (unsigned long)" and "void"*(*_Optlink) (unsigned long)"
is not allowed.
```

You can disregard the warning. The stub code does not have a problem. The idl compiler has generated code to allocate objects from the customer's C-runtime, in this example the VisualAge C++ C-runtime.

While you can safely ignore the stub compile warnings, you should check any similar warnings in the customer's main code. For example, the comparison function passed to the C-runtime function **qsort()** will cause an access violation because of the linkage convention mismatch.

Code Considerations

Certain coding considerations apply when you are using VisualAge C++.

For more informatin see:

File Streams

Memory Allocation and Deallocation

Structured Exception Handling (SEH)

File Streams

File streams are incompatible between the two runtime environments. The following code fragment would work properly in a Microsoft Visual C++ environment, but not in a mixed VisualAge C++ and Microsoft Visual C++ environment:

```
dce_error_inq_text ( *stp, err_string, &st) ;
dce_fprintf (stderr, ibm_msg_620, err_string) ;
```

In the mixed environment, the stream **stderr**, which can be any stream opened by **fopen()**, resides in the IBM C-runtime. This stream uses the VisualAge C++ data structure FILE from VisualAge C++ stdio.h. When the API **dce_fprintf()** is called into DCE, it dereferences **stderr** with the Microsoft Visual C++ FILE structure from Microsoft Visual C++ stdio.h, and finds the two streams incompatible. To solve this problem, use **dce_sprintf()** and **fwrite()**, remembering to free the memory returned from **dce_sprintf()** with **dce_free()**.

If you have porting code with many **dce_fprintf()** calls, you can override the **dce_fprintf()** behavior and implement a function like the following:

```
#if !defined (_MSC_VER) // If *not* a Microsoft compiler
#include <stdarg.h>
int IDL_STD_STDCALL dce_fprintf (FILE *fpStream, const unsigned32 msg_index, ...)
{
        char  *cpBuffer;
        va_list args;
        int        1;

        va_start (args, msg_index) ;
        cpBuffer = dce_sprintf (msg_index, args) ;
        va_end (arg) ;
        i = fprintf (fpStream, cpBuffer) ;
        dce_free (cpBuffer) ;

        return(i) ;
#endif                        //Not a Microsoft compiler
```

For more informatin see:

Memory Allocation and Deallocation

Structured Exception Handling (SEH)

Memory Allocation and Deallocation

Use **dce_free()** to free any memory that was allocated by DCE (from Microsoft's C-runtime). **dce_free()** calls into the Microsoft C-runtime. The following APIs require the use of **dce_free()**:

dce_msg_get ()
dce_msg_get_msg()
dce_sprintf()

dce_pgm_sprintf()
dce_aud_print()
dce_cf_find_name_by_key()
dce_cf_get_cell_name()
dce_cf_get_host_name()
dce_cf_dced_entry_from_host()
dce_cf_get_csrgy_filename()
dce_db_header_fetch()

For more informatin see:

Structured Exception Handling (SEH)

Structured Exception Handling (SEH)

When you want to use VisualAge C++ to build a C++ program, do not use the DCE TRY/CATCH/ENDTRY macros to catch DCE exceptions; otherwise, compilation errors will occur. The TRY/CATCH/ENDTRY macros use the Windows SEH keywords __**try** and __**except**, which are not valid keywords for the VisualAge C++ compiler in C++ mode compiling C++ modules.

Instead, use client programs to catch DCE exceptions to obtain status code for the DCE communications and server fault status.

Converting Applications from Microsoft RPC to DCE RPC

You can write your application, from the IDL files to the client and server source code, using the standard DCE API. To use the DCE RPC for Windows NT, make the following changes to your Microsoft RPC applications:

- Do *not* include **dceport.h** .
- Do *not* use the Microsoft Interface Definition Language (MIDL) stub compiler. Instead, use the DCE for Windows NT IDL stub compiler. This stub compiler provides all of the standard DCE data types.
- Do *not* use the Microsoft RPC API. Use the standard DCE API. For example, use the DCE RPC routine **rpc_binding_to_string_binding( )** instead of the Microsoft RPC routine **RpcBindingToStringBinding( )** .
- Build your DCE application using the guidelines documented in this book, instead of following the examples provided in *Distributing Applications Across DCE and Windows NT* .

dced Server Object Identities Restrictions

When you create a server object in **dced** on the UNIX platform**,** you normally specify the user ID (UID) under which the server will run. On the NT platform, this feature is not supported, for two reasons:

- The NT platform has no notion of a UID.
- The WIN32 process API requires a password if you want to start a process under another identity.

Therefore, on NT, when dced creates an instance of a server object, that server always runs under the system administrator identity. The UID attribute is ignored. In the case of certain server applications, this behavior may introduce a security vulnerability to the system.

dced Daemon Behavior Restrictions

The **dced** daemon for DCE NT always behaves as if it were started with the **-c** switch. It never requires privacy encryption for remote key table management. Clients who invoke remote keytab operations may have their operations encrypted with CDMF encryption if the client host does not support DES encryption.

dced Partial Service Mode Restriction

OSF DCE 1.2.2 provides a security feature for dced called *partial service mode* . This feature is not supported in the DCE NT V2.2 product.

Some of the OSF reference pages for **dced** objects, such as those for **hostdata** and **acl,** refer to the feature. Additionally, the OSF documentation mentions the **-local** argument that is used to update these objects when the local **dced** service is in partial service mode. Do not use the **-local** argument in **dcecp** commands.

Enable and Disable Endpoints Restriction

DCE for Windows NT does not use the dced endpoint mapper service, but instead uses Microsoft **rpcss** . (This is the only NT service that the product uses directly)

The following routines are not supported on DCE NT, because they modify endpoints registered specifically in the dced endpoint map:

- **dce_server_disable_service**
- **dce_server_enable_service**

To enable and disable endpoints, use the **rpc_ep** *API.

Note: This restriction does not impact the dce_server_register() service.

The **rpcss** service does not clean up stale endpoints. If a daemon does not clean up endpoints upon exit, use **rpccp** or the **ep-flush-binding** utility provided with this kit.

The **ep-flush-binding** utility accepts a single interface ID to delete a single endpoint or * to delete all endpoints from the endpoint map. The wildcard option should be used with extreme caution, as it will affect any RPC application on the system.

With Windows NT Version 4.0, **rpcss** cannot be stopped because it is essential to the operating system.

Commercial Data Masking Facility (CDMF)

The Commercial Data Masking Facility (CDMF) provides data privacy support for customers outside the U.S. and Canada. Like the Data Encryption Standard (DES) provided in OSF DCE, CDMF supports RPC application encryption and GSSAPI message encryption.

CDMF was developed as an alternative to DES. Because use of the DES encryption algorithm for data privacy is limited outside the U.S., CDMF has export approval from the U.S. government and is compatible with DES.

To support data privacy with CDMF, a new privacy level for communication between clients and servers is introduced. For GSSAPI message encryption, a new parameter for **gss_seal** and **gss_unseal** is added.

For more information see:

Application Development using CDMF

RPC APIs Supported by CDMF

GSS APIs Supported by CDMF

Application Development Using CDMF

DCE currently provides six levels of communication. To support data privacy with CDMF, a new privacy level for communication between clients and servers is introduced. Applications that do not have access to DES based data privacy need to use this new level. The seventh level is: **rpc_c_protect_level_cdmf_privacy**.

CDMF uses the DES encryption algorithm, but exposes a weaker 40-bit key to an application instead of the full 56-bit key in DES. The 40-bit key provides consistent application level data encryption for customer applications that might otherwise lack the functionality.

For GSSAPI message encryption, a new *qop* parameter for **gss_seal** and **gss_unseal** is introduced. The parameter is: **GSSDCE_C_QOP_CONF_CDMF**.

For more information see:

GSS APIs Supported by CDMF

RPC APIs Supported by CDMF

The APIs that include protection level parameters and the new seventh level for CDMF, **rpc_c_protect_level_cdmf_privacy**, are:

- **rpc_binding_inq_auth_caller**
- **rpc_binding_inq_auth_client**
- **rpc_binding_inq_auth_info**
- **rpc_binding_set_auth_info**
- **rpc_mgmt_inq_dflt_protect_level**

The **rpc_c_protect_level_cdmf_privacy** level performs protection as specified by all of the previous levels and also encrypts each remote procedure call parameter value. This protection level provides a lower level of packet privacy than **rpc_c_protect_level_pkt_privacy**.

GSS APIs Supported by CDMF

The two APIs that support message encryption are:

- **gss_seal**

  CDMF includes a constant for the confidentiality algorithm as part of the **qop_req** input parameter. The constant is:

  ```
  GSSDCE_C_QOP_CONF_CDMF
  ```

- **gss_unseal**

  CDMF includes a value for the confidentiality algorithm as part of the **qop_state** output parameter. The value is:

```
0x00010000 (GSSDCE_C_QOP_CONF_CDMF)
```

Use of Threads

The DCE for Windows NT product supports the POSIX 1003.4 Draft 4 interface through native NT threads, ensuring a standards-based thread capability as well as interoperability between Microsofts native Windows NT threads and DCE Threads.

Microsoft applications that use native Windows NT threads can continue to do so while calling DCE services, and new application modules can be written using DCE pthreads.

Pthreads Return Error Value Restriction

Most public Pthreads API return error values in the variable **errno.** This variable is specific to a particular C-Runtime DLL. For example, the two C-Runtime DLLs MSVCRT20.DLL (installed with Visual C++ Version 2.0) and MSVCRT40.DLL (installed with Visual C++ Version 4.X) have their own, independent **errno** storage locations.

The Pthreads API available in DCE for Windows NT has been built with Visual C++ Version 5.0. If an application is built with Visual C++ Version 4.X, then, it can correctly access **errno** set as a result of call to the Pthreads library. However, if the application is built with any other compiler other than Visual C++ Version 4.X, and it accesses the value **errno,** then the value of **errno** will be retrieved from a location within the C-Runtime Dll used by the application. This will result in retrieving an incorrect value for **errno** .

The DCE for Windows NT product solves this problem by providing this additional Pthreads function:

```
int  pthread_get_errno_np (void)
```

This function retrieves a specific **errno** value in the event of a pthread function call error. Applications using any C-Runtime DLL other than MSVCRT40.DLL should use this function to access **errno** .

Unsupported DECthreads Interface Routines

The following DECthreads interface routines are not implemented and are not supported on Windows NT:

- **cma_attr_set_sched/cma_attr_get_sched**
- **cma_attr_set_priority/cma_attr_get_priority**
- **cma_thread_set_sched/cma_thread_get_sched**
- **cma_thread_set_priority/cma_thread_get_priority**
- **pthread_attr_setsched/pthread_attr_getsched**
- **pthread_setscheduler/pthread_getscheduler**
- **pthread_attr_setprio/pthread_attr_getprio**
- **pthread_setprio/pthread_getprio**
- **pthread_attr_getguardsize_np/pthread_attr_setguardsize_np**

If these routines are called by your application (for example, you are using the same code base on multiple operating systems), you can conditionally compile the

unsupported routine calls. If you do not want to do this, then ensure that your code checks these functions' return values for unimplemented/unsupported exceptions or catches the unimplemented exception and takes appropriate action.

Calling one of the listed **pthread_*** routines returns a -1 and set **errno** to **ENOSYS** . Calling one of the above **cma_*** routines raises the **cma_e_unimp** exception.

The following DECthreads interface routines are *not* implemented, but do not return errors or generate exceptions. ***Do not use these routines***.

- **cma_attr_get_guardsize/cma_attr_set_guardsize**
- **cma_attr_set_inherit_sched/cma_attr_get_inherit_sched**
- **pthread_attr_setinheritsched/pthread_attr_getinheritsched**
- **cma_stack_check_limit_np**

The **cma_attr_set_stacksize** and **pthread_attr_setstacksize** interface routines do not change the stacksize of newly created threads. DECthreads threads on NT are created with their stack size set to the same size as the primary thread of the process they are created in. The stack size grows as needed. See the WIN32 documentation for **CreateThread()** . No errors or exceptions are generated as a result of using the **cma_attr_set_stacksize** and **pthread_attr_setstacksize** routines. Their corresponding routines **cma_attr_get_stacksize** and **pthread_attr_getstacksize** return the values set using **cma_attr_set_stacksize** and **pthread_attr_setstacksize** ; but this is not useful since DECthreads threads are not created using the stack size attribute.

Return Values for pthreads_setcancel() and pthread setasynccancel()

The DECthreads documentation incorrectly specifies the return values for **pthread_setcancel** and **pthread_setasynccancel**. On successful completion, these routines return the previous state of cancelability.

If an error condition occurs, these routines return -1 and set **errno** to the corresponding error value.

Possible return values are as follows:

Return	Error	Description
CANCEL_ON		Successful Completion
CANCEL_OFF		Successful Completion
-1	[EINVAL]	The specified state is not CANCEL_ON or CANCEL_OFF.

Data Returned by pthread_getspecific

The DECthreads documentation does not fully specify the data returned in the value parameter for **pthread_getspecific**. The routine's description should say:

The pthread_getspecific routine obtains the per-thread context associated with the specified key for the current thread. If a context has not been defined for the key in this thread (ie. pthread_setspecific has not been successfully executed), NULL is returned in value.

Maximum Time Interval for cma_delay and pthread_delay_np

These routines accept a maximum time_interval/interval value of 4294967.295 seconds. Values greater than this will raise the **cma_e_badparam** exception for **cma_delay** and return -1 and set **errno** to **EINVAL** for **pthread_delay_np** .

If the interval argument passed to **pthread_delay_np()** is more than an unsigned word value (roughly two weeks' time), the call returns without any delay. To work around this problem, use a counter and repeatedly reissue **pthread_delay_np( )** with smaller values.

Chapter 5. Public Key Certificate Login

OSF DCE Version 1.2.2 includes enhancements that support the public key protocol. This public key preauthentication protocol is used by DCE security clients to obtain Ticket Granting Tickets (TGTs) for users. OSF-RFC 68.4 extends the DCE 1.2.2 public key protocol to allow DCE users to use an X.509v3 digital certificate and its associated public key pair to prove their identity to the DCE authentication service. The following information describes the DCE for Windows NT, Version 2.2 implementation of Public Key Certificate Login which is based on OSF-RFC 68.4. This implementation requires the Entrust Public Key Infrastructure (PKI).

Overview of Public Key Login

DCE for Windows NT, Version 2.2 allows DCE users to prove their identity to the DCE authentication service using an X.509v3 digital certificate and its associated public key pair rather than a shared-secret key password. One benefit of this authentication mechanism is that, in the event of a compromise of the DCE Security Server, public key users do not have any identifying information exposed to the intruder. With shared-secret key authentication, all user secret keys could be revealed to an intruder. Another immediate benefit is that the basic authentication flows are made more secure by virtue of public key cryptographic methods.

This enhancement is intended for customers who are currently using the Entrust PKI and have a need to map Entrust users to DCE users for authentication and access to resources provided by DCE. The changes only apply to the acquisition of the initial TGT. Acquisition of additional service tickets occurs in the traditional manner once the TGT has been received.

In addition to changes to the information exchanged by the DCE client and the DCE Security Server, a new server, the Identity Mapping Server (IDMS) is provided with this enhancement. This server is called by the DCE Security Server to map users' digital certificates to DCE principal names. The following figure illustrates the authentication flow established by this enhancement.

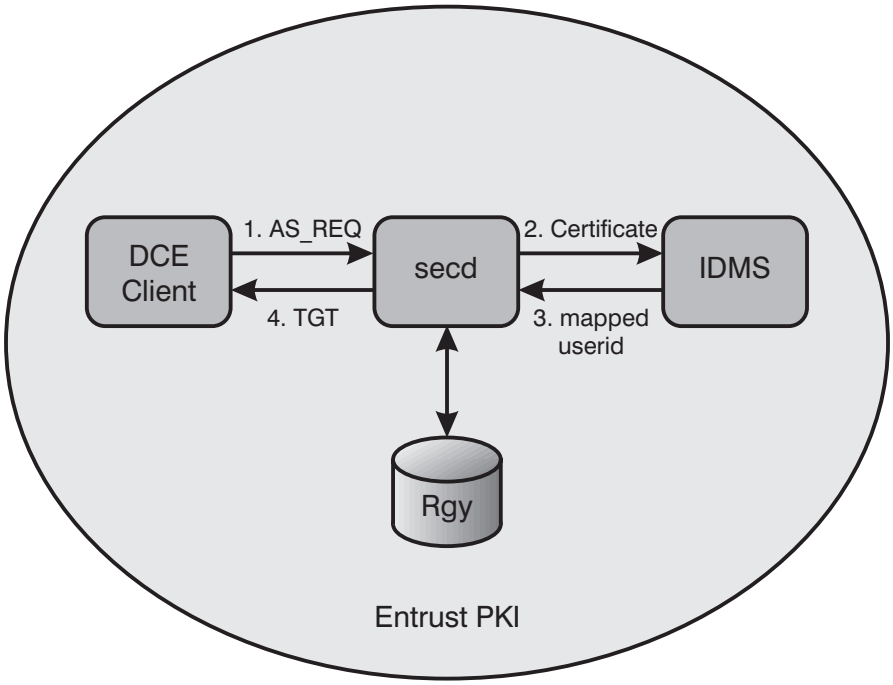


Figure 1. Public Key Certificate Authentication Flow

Entrust Prerequisites for Using Public Key Certificate Login

This enhancement requires an Entrust PKI. Additionally, DCE client and security server systems must have the Entrust client installed.

For each user of this enhancement, the Entrust administrator must create an Entrust user and issue to each user a public key pair for both signing and encrypting. The public keys are stored in public key certificates that are digitally signed by the Entrust Certificate Authority (CA) and stored in the user's X.500 directory entry. The private keys are stored in the Entrust user's profile and are protected by a passphrase (referred to as a password in the Entrust documentation). The Entrust profile associated with a user must reside on the user's client system.

Additionally, the DCE principal associated with each Entrust user must be added as an attribute of the user's X.500 directory entry. Entrust requires an X.500 directory and tracks its users by their X.500 Distinguished Name (DN). Tools for creating and managing Entrust users are provided by Entrust and are not integrated with existing DCE tools. Refer to the documentation provided with the Entrust PKI for detailed information on creating Entrust users.

The Entrust administrator must also create an Entrust user for the DCE Security Server to use. The certificates and corresponding public key pairs for this Entrust user can be used by all security servers in the cell, or individual Entrust users can be created for each DCE Security Server. An Entrust user profile must reside on each DCE Security Server. The IDMS will use the same Entrust user profile as the one being used by the security server on that system.

Enabling Public Key Certificate Login

There are two DCE Security Server changes required to enable Public Key Certificate Login. First, the DCE configuration tools have been updated to support the addition of the IDMS and to allow specification of the location of the Entrust user profile used by the DCE Security Server. Second, a new version of the DCE registry, 1.2.2a, has been created.

Public Key Certificate Login is enabled for a DCE cell only if the Master Security Server for the cell is running at version 1.2.2a. Version 1.2.2 is the default for cells configured using DCE for Windows NT, Version 2.2. A cell that has been migrated from a previous DCE version will continue to run the same security server version it was running before the migration.

The **dcecp registry modify** command should be used to enable Public Key Certificate Login by modifying the registry version number.

```
dcecp> registry modify -version secd.dce.1.2.2a
dcecp>
```

Warning: If this command is issued while any Security Replicas in the cell are running a version of DCE that does not support the 1.2.2a Security Server Version, the replicas will be automatically shut down.

See the Migration section of the *IBM DCE for Windows NT, Version 2.2: Quick Beginnings* for more detail on migrating an existing DCE cell to DCE for Windows NT, Version 2.2.

At least one IDMS is required in a DCE cell which uses Public Key Certificate Login. Ideally, an IDMS should be configured on every security server in the cell. The following list details the command line configuration options. These options are also supported by **DCEsetup** and are included as options when configuring a DCE Security Server.

- To enable Public Key Certificate Login when configuring a Security Server (Master or Replica), with an IDMS on this machine, specify the **sec_srv or sec_rep** component (as appropriate), the **idms_srv** component, and include the following options:

  -certificate_based_login yes

  -kdc_ini_file *kdc_ini_file*

  -kdc_profile *kdc_profile*

  -kdc_passphrase *kdc_passphrase*
- To configure a Security Server with Public Key Certificate Login enabled, but without an IDMS, omit the **idms_srv** component option in the previous example.
- To enable Public Key Certificate Login on an already configured Security Server, specify the following options:

  -certificate_based_login yes

  -kdc_ini_file *kdc_ini_file*

  -kdc_profile *kdc_profile*

  -kdc_passphrase *kdc_passphrase*

  As long as the security server version is 1.2.2a, the next time that **secd** is started, Public Key Certificate Login support will be enabled.

- To add an IDMS to an already configured Security Server that does not have Public Key Certificate Login enabled, specify the **idms_srv** component option with the options:

  -certificate_based_login yes

  -kdc_ini_file *kdc_ini_file*

  -kdc_profile *kdc_profile*

  -kdc_passphrase *kdc_passphrase*
- To add an IDMS to an already configured Security Server that does have Public Key Certificate Login enabled, just specify the **idms_srv** component option.

For more information on these configuration options, see “config.dce” on page 23.

Managing DCE User Authentication

You manage preauthentication for a given principal by attaching an instance of the *pre_auth_req* ERA to the principal and specifying a value to indicate *the lowest level protocol* the DCE Security Service should accept for the principal, as follows:

0 **(NONE)** Specifies that the DCE Security Service should accept, from this principal, login requests that use any of the five protocols (including the pre-DCE Version 1.1 protocol.) This is the least secure level and is provided only to enable DCE Version 1.1 servers to accept login requests from pre-DCE Version 1.1 clients.

Warning: Failing to attach an instance of the *pre_auth_req* ERA to a principal is equivalent to specifying **0 (NONE)**.

1 **(PADATA-ENC-TIMESTAMPS)** Specifies that the DCE Security Service should accept, from this principal, login requests using the timestamp, third-party, public key, or public key certificate protocol. The timestamp protocol protects against attackers masquerading as security clients and attacking replies from the DCE Authentication Service. The protocol is still vulnerable to attacks by processes capable of monitoring the network.

2 **(PADATA-ENC-THIRD-PARTY)** Specifies that the only login requests the DCE Security Service will accept from this principal are those using the third-party, public key, or public key certificate protocol. This protocol offers a high level of DCE preauthentication and provides protection against attacks. With third-party preauthentication, all authentication data sent over the network is encrypted with a strong random key known only to the local machine principal and the DCE Security Service.

3 **(PADATA-ENC-PUBLIC-KEY)** Specifies that the only login requests the DCE Security Service will accept from this principal are those using the OSF 1.2.2 public key or public key certificate protocol.

4 **(PADATA-ENC-PUBLIC-KEY-CERTIFICATE)** Specifies that the only login requests the DCE Security Service will accept from this principal are those using the public key certificate login protocol.

When the authentication service receives a login request for a principal, it always attempts to respond using the same protocol as the request, unless the *pre_auth_req* ERA value for that principal forbids it to do so. Table 2 on page 76 provides a matrix describing the actions taken by the authentication service under the various combinations of login (authentication) request type and *pre_auth_req* ERA value.

For complete information on the details of DCE authentication (including the operation of the preauthentication protocols), see the *OSF DCE Application Development Guide — Core Components*.

The following is an example of a **dcecp** command to modify a principal and attach a *pre_auth_req* ERA specifying that Public Key Certificate Login is required.:

```
dcecp> principal modify smitty -attribute {pre_auth_req 2}
dcecp>
```

Setting the *pre_auth_req* ERA is not necessary if users are allowed, but not required to use Public Key Certificate Login. For further information on how to use **dcecp** to attach ERAs to principals, see ″Creating and Using Extended Registry Attributes″ in the *OSF Administration Guide — Core Components*.

Authenticating Using Public Key Certificate Login

The DCE login interfaces were not changed for this enhancement, however the meanings of some of the input values have changed. Using Public Key Certificate Login, you can log in and establish your identity by providing the name of your Entrust user profile instead of your DCE principal name. Instead of a DCE password, you need to specify the passphrase that is used to unlock the private key stored in your Entrust user profile.

The DCE APIs affected by this enhancement are **sec_login_validate_identity()** and **sec_login_valid_and_cert_ident()**. Neither of these APIs have any additional flags or arguments but instead interpret the meanings of existing arguments differently. This allows existing login utilities and platform specific integrated login mechanisms to authenticate to DCE with an Entrust public key certificate, without requiring rewriting or recompilation.

As an example, if your Entrust profile is **henry.epf** and your passphrase is **Rottweiler9**, you can log in to DCE using the following command:

```
dce_login henry Rottweiler9
```

Your Entrust profile and passphrase, along with the Entrust client's **entrust.ini** file, allow DCE routines to call the Entrust **ETLogin()** API. This is the basis for all further public key signing, verifying, encrypting and decrypting operations through Entrust. These operations are for constructing and processing the preauthentication data used to validate the user during login to DCE.

Your Entrust profile and passphrase are passed as arguments on the security login APIs described previously. The **entrust.ini** file is stored in a well-known location on the Entrust client. On Windows NT, the **entrust.ini** file is stored in the directory where Windows NT is installed (typically **C:\WINNT**). The full path to your Entrust profile can be obtained from the **entrust.ini** file as follows:

1. The most recently used profiles listed in the **entrust.ini** file are searched.
2. The **DefaultProfileLocation** specified in the **entrust.ini** file is searched.

Falling Back to Traditional Authentication

The structure of the login interfaces allow for a ″fallback″ to a shared-secret-key login (that is, using the traditional DCE password) if login using the Public Key Certificate Login fails. This is because the DCE Security client runtime will build and send preauthentication data for the public key certificate login protocol, and either the DCE third-party protocol or the DCE timestamps protocol when it makes an authentication request to the DCE Security Server. If you choose to name your Entrust user profile with your DCE principal name, and your Entrust passphrase matches your traditional DCE password, this fallback will be transparent to you. The *pre_auth_req* ERA is used to determine whether fallback is allowed.

The Identity Mapping Server

The Identity Mapping Server (IDMS) is a new RPC server responsible for mapping a user's public key certificate to a DCE principal name. At least one IDMS is required in a DCE cell that uses Public Key Certificate Login. Ideally, an IDMS should be configured on every Security Server in a DCE cell which uses Public Key Certificate Login. The IDMS is called by the DCE Security Server when a user logs in to DCE using a public key certificate to authenticate. The IDMS supports the need to map many Entrust users to one DCE user as well as the more traditional one-to-one user mapping.

The Identity Mapping Server requires the DCE principal associated with an Entrust user to be added as an additional attribute of the user's X.500 directory entry. Entrust requires an X.500 directory and tracks its users by their X.500 DN. Specifically, an attribute named **dcePrincipal** must be added to the directory schema for objects representing Entrust users. Entrust users are typically of type **organizationalPerson** or **entrustUser** or both. The mapping from an Entrust user to a DCE principal is then done using the X.500 DN in the user's certificate, and performing a directory lookup to find the user's **dcePrincipal** attribute. See the documentation provided with the X.500 Directory Service for additional information on adding attributes.

Identity mapping is performed in this manner, because it is more flexible to have the DCE Security Server call a separate server to obtain the mapping. This makes it possible to customize the mapping algorithm. Because identity mapping policies will vary based on individual customer requirements, source code for the default IDMS is provided as a DCE example program.

Public Key Interoperability Between DCE Versions

Table 2 describes how login requests are handled between different versions of DCE that are in a single cell. Only Server Versions 1.1 or higher are included in this table because Pre-1.1 Servers always ignore preauthentication data in the login request and return a Pre-DCE Version 1.1 (unpreauthenticated) response.

Table 2. DCE Authentication Interoperation

Login Request Type	Versions 1.1 and 1.2 Server Response	Version 1.2.2a Server Response
DCE Version 1.0		

Table 2. DCE Authentication Interoperation (continued)

Login Request Type	Versions 1.1 and 1.2 Server Response	Version 1.2.2a Server Response
From any client.	Preauthentication. Checks for *pre_auth_req* ERA instance: If no ERA exists, or existing ERA has *value*=**0** (**NONE**), returns DCE Version 1.0 (unpreauthenticated) response. Otherwise, rejects login request.	Preauthentication. Checks for *pre_auth_req* ERA instance: If no ERA exists, or existing ERA has *value*=**0** (**NONE**), returns DCE Version 1.0 (unpreauthenticated) response. Otherwise, rejects login request.
TIMESTAMPS		
From DCE Version 1.1 and greater clients.	Preauthentication. Checks for *pre_auth_req* ERA instance: If no ERA exists, or existing ERA has *value*=**0** (**NONE**) or *value*=**1** (**PADATA-ENC-TIMESTAMPS**), returns DCE Version 1.1 **TIMESTAMPS** response. If existing ERA has *value*=**2** (**PADATA-ENC-THIRD -PARTY**), rejects login request. 1.2 Server Response: Also rejects login request if ERA has *value*=**3 (PADATA-ENC-PUBLIC-KEY)**	Preauthentication. Checks for ERA instance: If no ERA exists, or existing ERA has *value*=**0 (NONE)** or *value*=**1 (PADATA-ENC-TIMESTAMPS)** returns DCE Version 1.1 TIMESTAMPS response. If existing ERA has *value*=**2 (PADATA-ENC-THIRD-PARTY)**, *value*=**3 (PADATA-ENC-PUBLIC-KEY)**, or *value*=**4 (PADATA-ENC-PUBLIC-KEY -CERTIFICATE)**, rejects login request.
THIRD -PARTY		
From DCE Version 1.1 and greater clients.	1.1 Server Response: Preauthentication. Returns DCE Version 1.1 **THIRD-PARTY** response. 1.2 Server Response: Preauthentication. Checks for *pre_auth_req* ERA instance: If ERA exists and has *value*=**3 (PADATA-ENC-PUBLIC-KEY)**, rejects login request. Otherwise, returns **THIRD-PARTY** response.	Preauthentication. Checks for *pre_auth_req* ERA instance: If ERA exists and has *value*=**3 (PADATA-ENC-PUBLIC KEY)** or *value*=**4 (PADATA-ENC-PUBLIC KEY -CERTIFICATE)**, rejects login request. Otherwise, returns **THIRD-PARTY** response.
PUBLIC- KEY		
From DCE Version 1.2 .2 clients(excluding IBM DCE for Windows NT Version 2.2 clients).	1.1 Server Response: Preauthentication. Returns DCE Version 1.1 **THIRD-PARTY** response. 1.2 Server Response: Preauthentication. Returns DCE Version 1.2.2 **PUBLIC-KEY** response.	Preauthentication. Checks for *pre_auth_req* ERA instance: If ERA exists and has *value*=**4 (PADATA-ENC-PUBLIC-KEY -CERTIFICATE)**, rejects login request. Otherwise, returns DCE Version 1.2.2 **PUBLIC-KEY** response.
PUBLIC-KEY-CERTIFICATE		
From IBM DCE for Windows NT Version 2.2 clients.	1.1 Server Response: Preauthentication. Returns DCE Version 1.1 **THIRD-PARTY** response. 1.2 Server Response: Preauthentication. Checks for ERA instance: If ERA exists and has *value*=**3 (PADATA-ENC-PUBLIC-KEY)**, rejects login request. Otherwise, returns **THIRD-PARTY** response.	Preauthentication. Returns **PUBLIC-KEY-CERTIFICATE** response.

Restrictions of Public Key Certificate Login

There are several limitations for accounts that use Public Key Certificate Login. These include:

- The **kinit** command cannot be used to refresh expired DCE credentials unless the DCE password is provided. Using the Entrust user profile and passphrase for

this refresh operation is not supported. If the Entrust user profile name and passphrase are synchronized with the DCE principal name and password, this limitation is transparent to the user.

- When multiple Entrust users are mapped to a single DCE principal, the level of detail of DCE functionality such as auditing and access control is reduced. Only the DCE principal information is available and used in audit records and access control checks.
- If the *pwd_val_type* ERA that requires password strength checking is attached to a DCE principal, these checks are only enforced on the DCE password for that principal. The Entrust PKI establishes a separate set of rules which are enforced on the Entrust passphrase.
- The key management API is used only by applications that use the shared-secret key authentication protocol. Application servers cannot use the public key certificate login protocol.
- When using GSSAPI, the DCE administrator must set up an account in the DCE registry database for the initiator and the acceptor. The acceptor cannot use Public Key Certificate Login. No restrictions apply to the account for the initiator.

Summary of the Steps Required to Use Public Key Certificate Login

The following list summarizes the steps required to use the Public Key Certificate Login enhancement.

1. Configure an Entrust Public Key Infrastructure and create Entrust users.
2. Install the Entrust client on systems that are, or will be, DCE clients or Security Servers.
3. Place Entrust user profiles on the DCE client and Security Server systems.
4. Configure or migrate a DCE cell to registry version 1.2.2a.
5. Configure the IDMS and the Public Key Certificate options on each Security Server in the cell.
6. Add the **dcePrincipal** attribute to users' X.500 directory entries.
7. Users can log in, specifying the name of their Entrust profile and the passphrase protecting their profile.

Chapter 6. Multiple Network Interface Cards

DCE for Windows NT, Version 2.2 can operate on systems with multiple network cards or multiple IP addresses by providing the capability to restrict the operation of DCE to a single network interface and IP address.

Multiple Network Interface Restriction

Most systems have a single Network Interface Card (NIC) and a single IP address. Such systems require no special intervention before DCE for Windows NT, Version 2.2 can be configured. More complex system configurations may have multiple NICs and multiple IP addresses for each NIC. DCE for Windows NT, Version 2.2 can operate on systems with multiple NICs or multiple IP addresses, given the restrictions described in the subsections. Failure to obey these restrictions will result in a DCE configuration that starts successfully only sporadically or fails to start at all. In brief, this is a mis-configured system

In order to determine the IP configuration of a particular system, use the **ipaddr** progarm supplied with the DCE for Windows NT, Version 2.2 kit. (The kit must be installed to use this program, however, DCE need not be configured.) From a DOS-window, run the **ipaddr** program as follows:

```
>ipaddr
```

The program displays the IP configuration of the system on which it was run.

```
**  Environment Variables  **

RPC_UNSUPPORTED_NETIF  (undefined)
RPC_SUPPORTED_NETADDRS (undefined)

**  Detected IP addresses  **

Net Interface  IP address    Subnet mask   Broadcast address   Hostname
-------------  -----------   ------------  ------------------   --------
   DC21X41      1.2.3.79 (S)  255.255.0.0   255.255.255.255     dcedds
                1.2.3.235 (S) 255.255.0.0   255.255.255.255     dcedce
                1.2.3.80 (S)  255.255.0.0   255.255.255.255     dcerpc
   DC21X42      1.2.3.30 (S)  255.255.0.0   255.255.255.255     dcedyn
                1.2.3.31 (S)  255.255.0.0   255.255.255.255     dceecd

   DC21X43      1.2.3.50 (D)  255.255.0.0   255.255.255.255     dcesta
                 (S=Static, D=Dynamic)
```

From the previous data, exactly one network interface, and exactly one corresponding IP address must be chosen. (Manually-assigned addresses are tagged ″S″ for static, while DHCP-assigned addresses are tagged ″D″ for dynamic. Either type of address may be chosen, although the use of DHCP addresses should be restricted to client-systems only.)

Specifying the Network Interface

DCE for Windows NT, Version 2.2 can successfully operate on systems with multiple NICs, but one and only one of the available NICs must be selected, using the environment variable **RPC_UNSUPPORTED_NETIFS**. For example, assume a system is equipped with three NICs designated DC21X41, DC21X42, and

DC21X43. Only one of these three may be used, and if DC21X42 is selected, then the other two NICs are specified to the RPC runtime as unsupported by setting the environment variable.

To set the environment variable, use the Windows NT Control Panel System>Environment tab. Establish the focus by clicking in the System wide variables box, then type:

RPC_UNSUPPORTED_NETIFS

in the Variable text box and

DC21X41:DC21X43

in the Value text box.

Specifying the IP Address

Windows NT allows a single computer to be configured with many IP addresses. The environment variable **RPC_SUPPORTED_NETADDRS** specifies which addresses are intended for use by RPC. Currently, only one should be specified.

For example, assume a system has the following IP address configured:

```
1.2.3.31
```

If only the second address is intended for use by RPC, then specify that address to RPC by setting the environment variable.

To set the environment variable, use the Windows NT Control Panel System>Environment tab. Establish the focus by clicking in the System wide variables box, then type:

RPC_SUPPORTED_NETADDRS

in the Variable text box and

1.2.3.31

in the Value text box.

Note: The IP address chosen must be consistent with (that is configured on) the NIC that is chosen. Assuming that the actions described in this section and the section “Specifying the Network Interface” on page 79 were taken, rerunning the **ipaddr** utility will show the appropriate changes, as shown in the following:

```
**  Environment variables  **

   RPC_UNSUPPORTED_NETIFS parsed as:  DC21X41 :DC21X43
   RPC_SUPPORTED_NETADDRS parsed as:  1.2.3.31

**  Detected IP addresses  **

Net Interface  IP address    Subnet mask   Broadcast address   Hostname
-------------- -----------   ------------  ------------------  --------
   DC21X42     1.2.3.31 (S)  255.255.0.0   255.255.255.255     dceecd
                 (S=Static, D=Dynamic)
```

Notice that the **ipaddr** utility now displays only a single NIC and a single, associated IP address. If the display does not reflect the desired result, then re-specify the environment variables until the desired configuration is achieved. (Please remember to create a new DOS window each time the environment variables are modified. Otherwise, the changes you make will not be reflected in the output of the **ipaddr** utility.) Once the IP configuration is correct, the system should be rebooted before DCE is configured and started.

Chapter 7. Auto-Start and Integrated Login

DCE for Windows NT provides improvements in the following areas:

- "Auto-Start"
- "Integrated Login"

Auto-Start

DCE for Windows NT provides an auto-start option that automatically starts DCE services during Windows NT startup. When enabled, this feature adds the DCE Auto-Start Service to the list of services that are started automatically as part of the Windows NT startup procedure.

The auto-start feature is available through DCEsetup. The online help file, *Configuring with DCEsetup*, contains a full description of DCEsetup and its features.

Integrated Login

DCE for Windows NT provides the ability to configure your Windows NT system so that when a user logs in to Windows NT, that user is automatically logged in to DCE as well, through **dce_login**. The integrated login feature is available during configuration with DCEsetup.

Integrated login requires that the user name and password being used to log in to Windows NT at system startup time is the same as the principal name and password for the DCE cell.

Note: The user name and password are case sensitive.
The following procedures describe how to create user names and passwords.

To create a new DCE account and password that matches an existing Windows NT username and password, login as **cell_admin**:

1. From the **Programs** menu, click **DCE for Windows NT**, and then click **DCE Director**.
2. Click **Users** and then click either **Clone** or **Create** from the **Actions** menu.
3. On the **Create** menu, create a new account using your existing Windows username.

Or, to create a new Windows NT username and password that matches an existing DCE account and password:

1. Double-click the Passwords icon in the Control Panel.
2. Click the Change Windows Password button to modify your password to be consistent with the corresponding DCE account's password.

The integrated login feature automatically updates your DCE password whenever the you change it with the Change Password command button on the Login box or the Secure Attention sequence (**Ctrl-Alt-Delete)**. The passwords will *not* remain synchronized if you change them with the User Manager utility.

After you start Windows NT integrated login is disabled until you start DCE. If you attempt to log in during this time, login pauses for up to 2 minutes, waiting for DCE integrated login to be enabled.

Note: Each time you log into DCE, a fresh set of DCE credentials is obtained. These credentials are available to all of your processes in the system. However, the credentials are affected by normal cell and account policies and remain subject to credential expiration. If you remain logged on to Windows NT longer than your credential lifetime, you will need to do one of the following:

- Use **kinit** to refresh existing credentials
- Use **dce_login** to obtain new credentials

The online help file, *Configuring with DCEsetup*, contains a full description of DCEsetup and its features.

Note: If you are using the integrated login feature, and you are not using English, the XPG4 and DCE environment variables must be set in the system environment before the operating system is rebooted, in order for DCE character data and messages to be handled and displayed correctly during login.

Chapter 8. CDS Enhancements

DCE for Windows NT provides improvements in the following areas:

- "Inline CDS Clerk"
- "CDS Cache Restructuring"
- "CDS Preferred Clearinghouse"

Inline CDS Clerk

The Inline CDS Clerk runs as a shared library running within the client's address space. This shared library clerk code, known as the *inline clerk*, eliminates the necessity and overhead of a separate process for every unique user name attempting to use CDS.

CDS Cache Restructuring

The cache is now divided into a shared cache and private caches. These private caches are also known as *per user* caches.

The shared cache contains data that is global in nature, such as CDS clearinghouses, and that is accessible to all users. The CDS Advertiser controls the shared cache—creating it, updating it, and periodically flushing it. User applications can read data in the shared cache, but cannot write data to it.

Per user caches contain private data such as CDS objects that can be viewed only by the cache's owner. If a per user cache is populated with all the data that an application needs, that application can run without starting the Advertiser.

When a user application requires CDS information, the per user cache is searched. If the information is not found, the search is extended to the shared cache. If the information is not located there, the CDS server is contacted.

CDS Preferred Clearinghouse

This enhancement improves performance at CDS clients by ranking clearinghouses in the order in which they should be contacted by the client for CDS information. This can be accomplished automatically through the use of defaults associated with the location of cds clients with respect to cds servers or by manual overrides made by cell administrators.

This enhancement is useful in situations where, for example, there are multiple high-performance LANs connected by a low-performance WAN, and there are CDS replica clearinghouses in each of the LANs. With this feature, the clearinghouse with the best ranking is the one on the machine with the server, followed by one on the same lan with the client. The more local clearinghouses are preferred over distant clearinghouses; clients will use the distant clearinghouses only when the local clearinghouses are unable to satisfy a request. The administrators can override the defaults in order to more specifically order communications with clearinghouses.

The preferencing is achieved by assigning a rank to each clearinghouse. A rank is a 16-bit unsigned integer (range 0-65535). Lower numbers are preferred over higher numbers (and a rank of 65535 means ″don't ever use this clearinghouse″).

These ranks are specified in a text preference file called **%DCELOC%\dcelocal\etc\cds_serv_pref**. The format of the file is one clearinghouse name and one rank on each line of the file. Blank lines and comments (″#″ to the end of the line) are ignored. Ranks can be 0-65535 (0x0000-0xffff) and may be specified in decimal, octal (with leading ″0″) or hex (with leading ″0x″). Clearinghouse names can be in any of the following formats:

```
/.../cellname/foo_ch
 /foo_ch
 foo_ch
 /.:/foo_ch
```

If the clearinghouse's cellname is not specified, the local cell is assumed.

Example file:

```
/.:/foo_ch  50  # most preferred clearinghouse
 /.:/bar_ch  100
 /.../mycellname/baz_ch 100
```

If a clearinghouse is not mentioned in the prefs file, a rank will be calculated for it (thus, you only need to specify ranks for clearinghouses whose default ranks are to be overridden). The default ranks are calculated based on IP address:

- Clearinghouses with addresses that match the local host address get a default rank of 5000.
- Clearinghouses on the same IP subnet as the local host get a default rank of 20000.
- Clearinghouses on the same IP network as the local host get a default rank of 30000.
- All other clearinghouses get a default rank of 40000.

The clearinghouse preferences file is read upon cdsadv startup and the values are cached. If you change rank values, you must stop the CDS client, remove the cache, then restart the CDS client.

The following commands will now include a rank attribute:

```
dcecp -c cdscache show -clearinghouse /.:/foo_ch

cdscp show cached clearinghouse /.:/foo_ch
```

Chapter 9. Credential Cache Cleanup

DCE for Windows NT provides enhancements for optimizing disk space not available in OSF DCE. One enhancement, the credentials cache cleanup feature, consists of a thread that executes at regular, settable intervals to clean up the credentials directory. It searches the directory for files belonging to credentials that have expired and deletes those files, freeing up disk space.

The default interval (sometimes referred to as the *grace period*) is 7 days from the time the credential expires. You need to set the **credcleangrace** registry key value there. By default, this value does not exist and must be created. A value of 0 instructs dced to forgo credential file cleanup completely.

You can modify the grace period interval from the Windows 95 registry using the following procedure:

1. Run **Regedit**.
2. Open HKEY_LOCAL_MACHINE.
3. Open the registry key **SOFTWARE\DCE Provider**, and retrieve the value.
4. Open the registry key **SOFTWARE\<dceprovidervalue>**.
5. Create a value pair in this key: click **Edit**, click **New**, then click **String Value.**
6. Set the value name to **credcleangrace**.
7. Double-click the **credcleangrace** value entry to open the **Edit** dialog box.
8. Enter the number of days for the grace period in the **Value Data** window and click OK.

Chapter 10. MVS DCE Load Balancing Client Support

DCE for Windows NT includes support for load balancing in an MVS parallel sysplex environment. This support allows the automatic rerouting of remote procedure calls (rpc) from one host to another for load balancing. Server support is not available at this time.

With this support, DCE client requests are routed to servers based on the capacity and loads on the MVS parallel sysplex. DCE load balancing enables customers to better use system resources in a parallel sysplex environment on MVS, when that support becomes available in MVS DCE.

For more information, see:

"How Load Balancing Works"

"The rpc_enable_ep_resolve_v4 Environment Variable" on page 90

How Load Balancing Works

With load balancing, a server host endpoint map directs the first request of a series of rpc calls, for example a client transaction, to the best host to handle it.

In general, when a DCE application client wants to contact a server, it obtains a partial binding from the Directory Service. This partial binding contains a host address for the application server, but no endpoint for the specific server on that host.

Currently, when this partial binding is used, RPC contacts the endpoint mapper at the host address contained in the binding. The endpoint mapper determines the endpoint for a matching application server on the same host.When this support is available in MVS DCE, the endpoint mapper can select a matching application server on another MVS host within the parallel sysplex.

Only a call on a partial DCE binding is routed this way. All subsequent calls using this binding are directed to the same server. This new support requires a change to the RPC interface supporting the **rpc_ep_resolve_binding** call. The previous Version, called Version 3, adds an endpoint to the host address which is already in the binding. The new Version, Version 4, returns both a host address and an endpoint, and the host address can be different from the original host address. When a client with Version 4 support contacts a Version 3 server, the client determines that the server does not support Version 4 and uses Version 3. A Version 4 server does not support both Version 3 and Version 4 requests.

For more information, see:

"The rpc_enable_ep_resolve_v4 Environment Variable" on page 90

The rpc_enable_ep_resolve_v4 Environment Variable

Although the DCE NT client contains support for the new Version 4 interface, the default is to use Version 3, because the DCE NT server does not support the Version 4 interface. When MVS support is available, the Version 4 client support can be invoked by setting the environment variable **rpc_enable_ep_resolve_v4** in all machine processes. Setting the environment variable to ″yes″ makes the intent clear; however, any value can be used.

Chapter 11. Simple Network Management Protocol (SNMP)

The Simple Network Management Protocol (SNMP) provides network management support in the TCP/IP environment for monitoring DCE resources and services. System administrators and system management application programmers can use SNMP to easily monitor the DCE environment so that they can focus on making their resources and services more manageable. An SNMP network management system consists of the following:

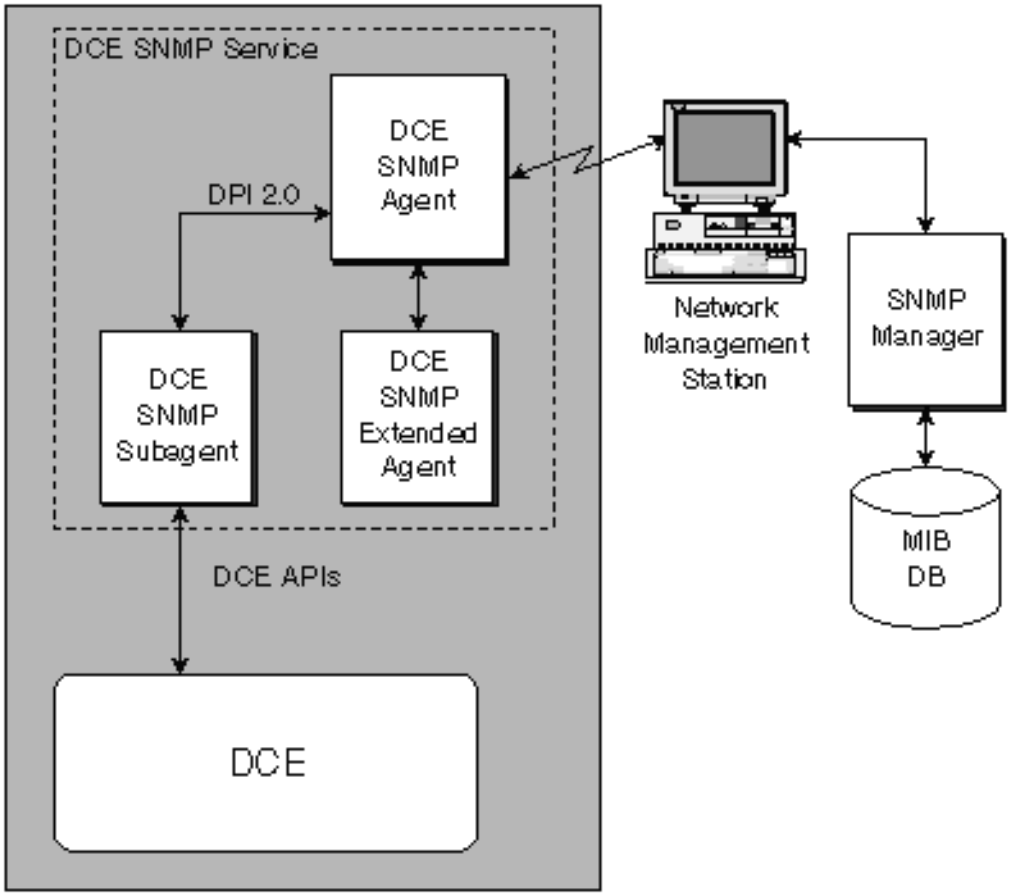
- One or more network elements (nodes), each containing an SNMP agent
- One or more Network Management Stations (NMS) containing an SNMP manager
- A network management protocol

SNMP uses a request and response message exchange model to monitor resources. The information about the monitored resources is defined in a standard format and is stored in the SNMP Management Information Base (MIB). The SNMP manager uses management operations to access the information about the monitored resources located in the MIB. The management operations are **get**, **getnext**, and **set**.

The DCE SNMP support consists of the following components:

- DCE SNMP Service
  - DCE SNMP Agent
  - DCE SNMP Extended Agent
  - DCE SNMP Subagent
- DCE SNMP MIB

The following figure shows the relationship among the SNMP Network Management System Components.



DCE SNMP Service

The DCE SNMP Service consists of the following three processes (or daemons):

- DCE SNMP Agent (snmpd.exe)
- DCE SNMP Extended Agent (extagent.exe).
- DCE SNMP Subagent (dceagtd.exe)

The three processes enable all of the SNMP support available on the local host. The subagent provides the DCE MIB data while the extended agent enables the system MIB data that is available to the NT SNMP service. The DCE SNMP agent provides the communications link for these processes and the network SNMP manager applications that can be use to manage a MIB database.

The DCE SNMP service is not compatible with the NT SNMP service. The DCE SNMP service will not start if the NT SNMP service is running. Undesirable results can occur if you start the NT SNMP service after starting the DCE SNMP service.

DCE SNMP Agent

The SNMP agent is responsible for performing the network management functions that are requested by the SNMP manager. The SNMP manager runs on a Network Management Station (NMS) and communicates with the SNMP agent on each node individually. The SNMP manager monitors and controls the network elements that have an SNMP agent.

The SNMP agent handles the requests from the SNMP manager and the responses to the management application. The communication between the agent and subagents are handled by the Distributed Protocol Interface (DPI) Version 2.0. The DPI is an extension of the SNMP agent that enables the system administrator to dynamically add, change, and remove variables from the local MIB without recompiling the SNMP agent. Any request received by the SNMP agent for a registered DCE variable is passed through the DPI to the SNMP subagent. The SNMP subagent performs the request and returns a response to the SNMP agent.

The SNMP agent supports the standard SNMP Protocol Data Unit (PDU) of **get**, **getnext**, and **set** operations as well as event notifications known as traps. The SNMP agent also encodes and decodes the PDU. Traps are unsolicited messages that the agent sends. Managed objects are passed in ASN.1 notation.

You can find the ASN.1 notation for the DCE MIB variables in the %dceloc%\declocal\etc\mib2.tbl file. %dceloc% represents the location where DCE was installed.

The agent is part of the IBM SystemView Agent Toolkit for Windows NT. For your convenience, it is packaged with DCE.

DCE SNMP Extended Agent

The DCE SNMP extended agent enables the system MIBs that are available through the NT SNMP Service. Thereby, enabling you to see all the MIB data that is available through the NT SNMP Service for the local host.

The extended agent is part of the IBM System View Agent Toolkit for Windows NT. For your convenience it is packaged with DCE for Windows NT.

DCE SNMP Subagent

The SNMP manager communicates with the SNMP agent on each managed DCE host. The SNMP agent uses an agent-to-subagent protocol (DPI 2.0) to communicate with the DCE SNMP subagent. The DCE SNMP subagent invokes the DCE Application Program Interfaces (APIs) to access information about DCE resources or services and returns the results to the SNMP agent. The subagent also provides the following capabilities:

- “Monitoring DCE EMS Events” on page 94
- “Monitoring DCE Serviceability Messages” on page 94
- “Monitoring Configured DCE Server Status Changes” on page 94

Monitoring Configured DCE Server Status Changes

The subagent heartbeat monitor periodically checks the status of the configured DCE servers and reports any status change by generating an SNMP trap for that condition. The status can change to one of the following:

Available
The server is configured and is running.

Not running
The server is configured but has been stopped.

Not configured
The server was unconfigured since the last check interval.

To change the polling interval type:

```
wsnmp -c dcesnmp set 1.3.22.1.7.1.3.0 integer x
```

where x is a numeric value measured in minutes. Entering a 0 (zero) stops the subagent heartbeat monitor. For more information see, WSNMP.

Monitoring DCE EMS Events

If EMS is configured and running, the DCE SNMP subagent registers with the DCE EMS as an event consumer to receive DCE event notifications. You can use the SNMPTRAP.TBL file (located in the %dceloc%\DCELOCAL\ETC directory) to filter the events you want the subagent to receive. After the SNMP subagent receives these events, it converts them into DPI traps and sends them to the SNMP agent where they are converted to SNMP traps. The SNMP agent routes the traps to the manager application.

You must also set up the appropriate SVC routing file information. You can use the following routing file entry to route irrecoverable error messages to EMS:

```
fatal:ems:-;
```

Note: The above example does not enable DCE core server messages.

Because the subagent creates EMS filters for the items specified in the SNMPTRAP.TBL file, EMS must be active. The subagent waits for EMS to start before attempting to create any filters. If you need to change the SNMPTRAP.TBL file while EMS is running, you must stop the subagent, make your changes, and restart the subagent. If EMS is not running, you can make your changes and start the subagent, or you can configure EMS.

Monitoring DCE Serviceability Messages

The DCE SNMP subagent monitors changes to the BIN.LOG file (located in the %dceloc%\DCELOCAL\VAR\SVC directory). To enable the SNMP subagent to monitor for changes to the BIN.LOG file, add a BINFILE entry to the serviceability routing file for each severity you want logged. Use the SNMPTRAP.TBL file (located in the %dceloc%\DCELOCAL\ETC directory) to filter the events you want the subagent to receive.

Use the following routing file entry to route irrecoverable error messages to the SVC binary log file:

```
fatal:binfile:%dceloc%\dcelocal\var\svc\bin.log;
```

Note: This method enables any DCE application that is also DCE Serviceability–enabled (for example, issues messages using dce_svc_printf). If you use both mechanisms, you will receive duplicate application server messages.

Because the log file grows, you must maintain the BIN.LOG file. You must stop DCE and the subagent BIN.LOG processing before you can make changes to the BIN.LOG file.

If you want to change the SNMPTRAP.TBl file, you can change it at any time. The changes will be picked-up on the next pass at checking the BIN.LOG file for changes.

Using the SNMPTRAP.TBL File

The SNMPTRAP.TBL file contains information on the DCE messages that you want to monitor. The file contains the following elements:

- One of the following strings: ALL_FATAL, ALL_ERROR, ALL_WARNING, ALL_NOTICE, and ALL_NOTICE_VERBOSE. You can use one or more of these to indicate that you want to monitor all messages of that severity. Any information following one of these strings on the same line is ignored. Leading white space is also ignored.
- A message index specified as 0xhhhhhhhh. The entry should contain 10 characters. The first two characters should be 0x and the remaining 8 characters can be a combination of 0–9 and a–f. The value is not checked for characters that are not valid. Verify that you have entered the correct message ID.

Any line that does not begin with one of the previous items (excluding the preceeding white space) is ignored.

If the SNMPTRAP.TBL file does not exist, a default of ALL_FATAL and ALL_ERROR is used. Otherwise, all message filtering is described in the file. If the file does not contain any filtering, no messages are enabled and no traps are generated. This does not affect the traps generated by the heartbeat monitor.

Techniques for Managing DCE

SNMP does not guarantee delivery of messages; therefore, some messages might get lost. For example, message can get lost during network congestion. To avoid this problem, SNMP allows the SNMP manager to do the following:

Polling
The management applications query the status of managed resources periodically. SNMP implicitly monitors the state of the network by polling for appropriate information on the part of the SNMP manager. Traps guide the timing and focus of the polling.

Management by Exception
SNMP detects events by receiving traps and performing trap-directed polling.

Management by Delegation
SNMP delegates a proxy, agent, or subagent to perform status polling locally.

Working With the DCE SNMP Service

After you install the DCE SNMP support you can configure, start, and stop the DCE SNMP Service. In this section we discuss the following:

- Starting and stopping DCE
- Starting and stopping the DCE SNMP service
- Using the commands

Starting and Stopping DCE

The **start.dce** command starts all DCE that are configured and not already started.

To start the DCE, on the command line type:

```
dcecp> start.dce
```

If the subagent is configured, it is started before any of the other configured deamons.

To stop all of DCE, on the command line type:

```
dcecp> stop.dce
```

If the DCE SNMP Service is configured, it is stopped after the rest of DCE is stopped.

Starting and Stopping the DCE SNMP Service

To start the DCE SNMP Service, do one of the following:

- The **start.dce snmp_srv** command starts the DCE SNMP Service through DCE if it is configured. On the command line type:

  ```
  dcecp> start.dce snmp_srv
  ```

- The **start.dce** command starts all of DCE that is configured and is not already started. On the command line type:

  ```
  dcecp> start.dce
  ```

- Use the Control Panel Services window to start the DCE SNMP Service. To display the Control Panel Services window, click the **Start** button, point to **Setting**, and then click **Control Panel**. In Control Panel, double-click the **Services** icon.

To stop the DCE SNMP Service, do one of the following:

- The **stop.dce snmp_srv** command stops the DCE SNMP Service only if it was configured through DCE. On the command line type:

  ```
  dcecp> stop.dce snmp_srv
  ```

- The **stop.dce** command stops all of the configured servers that are running with a stopped DCE SNMP Service, only if they were configured through DCE. On the command line type:

  ```
  dcecp> stop.dce
  ```

- Use the Control Panel Services window to stop the DCE SNMP Service. To display the Control Panel Services window, click the **Start** button, point to **Setting**, and then click **Control Panel**. In Control Panel, double-click the **Services** icon.

Using the Commands

dceagtd is the DCE SNMP subagent and is a deamon or server in the same sense as any of the other DCE deamon executables, except that the subagent does not require DCE to be active or configured.

Use this command if you need additional debug information. However, starting the subagent from the command line does not enable all the aspects of the DCE SNMP Service. For example, if you log out, the subagent stops soon after you log out.

DCEAGTD
: This is the DCE SNMP subagent and is a deamon or server in the same sense as any of the other DCE daemon executables, except that the subagent does not require DCE to be active or configured.

The following commands were derived from the IBM SystemView Agent Toolkit for Windows NT. You can find this package at:

```
http://www.networking.ibm.com/sha/shawin.html
```

SNMPCFG
: This tool provides the Win32 platform a graphical user interface through which you can configure community names, trap destinations, and some system information. You can get additional information by selecting Help on the window that is displayed.

  DCE SNMP provides public (read-only) and dcesnmp (read-write) community names for SNMPv1; these are the only ones that are needed, but you can add more. You can also specify SNMPv1 trap destinations for the community names. If during configuration the public community was added, a loopback trap destination was also created. If the public community name was already set, the loopback trap destination is not set. If the *dcesnmp* community name is not specified at configuration, it is added. You can also specify the system information. This information is stored in the NT registry.

WSNMP
: This tool can be used to query SNMP data from any SNMP agent in the network. Besides providing a means of querying SNMP data, it also provides a way to set the DCE SNMP polling intervals while the DCE SNMP subagent is running. It is not meant to replace your SNMP manager, but is provided as a convenient way to do this without having to physically return to the machine where the SNMP manager is running.

WTRAPD
: This tool captures SNMP trap data sent to the DCE host. From the Windows NT machine, you can use the SNMPCFG command to specify trap destinations of loopback (127.0.0.1) IP address, the local IP address, or a remote IP address.

  Note: You can use WTRAPD with the DCE SNMP Service, but make sure you do not intermix it with any of the NT SNMP Service.

SNMPD
: This is the SNMP agent. The DCE SNMP Service starts SNMPD during its initialization (if it is not already running). When you stop DCE SNMP Service, it will stop SNMPD. When you stop SNMPD it causes EXTAGENT and the subagent to be stopped.

EXTAGENT

This is the extended agent. It provides the NT SNMP MIB support available through the NT SNMP Service. The DCE SNMP Service starts EXTAGENT during its initialization (if it is not already running). When you stop the DCE SNMP Service, it stops SNMPD. If you stop EXTAGENT then you will no longer have access to the MIB II support provided by the extended agent.

dceagtd

Purpose

This is the DCE SNMP subagent and is a deamon or server in the same sense as any of the other DCE deamon executables, except that the subagent does not require DCE to be active or configured.

Format

DCEAGTD [-h *hostname* **] [-c** *community***] [-d** *debuglevel***] [-p** *heartbeat_poll_interval***] [-l** *bin_log_poll_interval***] [?]**

Options

-h *hostname*

This is the name of the host that sends requests. The default name is the name of the local host.

-c *community*

This option is used to specify the community name. The default community name is DCESNMP.

-d *debuglevel*

This option specifies the debug level. The valid values are 0–9. If you enter a value of 1–9, additional SNMPD debug information is displayed. If you enter a value of 0, the debug information is not displayed. However, the zero value does cause the time intervals to be shortened in order of magnitude. For example, **-d 0 -p 60**, changes the heartbeat poll interval from 60 minutes to 60 seconds.

-p *heartbeat_poll_interval*

This option specifies the subagent heartbeat poll interval in minutes. The default time is 60 minutes. If you enter a value of 0 value, then all subagent heartbeat polling is disabled. However, you can change it by using the SNMP set function on the MIB variable, *aSubagtHeartbeatInterval*. For example, the following changes the DCE configured server status heartbeat poll interval to 2 hours:

```
wsnmp -c dcesnmp set 1.3.22.1.7.1.3.3.0 integer 120
```

This pollarization is used to determine how often to check the configured servers status for changes since the last check. Any changes in status results in an SNMP trap being sent back to the SNMP manager and to the Wtrapd window, if applicable.

-l *bin_log_poll_interval*

This option specifies the subagent BIN.LOG interval in minutes. The default is 60 minutes. A 0 value disables all subagent BIN.LOG polling. However, this can be changed by using the SNMP set function on the MIB variable,*aSubagtLogPollInterval* . For example, the following changes the DCE subagent BIN.LOG poll interval to 24 hours.

```
wsnmp -c dcesnmp set 1.3.22.1.7.1.3.4.0 integer 1440
```

? Shows the command syntax.

If you do not specify the parameters on the command line, the DCE SNMP subagent uses the default values.

wsnmp

Purpose

Use this tool to query SNMP data from local or remote machines or to set the DCE SNMP polling intervals while the DCE SNMP subagent is running. It is included as part of DCE for Windows NT for your convenience.

Format

wsnmp [-d [*level* **]] [-h** *dest***] [-p** *port***] [-c** *community***][-t** *timeout***][-r** *retries***][-n** *non_rep***][-m** *max_rep***]** function variable [[type] [value]][...]

Options

-d *level*

This option specifies the debug level. The default level is 1. The valid values are 0 – 9. If you specify 0, it is the same as not specifying this option. If you specify debug level 2, you receive more details than debug level 1. Increasing the debug level increase the details displayed.

-h *dest*

This option specifies the destination argument. It enables you to send a request to the specified hostname or IP address. If you do not specify this option, the default is the local node. The actual command usage refers to an entry in the SNMPV2.CONF file, but does not apply to the DCE support provided. You can use this option if you want to view the DCE SNMP MIB variables on another machine.

-p *port* This option specifies the port you want to use for sending and receiving messages. If you do not specify a port, it defaults to port 161.

-c *community*

This option specifies the name of the community. Normally, you do not need to specify the community name if you define the *public* (read-only) community. The *dcesnmp* community is read-write and you must specify it on set operations for the heartbeat poll interval and the BIN.LOG poll interval. The heartbeat poll interval object identifier (OID) is 1.3.22.1.7.1.3.3.0 and the type is INTEGER. The BIN.LOG poll interval OID is 1.3.22.1.7.3.4.0 and the type is INTEGER.

Community names are case-sensitive.

-t *timeout*

This option specifies the timeout value in seconds. The minimum value is 1. The suggested maximum is 60. The default is 3 seconds. When you set the debug level to greater than 0, the timeout value doubles.

-r *retries*

This option specifies the number of retries. The minimum value is 0. The suggested maximum is 10. The default is value is 0.

-n *non_rep*

This option specifies the number of non_repeaters for the GETBULK function.

Note: Do not use this option.

-m *max_rep*

This option specifies the maximum number of repetitions for the GETBULK function.

Note: Do not use this option.

function

This option specifies the SNMP functions you want to perform. The functions are **get**, **getnext** ,**getbulk**, **set**, **walk**, and **bulkwalk**.

To use **getbulk** or **bulkwalk**, you must configure the target hostname**(-d dest)** in the SNMPV2.CONF file as a SNMPv2c target.

Note: For DCE, do not use **getbulk** or **bulkwalk**.

variable

This option specifies the object identifier (OID). The MIB variable names and their corresponding OIDs are shown in the MIB2.TBL file (located in the \opt\dcelocal\etc directory). However, the **wsnmp** command does not check the MIB2.TBL file. You can find a more extensive definition of the MIB variable in the DCE.MIB file in the same sub-directory. To make the functions run correctly, do the following:

GET To make the GET function run correctly append **.0** to the variable OID shown in the MIB2.TBL file. If the variable is part of a table, append **.row** (where row=1 for the first row or table array) to the table OID.

GETNEXT
This function provides an easier way to obtain variable length table data. Typing the prefix of any valid OID returns the value for the next variable after that sequence.

WALK This function works like the GETNEXT function, except that it retrieves all of the MIB variable with the specified OID prefix. If you have a doubt as to the OID to use on a GET request, use this function first to display the OID information.

SET The OID for this function is specified the same way as the GET OID.

type This option specifies the type of data you are sending. Type is only valid for the SET function. DCE SNMP only supports a type of INTEGER.

value This option specifies the new value for the variable. Value is only valid for the SET function. DCE SNMP supports only the following two variables:

```
1.3.22.1.7.1.3.3.0 (the heartbeat polling interval)
1.3.22.1.7.1.3.4.0 (the BIN.LOG polling interval)
```

You must also specify the *dcesnmp* community name. The commands are:

```
wsnmp -c dcesnmp set 1.3.22.1.7.1.3.3.0 integer 120
wsnmp -c dcesnmp set 1.3.22.1.7.1.3.4.0 integer 120
```

wtrapd

Purpose

Use this tool to capture SNMP trap data sent to the DCE host. **Wtrapd** uses the WinSNMP API to capture the traps; therefore, if you run on a system where DLLs are used, then you must have access to the WSNMPDLL.DLL library.

Notes:

1. You must first do the necessary setup at the IP address to have the traps sent to the correct IP address. Specifying the **-hd** option alone does not cause traps to display. Each platform has its own procedure for setting trap destinations. For DCE SNMP, you can use **SNMPCFG** to set the trap destinations.
2. If you do not specify **-hd** and **-hs**, you see all the traps generated from those nodes where this node has been set up as a trap destination.
3. For DCE SNMP on Win32 platforms, you can use **snmpcfg** to set up the trap destination.

Format

wtrapd [-d[*level***]] [-hd** *addr***] [-hs** *addr***] [-p** *port***] [-c** *community***] [-n** *oid***]**

Options

-d *level*
This option specifies the debug level. The default level is 1. The usable values are 1 – 9. If you specify 0, it is the same as not specifying this option. If you specify debug level 2, you receive more details than debug level 1. Increasing the debug level increase the details displayed.

-hd *addr*
This option specifies the IP address of the location where you want to receive traps from. After you specify an IP address, traps sent from other IP addresses are ignored. If you do not specify this option, you will receive all traps sent to the local host.

-hs *addr*
This option specifies the receiving IP address that you want to use. On a multi-homed system, traps are listened for from all IP addresses at port 162. If you specify an IP address using this option, only traps on port 162 for that address are displayed. If you do not specify this option, the default is any local address.

-p *port* This option specifies the port you want to use for listening to notifications. If you do not specify a port, the default is 162.

-c *community*
This option specifies the name of the community. Only traps with the community name that you specify are displayed. If you do not specify a community name, the default is to allow all traps to display.

-n *oid* This option specifies the OID prefix that you want to use to listen for notifications. If you do not specify this option, all prefixes are used.

snmpd

Purpose

This is the SNMP agent. It supports the following three versions of SNMP:

- SNMPv1 (as defined in RFC1157)
- SNMPv2c (as defined in RFC1901–1908)
- SNMPv2u (as defined in RFC1902–1910)

In addition to the above versions, it supports the DPI 2.0 protocol (RFC1592) in support of subagents that implement MIBs in a separate process. For example, it supports the DCE SNMP subagent.

Note: The DCE SNMP Service starts the SNMP agent (if it is not already running) with the correct transport and dpi options. You should let the DCE SNMP Service start the agent so that it will continue to run if you log off. Stopping the agent causes the extended agent and subagent to stop if either is running.

Format

snmpd [-d [*level***]]** *-transport udp***[***=port***]****[***-dpi tcp***]****[-dpi** *shm***]**

Options

-d *level*

This option specifies the debug level. If you only specify **-d**, level 31 is used as the default . The range for this option is 0 – 255. If you specify 0, it is the same as not specifying this option. The following are the debug levels:

```
  1=  Incoming SNMP requests
  2=  Outging SNMP responses
  4=  Outgoing SNMP traps
  8=  DPIdebug at level 1
 16=  DPIdebug at level 2
 32=  Internal trace lvl-1
 64=  Internal trace lvl-2
128=  Internal trace lvl-3
```

You can combine debug levels by adding them together.

-transport udp [=port]

This option specifies the SNMP transport. For DCE, you must specify *-transport udp* or *-transport udp=161*.

-dpi tcp

This option specifies the DPI transport. For DCE, you must specify tcp as the dpi transport.

Note: DCE SNMP does not support *-dpi tcp*.

-dpi shm

This option specifies that the connection you want to use is shared memory, instead of TCP.

Note: DCE SNMP does not support *-dpi shm*.

extagent

Purpose

This is the IBM extended agent that provides the runtime support for the Microsoft extensible agent API. It provides the support by mapping the API onto DPI version 2.0 (RCF1592), so both DPI based subagents (like the DCE subagent) and existing Win32–based subagents can be active at the same time.

Extagent also enables the Microsoft SNMP instrumentation (like MIB II support) to be activated and accessible through the IBM provided system ViewAgent.

The DCE SNMP subagent starts EXTAGENT during the subagent startup initialization (if it is not already running). If the SNMP agent is not running, both the extended agent and DCE subagent have stopped.

Format

extagent [-d[*level***]] [-h** *hostname***][-c** *community***][-shm]**

Options

-d *level*

This option specifies the debug level. If you specify only **-d**, the default level is 1. The minimum value is zero. A reasonable maximum is 255. If you specify 0, it is the same as not specifying this option. If you specify debug level 2, you receive more details than debug level 1. Increasing the debug level increases the details displayed.

-h *hostname*

This option specifies the hostname that is used to DPI-connect. If you do specify this option, the default is *loopback* (127.0.0.1). If loopback is not defined as the hostname for the loopback address, you must specify your hostname.

If your system does not have a definition for *loopback* then it will not connect. To make it work correctly, use *-h yourhostname*. Some systems do not have a *loopback* address (127.0.0.1).

-c *community*

This option specifies the name of the community. If you do not specify this option, the default community is *public*. If you did not configure or you removed the community name *public* from your configuration, you must specify a community name that does provide read-access to the MIB data.

-shm This option specifies the connection that you want to use as shared memory, instead of TCP. To make this option work, you must first start SNMPD with the **-dpi shm** option.

DCE SNMP Management Information Base

Simple Network Management Protocol (SNMP) is a TCP/IP network management protocol and is based on a manager-agent interaction. The SNMP manager (such as NetView for OS/2) communicates with its agents. Agents gather management data (such as physical and logical characteristics of network objects) and store it, while managers solicit this data and process it. This collection of management information is called a management information base (MIB). The DCE SNMP subagent implements the DCE MIB for the agent and accesses the DCE data upon request. The individual pieces of information that comprise a MIB are called MIB objects and they reside on the agent system. The subagent can generate traps from DCE Serviceability (SVC) messages and communicates the traps to the agent asynchronously. With the exception of traps, MIB objects can be accessed by the agent at the manager's request.

Note: Even though DCE provides location transparency, which means the views can be cell-wide as opposed to node-wide, the DCE SNMP MIB is defined on a per-node basis due to the nature of the SNMP model.

The DCE SNMP MIB consists of the following groups:

Component ID Group
This group provides base-level identificaton of the component, which in this case is DCE. For example, it indicates when the product that was installed, the version, and the serial number.

Software Component Information Group
This group extends the Component ID group with attributes that further define DCE. This relates the operating system that the DCE is installed on and the language that is used in its interface.

DCE Subagent Group
This group contains information about the DCE SNMP subagent. For example, when it was started, its name and, polling intervals.

DCE Host Information Group
This group contains the basic information about the DCE host. For example, its name and which DCE cell it is in.

DCE Server Table Group
This group is a table of DCE server entries. Each entry contains the server by its name and its current state: Available or Not running.

DCE Host Server Group
This group contains information about the host daemon on this DCE host. For example, the process ID, group ID, user ID, server state, and a set of RPC statistics. The remote procedure call (RPC) statistics contain the number of RPC calls sent and received by this server and the number of network RPC packets sent and received by this server.

DCE Event Management Service Server Group
This group contains information about the Event Management Service (EMS) server on the host. It contains the same type of information (IDs, server state, and RPC statistics) provided by the DCE Host Server Group.

DCE Security Server Group
This group contains information about the Security server on the DCE host. This group contains the same type of information (IDs, server name, and RPC statistics) provided by the DCE Host Server Group, and indicates the

server role (master or replica), mode (service or maintenance), and some replica data (last updated and last sequence number).

DCE Cell Directory Service Server Group
This group contains the same type of information (IDs, server name, and RPC statistics) provided by the DCE Host Server Group as well as the up time for this server, error statistics, and the number of read and write operations.

DCE Cell Directory Service Advertiser Group
This group contains the same type of information (IDs, server name, and RPC statistics) provided by the DCE Host Server Group as well as the up time for the CDS Advertiser on the DCE host.

DCE Cell Directory Service Clerks Group
This group contains information about operations performed by the CDS clerks on the DCE host. This includes some error data and operation counts (read, writes, and miscellaneous).

DCE Cell Directory Service Clerk Table Group
This group contains the information about the CDS clerks on the DCE host. Each entry contains the clerk name and up time.

DCE Cell Directory Service Clearinghouse Table Group
This group contains the information about the CDS clearinghouses on the host. Each entry contains full clearinghouse name, some error data, and the number of reads and writes.

DCE Cell Directory Service Cached Clearinghouse Table Group
This group contains a table of entries about the cached clearinghouses on the host. Each entry contains the cached clearinghose name, up time, the number of operations, (read, writes, and miscellaneous).

DCE Global Directory Agent Server Group
This group contains information about the GDA server on the host. It contains the same type of information provided by the DCE Host Server Group (IDs, server state, and the RPC statistics).

DCE Distributed Time Service Entity Group
This group contains information about the DTS entity on the host. It contains the same type of information provided by the DCE Host Server Group (IDs, server state, and the RPC statistics). It also contains the current time on the DTS node, some error data, some time values (for example, time to wait for a response before taking an action).

DCE Distributed Time Service Server Group
This group contains information about the local or global DCE DTS server on this host. It has the following attributes: role (local, global, or clerk), courier role, and some error statistics.

DCE Distributed Time Service Known Server Table Group
This group contains the information of a list of local or global DTS servers known by this DTS entity. Each entry contains the name of the known DTS server by this host, the time that it was last polled, the last observed time difference, the time that it was last synchronized, the time that it was last observed, and its transport protocol (for example, RPC).

DCE Traps Group
This group contains the network events sent from the DCE SNMP subagent

to an SNMP agent. The information generated and sent out from the DCE Traps group cannot be queried. The information generated by all of the other groups can be queried.

DCE Security Audit Server Group
This group contains informtion about the Security Audit server on the DCE host. It contains the same type of information provided by the DCE Host Server Group (IDs, server state, and the RPC statistics).

DCE Password Strength Server Table Group
This group contains a table entry for each Password Strength server that is configured. Each entry contains the name of the server. It also contains the same type of information provided by the DCE Host Server Group (IDs, server state, and the RPC statistics).

Note: The comments in the DCE.MIB file (located in the %dceloc%\declocal\etc subdirectory) contain additional information. All MIB comments are preceeded by the double dash (**- -**) character string.

DCE MIB Definitions

The DCE.MIB file is located in the %dceloc%\dcelocal\etc directory. You can use it to describe the MIB to the SNMP manager applications such as NetView/6000 or NetView for OS/2. Refer to the manager application documentation to learn how to load this MIB for use by the application. The MIB2.TBL file, located in the same directory, corelates the ASN.1 notation to the name for the variable in the DCE MIB. There is enough information in the DCE.MIB file to help you determine the ASN.1 notation, but the MIB2.TBL file documents it concisely. For example, the first object in the DCE.MIB file is “aManufacturer;” in the MIB2.TBL file it is identified as 1.3.22.1.7.1.1.1.

```
IMPORTS
   Counter, enterprises
           FROM RFC1155-SMI
   OBJECT-TYPE
           FROM RFC-1212
   DisplayString
           FROM RFC1213-MIB
   TRAP-TYPE
           FROM RFC-1215;

   osf                     OBJECT IDENTIFIER ::= { iso org(3) 22 }
   dce                     OBJECT IDENTIFIER ::= { osf 1 }
   dcesnmp                 OBJECT IDENTIFIER ::= { dce 7 }
   dcemib                  OBJECT IDENTIFIER ::= { dcesnmp 1 }
```

Notes:

1. The subagent is implemented as an NT Service. You must configure it. You can start and stop it using DCE commands. You can start it from the command line (but it will not have all the attributes of a true NT Service). The DCE SNMP Service is mutually exclusive with the NT SNMP Service.

   The DCE SNMP Service starts the SNMP agent and the Extended Agent (if they are not already started). The SNMP Extended Agent (*extagent.exe*) supports other NT MIBs defined for the NT SNMP Service and enables them. The SNMP agent (*snmpd.exe*) must be running for the subagent and extended agent to continue running.
2. The MIB data is cached in the subagent for performance reasons. This is primarily because access to a single MIB group value will in most cases cause

the entire group to be updated. Therefore, any access by group (walk or dump) can be made faster by using the cached data.

The cached data is also used when the server is stopped and the subagent is still running. This allows information such as the security server role to be accessible. Therefore, once a server is stopped, its cached data will continue to be seen until a new set can be obtained (after the server is restarted). If the server is not started, the cache numerics will be zero and the cache strings will be ″Unavailable″ until the server is restarted.

Component ID Group

```
componentIDGroup OBJECT IDENTIFIER ::= {dcemib 1}

  aManufacturer OBJECT-TYPE
      SYNTAX            DisplayString (SIZE (0..32))
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The company that produced this component."
       --               The string is: "IBM Corporation".
      ::= {componentIDGroup 1}

  aProduct OBJECT-TYPE
      SYNTAX            DisplayString (SIZE (0..32))
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The name of this component or product."
       --               The string is: DCE for Windows NT.
      ::= {componentIDGroup 2}

  aVersion OBJECT-TYPE
      SYNTAX            DisplayString (SIZE (0..16))
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The version string for this component."
       --               If the string cannot be determined, then "Unavailable"
       --               is displayed.
             ::= {componentIDGroup 3}

  aSerialNumber OBJECT-TYPE
      SYNTAX            DisplayString (SIZE (0..16))
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The serial number for this component."
       --               If the string cannot be determined, then "Unavailable"
       --               is displayed.
              ::= {componentIDGroup 4}

  aInstallation OBJECT-TYPE
      SYNTAX            DisplayString (SIZE (0..28))
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "A date and time indication of the last install."
       --               If the string cannot be determined, then "Unavailable"
       --               is displayed.
              ::= {componentIDGroup 5}

  aVerify OBJECT-TYPE
      SYNTAX            INTEGER
   -- {
       -- vAnErrorOccurred;CheckStatusCode       (0),
       -- vThisComponentDoesNotExist             (1),
       -- vTheVerifyIsNotSupported               (2),
       -- vReserved                              (3),
       -- vComponent'sFunctionalityUntested      (4),
       -- vComponent'sFunctionalityUnknown       (5),
       -- vComponentIsNotFunctioningCorrectly    (6),
```

```
     -- vComponentFunctionsCorrectly          (7)
 -- }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "A code that provides a level of verification
                      that the component is still installed and
                      working.  This value is 2 for this release."
    ::= {componentIDGroup 6}

aVerifyString OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "A string that corresponds to the aVerify
                      value.  The string for this release will
                      be:  Verify is not supported."
    ::= {componentIDGroup 7}
```

Software Component Information Group

```
softwareCompInfoGroup OBJECT IDENTIFIER ::= {dcemib 2}

aMajorVersion OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..12))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "Major version of this software component."
     --               If the string cannot be determined, then "Unavailable"
     --               is displayed.
     ::= {softwareCompInfoGroup 1}

aMinorVersion OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..12))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "Minor version of this software component."
     --               If the string cannot be determined, then "Unavailable"
     --               is displayed.
    ::= {softwareCompInfoGroup 2}

aRevision OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..12))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "Revision of this software component."
     --               If the string cannot be determined, then "Unavailable"
     --               is displayed.
    ::= {softwareCompInfoGroup 3}

aBuild OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..12))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "Manufacturer's internal identifier for this
                      compilation."
     --               If the string cannot be determined, then "Unavailable"
     --               is displayed.
    ::= {softwareCompInfoGroup 4}

aTargetOperatingSystem OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The operating system which this software
                      component is intended for.
     --               The value is 6 for this release (6=Win32)."
    ::= {softwareCompInfoGroup 5}
```

```
aLanguageEdition OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The language edition of this software component.
                      This string will be: English."
    ::= {softwareCompInfoGroup 6}

aIdentificationCode OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "Stock Keeping Unit (SKU) for this software component."
     --               If the string cannot be determined, then "Unavailable"
     --               is displayed.
     ::= {softwareCompInfoGroup 7}

aTargetOSString OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The operating system which this software
                      component is intended for.  This is Windows NT
     --               for this release."
    ::= {softwareCompInfoGroup 8}
```

DCE Subagent Group

```
dceSubagentGroup OBJECT IDENTIFIER ::= {dcemib 3}

aSubagtName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The name of this subagent is: DCE SNMP Subagent."
    ::= {dceSubagentGroup 1}

aSubagtUpTime OBJECT-TYPE
    SYNTAX            Date
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The date and time the DCE subagent was last started."
    ::= {dceSubagentGroup 2}

aSubagtHeartbeatInterval OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-write
    STATUS            mandatory
    DESCRIPTION      "This is the delay in minutes between checks for
                      changes in status for configured DCE servers.  The
                      default is 10 minutes.  A value of zero disables
                      the checking."
     --
     --               The heartbeat checking that is done will cause an SNMP trap to
     --               be generated for any changes in the status of a DCE server.
     --
     --               If a large value is used, you can lose any
     --               functional value provided by this function.
     --               If a small value is used, it could cause a
     --               performance degradation.
     --
     --               Using the dceagtd -d option when starting the subagent
     --               causes the time interval to change to seconds rather
     --               than minutes (-d 0 causes the default to be 10 seconds).
     --
     --               The wsnmp command can be used to adjust this value
     --               as follows:
```

```
     --
     --                      wsmp -c dcesnmp set 1.3.22.1.7.1.3.3.0 integer 60
     --
     --               Issuing the previous command from the command line
     --               will change the interval to 60 minutes after the
     --               current time interval expires.
    ::= {dceSubagentGroup 3}

aSubagtBinLogInterval OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-write
    STATUS            mandatory
    DESCRIPTION      "This is the delay in minutes between bin.log error
                      checks.  The default is 10 minutes.  A value of zero
                      disables binary log checking."
     --
     --               If a large value is used, you can lose any
     --               functional value provided by this function.
     --               If a small value is used, it could cause a
     --               performance degradation.
     --
     --               Using the dceagtd -d option when starting the subagent
     --               causes the time interval to change to seconds rather
     --               than minutes (-d 0 cause default to be 10 seconds).
     --
     --               System tools can be used to adjust this value
     --               as follows:
     --
     --               wsnmp -c dcesnmp set 1.3.22.1.7.1.3.4.0 integer 60
     --
     --               Issuing the previous command from the command line
     --               will change the interval to 60 minutes
    --               after the current time interval expires.
    ::= {dceSubagentGroup 4}
```

DCE Host Information Group

```
dceHostInfoGroup OBJECT IDENTIFIER ::= {dcemib 4}

aDCEHostName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..64))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The name of the DCE host."
    ::= {dceHostInfoGroup 1}

aDCEHostCellName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..64))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The cell name that this DCE host belongs to."
    ::= {dceHostInfoGroup 2}
```

DCE Server Table Group

```
dceSvrTable OBJECT-TYPE
    SYNTAX            SEQUENCE OF     DCEHostSvrEntry
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "A list of server entries configured on this
                      DCE host."
    ::= {dcemib 5}

dceHostSvrEntry OBJECT-TYPE
    SYNTAX            DCEHostSvrEntry
    ACCESS            not-accessible
    STATUS            mandatory
```

```
        DESCRIPTION      ""
--      INDEX             {aDCESvrName}
        ::= {dceSvrTable 1}

    DCEHostSvrEntry ::= SEQUENCE
      {
        aDCESvrName          DisplayString,
        aDCESvrState         DisplayString,
        aDCESvrStateValue    INTEGER
      }

    aDCESvrName OBJECT-TYPE
        SYNTAX            DisplayString (SIZE (0..128))
        ACCESS            read-only
        STATUS            mandatory
        DESCRIPTION      "The name of the DCE server."
         --               The string can be one of the following:
         --
         --                   "Not applicable" (eg, when DCE is not configured)
         --       DCED        "DCE Host Server"
         --       SECD        "Master Security Server"
         --                   "Replica Security Server"
         --                   "Security Server" (if unable to determine)
         --       CDSD        "CDS Server"
         --       CDSADV      "CDS Advertiser"
         --       DTSD        "DTS Global Server"
         --                   "DTS Local Server"
         --                   "DTS Clerk"
         --                   "DTS Server" (if unable to determine)
         --       GDAD        "GDA Server"
         --       EMSD        "EMS Server"
         --       AUDITD      "Audit Server"
         --       DFSD        "DFS Client"
         --       NSID        "NSI Gateway"
         --       (name)      "Password Strength Server (name)"
         --                    The "name" for password strength server is the
         --                    one associated with it through configuration.
         --                    There can be multiples in this category.
        ::= {dceHostSvrEntry 1}

    aDCESvrState OBJECT-TYPE
        SYNTAX            DisplayString (SIZE (0..16))
        ACCESS            read-only
        STATUS            mandatory
        DESCRIPTION      "The state of this DCE server."
         --               One of the following strings will be set
         --               based on the state value:
         --               "Unknown"           See note.
         --               "Not running"       The server is configured, but not running.
         --               "Available"         The server is configured and is running.
         --               "Not configured"    DCE server is not configured.  A server that is partia
         --                                   configured, displays as "Not running."  To
         --                                   determine the state of the configuration, run
         --                                   dcecp show.cfg.
         --
         -- NOTE: The "Unknown" state may be seen if an error is encountered
         --       while trying to obtain the state; however, it is unlikely
         --       that this will happen for any of the servers.  It is a
         --       primarily a zero place holder.
        ::= {dceHostSvrEntry 2}

    aDCESvrStateValue OBJECT-TYPE
        SYNTAX            INTEGER
        {
             vUnknown                              (0),
             vNotRunning                           (1),
             vAvailable                            (2),
```

```
          vNotConfigured                          (3)
      }
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The state value of this DCE Server."
      ::= {dceHostSvrEntry 3}
```

DCE Host Server Group

```
dceHostSvrGroup OBJECT IDENTIFIER ::= {dcemib 6}

 aHostSvrPid OBJECT-TYPE
      SYNTAX            INTEGER
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The process ID of the DCE host server."
      ::= {dceHostSvrGroup 1}

 aHostSvrUid OBJECT-TYPE
      SYNTAX            INTEGER
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The user ID of the DCE host server."
      ::= {dceHostSvrGroup 2}

 aHostSvrGid OBJECT-TYPE
      SYNTAX            INTEGER
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The group ID of the DCE host server."
       ::= {dceHostSvrGroup 3}

 aHostSvrInRpcCalls OBJECT-TYPE
      SYNTAX            Counter
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The number of RPC calls received by the server."
      ::= {dceHostSvrGroup 4}

 aHostSvrOutRpcCalls OBJECT-TYPE
      SYNTAX            Counter
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The number of RPC calls initiated by the server."
      ::= {dceHostSvrGroup 5}

 aHostSvrInRpcPkts OBJECT-TYPE
      SYNTAX            Counter
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The number of RPC packets received by the server."
      ::= {dceHostSvrGroup 6}

 aHostSvrOutRpcPkts OBJECT-TYPE
      SYNTAX            Counter
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The number of RPC packets initiated by the server."
      ::= {dceHostSvrGroup 7}

 aHostSvrState OBJECT-TYPE
      SYNTAX            DisplayString (SIZE (0..16))
      ACCESS            read-only
      STATUS            mandatory
      DESCRIPTION      "The state of this server."
       --               One of the following strings will be set
       --               based on the state value:
       --               "Unknown"          A problem was encountered while trying to resolve
```

```
     --                                  the state of the server.
     --              "Not running"       The server is configured, and is not running.
     --              "Available"         The server is configured and running.
     --              "Not configured"    DCE server is not configured.
    ::= {dceHostSvrGroup 8}

aHostSvrStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this server."
    ::= {dceHostSvrGroup 9}
```

DCE Event Management Service Server Group

```
dceEmsSvrGroup OBJECT IDENTIFIER ::= {dcemib 7}

aEmsSvrPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of the server."
    ::= {dceEmsSvrGroup 1}

aEmsSvrUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID of the server."
    ::= {dceEmsSvrGroup 2}

aEmsSvrGid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The group ID of the server."
    ::= {dceEmsSvrGroup 3}

aEmsSvrInRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls received by the server."
    ::= {dceEmsSvrGroup 4}

aEmsSvrOutRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls initiated by the server."
    ::= {dceEmsSvrGroup 5}

aEmsSvrInRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets received by the DCE EMS server."
    ::= {dceEmsSvrGroup 6}

aEmsSvrOutRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
```

```
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets initiated by the DCE EMS server."
    ::= {dceEmsSvrGroup 7}

aEmsSvrState OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state of this server."
     --               One of the following strings will be set
     --               based on the state value:
     --               "Unknown"          A problem was encountered while trying to resolve
     --                                  the state of the server.
     --               "Not running"      The server is configured, and is not running.
     --               "Available"        The server is configured and running.
     --               "Not configured"   DCE server is not configured.
    ::= {dceEmsSvrGroup 8}

aEmsSvrStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this server."
    ::= {dceEmsSvrGroup 9}
```

DCE Security Server Group

```
dceSecSvrGroup OBJECT IDENTIFIER ::= {dcemib 8}

aSecSvrRole OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                     (0),
         vMaster                      (1),
         vReplica                     (2)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The role of the server."
    ::= {dceSecSvrGroup 1}

aSecSvrMode OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                     (0),
         vService                     (1),
         vMaintenance                 (2)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The mode of the Master Security server."
    ::= {dceSecSvrGroup 2}

aSecRgyUpdTime OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The localized date and time that the replica
                      was last updated."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
```

```
     --               object does not apply.
    ::= {dceSecSvrGroup 3}

aSecRgyUpdSeq OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The sequence number of the last update that
                      the replica received."
    ::= {dceSecSvrGroup 4}

aSecSvrPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of the DCE Security server."
    ::= {dceSecSvrGroup 5}

aSecSvrUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID of the server."
    ::= {dceSecSvrGroup 6}

aSecSvrGid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The group ID of the server."
    ::= {dceSecSvrGroup 7}

aSecSvrInRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls received by the DCE Security server."
    ::= {dceSecSvrGroup 8}

aSecSvrOutRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls initiated by the server."
    ::= {dceSecSvrGroup 9}

aSecSvrInRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets received by the server."
    ::= {dceSecSvrGroup 10}

aSecSvrOutRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets initiated by the server."
    ::= {dceSecSvrGroup 11}

aSecSvrState OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state of this server."
     --               One of the following strings will be set
     --               based on the state value:
```

```
     --               "Unknown"          A problem was encountered while trying to resolve
     --                                  the state of the server.
     --               "Not running"      The server is configured, and is not running.
     --               "Available"        The server is configured and running.
     --               "Not configured"   DCE server is not configured.
    ::= {dceSecSvrGroup 12}

aSecSvrStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this DCE Security server."
    ::= {dceSecSvrGroup 13}

aSecSvrRoleString OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The role of this DCE Security server."
     --               One of the following strings will be set
     --               based on the aSecSvrRole value:
     --                      "Master"
     --                      "Replica"
     --               The string will be "Unavailable" when an error is
     --               encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceSecSvrGroup 14}

aSecSvrModeString OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The mode of this DCE Security server."
     --               One of the following strings will be set
     --               based on the aSecSvrMode value:
     --                      "Service"
     --                      "Maintenance"
     --               The string will be "Unavailable" when an error is
     --               encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceSecSvrGroup 15}
```

DCE Cell Directory Service Server Group

```
dceCdsSvrGroup OBJECT IDENTIFIER ::= {dcemib 9}

aCdsSvrUpTime OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The time this server was created."
     --               The string will be "Unavailable" when an error is
     --               encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceCdsSvrGroup 1}

aCdsSvrChildUpdFails OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
```

```
    STATUS            mandatory
    DESCRIPTION      "The number of times the server was unable to
                      contact all the clearinghouses that store a
                      replica of a particular child directory's
                      parent directory and apply the child updates
                      that have occurred since the last skulk."
    ::= {dceCdsSvrGroup 2}

aCdsSvrCrucialReps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times a user attempted to remove
                      a crucial replica from this server."
    ::= {dceCdsSvrGroup 3}

aCdsSvrMaxSkewTime OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The maximum amount of time that a timestamp on
                      a new or modified entry can vary from local
                      system time."
    ::= {dceCdsSvrGroup 4}

aCdsSvrLookupPathBrokens OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of broken connections between
                      clearinghouses on this server and
                      clearinghouses closer to the root."
    ::= {dceCdsSvrGroup 5}

aCdsSvrSecFails OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times a server principal for
                      this server was found to have inadequate
                      permissions to perform a request operation."
    ::= {dceCdsSvrGroup 6}

aCdsSvrSkulkInitd OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of skulks initiated."
    ::= {dceCdsSvrGroup 7}

aCdsSvrSkulkCmpltd OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of skulks successfully completed."
    ::= {dceCdsSvrGroup 8}

aCdsSvrReadOps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of read operations directed to this server."
    ::= {dceCdsSvrGroup 9}

aCdsSvrWriteOps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
```

```
    STATUS            mandatory
    DESCRIPTION      "The number of write operations directed to this
                      server."
    ::= {dceCdsSvrGroup 10}

aCdsSvrPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of this DCE CDS server."
    ::= {dceCdsSvrGroup 11}

aCdsSvrUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID this DCE CDS server."
    ::= {dceCdsSvrGroup 12}

aCdsSvrGid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The group ID of this server."
    ::= {dceCdsSvrGroup 13}

aCdsSvrInRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls received by the server."
    ::= {dceCdsSvrGroup 14}

aCdsSvrOutRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls initiated by the server."
    ::= {dceCdsSvrGroup 15}

aCdsSvrInRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets received by the server."
    ::= {dceCdsSvrGroup 16}

aCdsSvrOutRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets initiated by the server."
    ::= {dceCdsSvrGroup 17}

aCdsSvrState OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state of this DCE CDS server."
     --               One of the following strings will be set
     --               based on the state value:
     --               "Unknown"          A problem was encountered while trying to resolve
     --                                  the state of the server.
     --               "Not running"      The server is configured, and is not running.
     --               "Available"        The server is configured and running.
     --               "Not configured"   DCE server is not configured.
    ::= {dceCdsSvrGroup 18}
```

```
aCdsSvrStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this DCE CDS server."
    ::= {dceCdsSvrGroup 19}
```

DCE Cell Directory Service Advertiser Group

```
dceCdsAdvGroup OBJECT IDENTIFIER ::= {dcemib 10}

aCdsAdvUpTime OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The start time of the DCE CDS advertiser."
     --               The string will be "Unavailable" when an error is
     --               encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceCdsAdvGroup 1}

aCdsAdvPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of the DCE CDS advertiser."
    ::= {dceCdsAdvGroup 2}

aCdsAdvUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID of this DCE CDS advertiser."
     ::= {dceCdsAdvGroup 3}

aCdsAdvGid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The group ID of this DCE CDS advertiser."
    ::= {dceCdsAdvGroup 4}

aCdsAdvInRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls received."
    ::= {dceCdsAdvGroup 5}

aCdsAdvOutRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls initiated."
    ::= {dceCdsAdvGroup 6}

aCdsAdvInRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
```

```
    DESCRIPTION      "The number of RPC packets received."
    ::= {dceCdsAdvGroup 7}

aCdsAdvOutRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets initiated."
    ::= {dceCdsAdvGroup 8}

aCdsAdvState OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state of this DCE CDS advertiser."
     --               One of the following strings will be set
     --               based on the state value:
     --               "Unknown"          A problem was encountered while trying to resolve
     --                                  the state of the server.
     --               "Not running"      The server is configured, and is not running.
     --               "Available"        The server is configured and running.
     --               "Not configured"   DCE server is not configured.
    ::= {dceCdsAdvGroup 9}

aCdsAdvStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this DCE CDS advertiser."
    ::= {dceCdsAdvGroup 10}
```

DCE Cell Directory Service Clerks Group

Note: On Windows NT this group is not available and will return zeros for counters.

```
dceCdsClerksGroup OBJECT IDENTIFIER ::= {dcemib 11}

aCdsClerksAuthFails OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times requesting principals
                      failed authentication procedures."
    ::= {dceCdsClerksGroup 1}

aCdsClerksCacheBypasses OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of requests to read attributes
                      for which the clerk was specifically
                      directed by the requesting application to
                      bypass its own cache."
    ::= {dceCdsClerksGroup 2}

aCdsClerksCacheHits OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The total number of read requests directed
                      to this clerk that were satisfied entirely
```

```
                        by its cache.  This attribute accounts only
                        for requests to read attribute values and does
                        not include requests to look up names or
                        enumerate the contents of directories."
    ::= {dceCdsClerksGroup 3}

aCdsClerksMiscOps OBJECT-TYPE
    SYNTAX              Counter
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The number of operations other than read and
                        write performed by this clerk."
    ::= {dceCdsClerksGroup 4}

aCdsClerksReadOps OBJECT-TYPE
    SYNTAX              Counter
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The number of lookup operations performed by
                        this clerk."
    ::= {dceCdsClerksGroup 5}

aCdsClerksWriteOps OBJECT-TYPE
    SYNTAX              Counter
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The number of requests to modify data
                        processed by this clerk."
    ::= {dceCdsClerksGroup 6}
```

DCE Cell Directory Service Clerk Table Group

Note: On Windows NT this group is not available, but will contain one clerk entry showing zeroes for numerics and ″Not applicable″ for strings.

```
cdsClerkTable OBJECT-TYPE
    SYNTAX              SEQUENCE OF     CdsClerkEntry
    ACCESS              not-accessible
    STATUS              mandatory
    DESCRIPTION       "A list of DCE CDS clerks on the host."
    ::= {dcemib 12}

cdsClerkEntry OBJECT-TYPE
    SYNTAX              CdsClerkEntry
    ACCESS              not-accessible
    STATUS              mandatory
    DESCRIPTION       ""
    INDEX               {aClerkPid}
    ::= {cdsClerkTable 1}

CdsClerkEntry ::= SEQUENCE
  {
    aClerkUser          DisplayString,
    aClerkUid           INTEGER,
    aClerkPid           INTEGER,
    aClerkUpTime        OCTET STRING (SIZE (50))
  }

aClerkUser OBJECT-TYPE
    SYNTAX              DisplayString (SIZE (0..64))
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The user name of a clerk on the host."
     --  UNIX:          This contains the string, "Root".  There will
     --                 be only be one table entry possible because
     --                 any other clerks are transient and known only
     --                 to CDS.
```

```
     --  Non-UNIX:    This contains the string, "Not applicable".
    ::= {cdsClerkEntry 1}

aClerkUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID of a clerk on the host.  This
                      field is returned as zero."
    ::= {cdsClerkEntry 2}

aClerkPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of a clerk on the host."
    ::= {cdsClerkEntry 3}

aClerkUpTime OBJECT-TYPE
    SYNTAX            Date
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The start time of a clerk on the host."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {cdsClerkEntry 4}
```

DCE Cell Directory Service Clearinghouse Table Group

```
cdsCHTable OBJECT-TYPE
    SYNTAX            SEQUENCE OF     CdsCHEntry
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "A list of clearinghouses on the host."
    ::= {dcemib 13}

cdsCHEntry OBJECT-TYPE
    SYNTAX            CdsCHEntry
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      ""
    INDEX            {aChName}
    ::= {cdsCHTable 1}

CdsCHEntry ::= SEQUENCE
  {
    aChName          DisplayString,
    aChDataCorrupts  Counter,
    aChEnableCounts  Counter,
    aChDisableCounts Counter,
    aChRefReturns    Counter,
    aChSkulkFails    Counter,
    aChEntryMisses   Counter,
    aChRootLosts     Counter,
    aChUpgFails      Counter,
    aChReadOps       Counter,
    aChWriteOps      Counter
  }

aChName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..64))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The full name of the clearinghouse."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
```

```
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {cdsCHEntry 1}

aChDataCorrupts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times that the data corruption
                      event was generated."
    ::= {cdsCHEntry 2}

aChEnableCounts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times that the clearinghouse
                      was enabled since it was last started."
    ::= {cdsCHEntry 3}

aChDisableCounts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times that the clearinghouse
                      was disabled since it was last started."
    ::= {cdsCHEntry 4}

aChRefReturns OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of requests directed to this
                      clearinghouse that resulted in the return of
                      a partial answer instead of satisfying the
                      client's request."
    ::= {cdsCHEntry 5}

aChSkulkFails OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times that a skulk of a
                      directory, initiated from this clearinghouse,
                      failed to complete."
    ::= {cdsCHEntry 6}

aChEntryMisses OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the clearinghouse entry
                      missing event was generated."
    ::= {cdsCHEntry 7}

aChRootLosts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the root lost event was
                      generated."
    ::= {cdsCHEntry 8}

aChUpgFails OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
```

```
    DESCRIPTION      "The number of times that upgrades failed."
    ::= {cdsCHEntry 9}

aChReadOps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of read operations directed to
                      this clearinghouse."
    ::= {cdsCHEntry 10}

aChWriteOps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of write operations directed to
                      this clearinghouse."
    ::= {cdsCHEntry 11}
```

DCE Cell Directory Service Cached Clearinghouse Table Group

```
cdsCachedCHTable OBJECT-TYPE
    SYNTAX            SEQUENCE OF     CdsCachedCHEntry
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "A list of cached clearinghouses on the host."
    ::= {dcemib 14}

cdsCachedCHEntry OBJECT-TYPE
    SYNTAX            CdsCachedCHEntry
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      ""
    INDEX            {aCachedCHName}
    ::= {cdsCachedCHTable 1}

CdsCachedCHEntry ::= SEQUENCE
  {
    aCachedCHName     DisplayString,
    aCachedCHUpTime   OCTET STRING (SIZE (50)),
    aCachedCHMiscOps  Counter,
    aCachedCHReadOps  Counter,
    aCachedCHWriteOps Counter
  }

aCachedCHName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..64))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The name of the cached clearinghouse."
     --               The string will be "Unavailable" when an error
     --                is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {cdsCachedCHEntry 1}

aCachedCHUpTime OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The date and time at which the clearinghouse
                      was added to the cache."
     --               The string will be "Unavailable" when an error
     --                is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {cdsCachedCHEntry 2}
```

```
aCachedCHMiscOps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of operations other than read
                      and write the clerk has performed on the
                      clearinghouse represented by the cache
                      entry."
    ::= {cdsCachedCHEntry 3}

aCachedCHReadOps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of lookup operations the clerk has
                      performed on the clearinghouse represented by
                      the cache entry."
    ::= {cdsCachedCHEntry 4}

aCachedCHWriteOps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of write operations the clerk has
                      sent to the clearinghouse represented by the
                      cache entry."
    ::= {cdsCachedCHEntry 5}
```

DCE Global Directory Agent Server Group

```
dceGdaSvrGroup OBJECT IDENTIFIER ::= {dcemib 15}

aGdaSvrPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of this DCE GDA server."
    ::= {dceGdaSvrGroup 1}

aGdaSvrUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID of this DCE GDA server."
    ::= {dceGdaSvrGroup 2}

aGdaSvrGid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The group ID of this DCE GDA server."
    ::= {dceGdaSvrGroup 3}

aGdaSvrInRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls received by the GDA server."
    ::= {dceGdaSvrGroup 4}

aGdaSvrOutRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls initiated by the GDA server."
    ::= {dceGdaSvrGroup 5}

aGdaSvrInRpcPkts OBJECT-TYPE
```

```
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets received by the GDA server."
    ::= {dceGdaSvrGroup 6}

aGdaSvrOutRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets initiated by the GDA server."
    ::= {dceGdaSvrGroup 7}

aGdaSvrState OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state of this DCE GDA server."
     --               One of the following strings will be set
     --               based on the state value:
     --               "Unknown"          A problem was encountered while trying to resolve
     --                                  the state of the server.
     --               "Not running"      The server is configured, and is not running.
     --               "Available"        The server is configured and running.
     --               "Not configured"   DCE is not configured.
    ::= {dceGdaSvrGroup 8}

aGdaSvrStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this DCE GDA server."
    ::= {dceGdaSvrGroup 9}
```

DCE Distributed Time Service Entity Group

```
dceDtsEntityGroup OBJECT IDENTIFIER ::= {dcemib 16}

aDtsdCurrTime OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "Current time on the node."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceDtsEntityGroup 1}

aDtsdBadProtocols OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the local node failed
                      to process a received message containing an
                      incompatible protocol version."
    ::= {dceDtsEntityGroup 2}

aDtsdBadTimeReps OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
```

```
    DESCRIPTION      "The number of times the local node failed to
                      process a received message containing an
                      incompatible timestamp format."
    ::= {dceDtsEntityGroup 3}

aDtsdInsufRes OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the node has been unable
                      to allocate virtual memory."
    ::= {dceDtsEntityGroup 4}

aDtsdLocalNotIntersects OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the node's time interval
                      failed to intersect with the computed interval
                      of the servers."
    ::= {dceDtsEntityGroup 5}

aDtsdSyncCmpltd OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the node successfully
                      synchronized time."
    ::= {dceDtsEntityGroup 6}

aDtsdSysErrors OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the DTS entity detected
                      a system error."
    ::= {dceDtsEntityGroup 7}

aDtsdTooFewSvrs OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times a node failed to
                      synchronize because it could not contact the
                      required minimum number of servers."
    ::= {dceDtsEntityGroup 8}

aDtsdGlobalTo OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The amount of time the node waits for a
                      response to a WAN synchronization request
                      before sending another request or declaring
                      a global server to be unavailable."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceDtsEntityGroup 9}

aDtsdLocalTo OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The amount of time the node waits for a
                      response to a synchronization request before
```

```
                        sending another request or declaring a server
                        to be unavailable."
     --                 The string will be "Unavailable" when an error
     --                 is encountered trying to retrieve the string.
     --                 The string will be "Not applicable" when
     --                 object does not apply.
    ::= {dceDtsEntityGroup 10}

aDtsdPid OBJECT-TYPE
    SYNTAX              INTEGER
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The process ID of the DCE DTS entity."
    ::= {dceDtsEntityGroup 11}

aDtsdUid OBJECT-TYPE
    SYNTAX              INTEGER
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The user ID of the DCE DTS entity."

aDtsdGid OBJECT-TYPE
    SYNTAX              INTEGER
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The group ID of the DCE DTS entity."
    ::= {dceDtsEntityGroup 13}

aDtsdInRpcCalls OBJECT-TYPE
    SYNTAX              Counter
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The number of RPC calls received by the DCE
                        DTS entity."
    ::= {dceDtsEntityGroup 14}

aDtsdOutRpcCalls OBJECT-TYPE
    SYNTAX              Counter
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The number of RPC calls initiated by the DCE
                        DTS entity."
    ::= {dceDtsEntityGroup 15}

aDtsdInRpcPkts OBJECT-TYPE
    SYNTAX              Counter
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The number of RPC packets received by the DCE
                        DTS entity."
    ::= {dceDtsEntityGroup 16}

aDtsdOutRpcPkts OBJECT-TYPE
    SYNTAX              Counter
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The number of RPC packets initiated by the
                        DCE DTS entity."
    ::= {dceDtsEntityGroup 17}

aDtsdState OBJECT-TYPE
    SYNTAX              DisplayString (SIZE (0..16))
    ACCESS              read-only
    STATUS              mandatory
    DESCRIPTION       "The state of this DCE DTS entity."
     --                 One of the following strings will be set
     --                 based on the state value:
```

```
     --              "Unknown"          A problem was encountered while trying to resolve
     --                                 the state of the server.
     --              "Not running"      The server is configured, and is not running.
     --              "Available"        The server is configured and running.
     --              "Not configured"   DCE is not configured.
    ::= {dceDtsEntityGroup 18}

aDtsdStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this DCE DTS entity."
    ::= {dceDtsEntityGroup 19}
```

DCE Distributed Time Service Server Group

```
dceDtsSvrGroup OBJECT IDENTIFIER ::= {dcemib 17}


aDtsSvrRole OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                    (0),
         vLocal                      (1),
         vGlobal                     (2)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The role of the DTS server."
    ::= {dceDtsSvrGroup 1}

aDtsSvrCourierRole OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                  (0),
         vCourier                  (1),
         vBackup                   (2),
         vNoncourier               (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "A server's interaction with the set of global
                      servers."
    ::= {dceDtsSvrGroup 2}

aDtsSvrDiffEpochs OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the node received time
                      response messages that had a different epoch
                      number."
    ::= {dceDtsSvrGroup 3}

aDtsSvrNoGlobals OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the clock could not
                      contact any global clocks."
    ::= {dceDtsSvrGroup 4}
```

```
aDtsSvrNotResponds OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the clock could not
                      contact a specific global clock."
    ::= {dceDtsSvrGroup 5}

aDtsSvrProviderErrors OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of times the external time
                      provider signaled a failure or the node was
                      unable to access the time provider."
    ::= {dceDtsSvrGroup 6}

aDtsSvrRoleString OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The role of the DTS server."
     --               One of the following strings will be set
     --               based on the aDtsSvrRole value:
     --                    "Unknown"
     --                    "Local"
     --                    "Global"
     --               If there are no errors encountered and the
     --               role is not local or global, then "Clerk" is
     --               used instead of "Unknown."
     --
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceDtsSvrGroup 7}

aDtsSvrCourierRoleString OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "A server's interaction with the set of global
                      servers."
     --               One of the following strings will be set
     --               based on the aDtsSvrCourierRole value:
     --               "Unknown"
     --               "Courier"
     --               "Backup"
     --               "Noncourier"
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dceDtsSvrGroup 8}
```

DCE Distributed Time Service Known Server Table Group

```
dtsKnownSvrTable OBJECT-TYPE
    SYNTAX            SEQUENCE OF     DtsKnownSvrEntry
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "A list of DCE DTS local or global servers
                      known by this DCE DTS entity that do not reside on this DCE host."
    ::= {dcemib 18}

dtsKnownSvrEntry OBJECT-TYPE
    SYNTAX            DtsKnownSvrEntry
    ACCESS            not-accessible
```

```
    STATUS            mandatory
    DESCRIPTION      ""
    INDEX            {aDtsKnownSvrName}
    ::= {dtsKnownSvrTable 1}

DtsKnownSvrEntry ::= SEQUENCE
  {
    aDtsKnownSvrName    DisplayString,
    aLastPolled         OCTET STRING (SIZE (50)),
    aLastObsSkew        DisplayString,
    aUsedInLastSync     DisplayString,
    aLastObsTime        OCTET STRING (SIZE (50)),
    aDtsSvrProto        INTEGER,
    aDtsSvrProtoString  DisplayString
  }

aDtsKnownSvrName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..64))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The name of the known DCE DTS server known
                      by this host."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dtsKnownSvrEntry 1}

aLastPolled OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The time that the known DCE DTS server was
                      last polled."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dtsKnownSvrEntry 2}

aLastObsSkew OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The last observed time difference of the
                      known DCE DTS server."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dtsKnownSvrEntry 3}

aUsedInLastSync OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The time that the known DTS server was last
                      synchronized."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dtsKnownSvrEntry 4}

aLastObsTime OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            read-only
```

```
    STATUS            mandatory
    DESCRIPTION      "The time that the known DTS server was last
                      observed."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dtsKnownSvrEntry 5}

aDtsSvrProto OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                    (0),
         vIEEE8023                   (1),
         vDCEnet                     (2),
         vUDPIP                      (3),
         vTCPIP                      (4),
         vRPC                        (5)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The transport protocol of the known DTS
                      server."
    ::= {dtsKnownSvrEntry 6}

aDtsSvrProtoString OBJECT-TYPE
    SYNTAX            DisplayString( SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The transport protocol of the known DTS
                      server."
     --               One of the following strings will be set
     --               based on the aDtsSvrProto value:
     --               "Unknown"
     --               "IEEE 802.3"
     --               "UDP/IP"
     --               "TCP/IP"
     --               "RPC"
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {dtsKnownSvrEntry 7}
```

DCE Traps Group

```
trapsGroup OBJECT IDENTIFIER ::= {dcemib 19}

aEventType OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..12))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The type of the event."
     --               The event type will be one of the following
     --               strings:
     --                   "DCE Generic"
     --                   "DCE Service" --- "DCE SNMP"
         ::= {trapsGroup 1}

aEventText OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..256))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The description of the event."
     --
     --               If you start the subagent from the command line,
     --               you can issue the following before starting it.
     --
```

```
     --                          "set svc_ssa_dbg=ssa:*.1"
     --
     --               You can also change the routing file in the ..\var\svc directory.
     --               This causes some information to prefix the event text.
     --               The prefix has the following appearance:
     --
     --                  (msgid: total_trap_count; trap_count_by_severity)
     --
     --               The msgid displays as: 0xhhhhhhhh.  The total trap
     --               count is the total number of traps issued since the
     --               subagent was started.  The trap count by severity
     --               appears as: Xcount, where:
     --
     --                  X = F (fatal),  E (error),  W (warning),
     --                      N (notice), V (notice verbose),
     --                      H (Statuschange), or U (unknown).
     --
     --                0x00000000 is seen for the subagent start and
     --                stop traps.
    ::= {trapsGroup 2}

aEventSeverity OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The severity level of the event."
     --               The event severity will be one of the
     --               following strings:
     --                   "UNKNOWN"
     --                   "FATAL"
     --                   "ERROR"
     --                   "WARNING"
     --                   "NOTICE"
     --                   "NOTICE_VERBOSE"
     --                   "NOT RUNNING"    (for status change check only)
     --                   "AVAILABLE"      (for status change check only)
    ::= {trapsGroup 3}

aCellName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..64))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The name of the cell where the event occurred."
    ::= {trapsGroup 4}

aHostName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..128))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The name of the host where the event occurred."
    ::= {trapsGroup 5}

aTime OBJECT-TYPE
    SYNTAX            OCTET STRING (SIZE (50))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The time when the event occurred."
    ::= {trapsGroup 6}

aProgram OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The name of the program that generated the
                      event."
     --               For server status change checking, this contains the
     --               name of the server that has undergone the status
```

```
     --               change.
    ::= {trapsGroup 7}

aComponent OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The name of the component that generated the
                      event."
    ::= {trapsGroup 8}

aSubComponent OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..32))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The name of the subcomponent that generated
                      the event."
    ::= {trapsGroup 9}

aThreadId OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The ID of the thread where the event occurred."
    --                This is zero for server status change traps.
    ::= {trapsGroup 10}

aFile OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..128))
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The name of the source file that generated
                      the event."
    ::= {trapsGroup 11}

aLine OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "The number of the source line that generated
                      the event."
    ::= {trapsGroup 12}
```

DCE Security Audit Server Group

```
dceAuditSvrGroup OBJECT IDENTIFIER ::= {dcemib 20}

aAuditSvrPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of this DCE Security Audit server."
    ::= {dceAuditSvrGroup 1}

aAuditSvrUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID of this DCE Security Audit server."
     ::= {dceAuditSvrGroup 2}

aAuditSvrGid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The group ID of this DCE Security Audit server."
    ::= {dceAuditSvrGroup 3}
```

```
aAuditSvrInRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls received by the server."
    ::= {dceAuditSvrGroup 4}

aAuditSvrOutRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls initiated by the server."
    ::= {dceAuditSvrGroup 5}

aAuditSvrInRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets received by the server."
    ::= {dceAuditSvrGroup 6}

aAuditSvrOutRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets initiated by the server."
    ::= {dceAuditSvrGroup 7}

aAuditSvrState OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state of this DCE Security Audit server."
     --               One of the following strings will be set
     --               based on the state value:
     --               "Unknown"          A problem was encountered while trying to resolve
     --                                  the state of the server.
     --               "Not running"      The server is configured, and is not running.
     --               "Available"        The server is configured and running.
     --               "Not configured"   DCE is not configured.
    ::= {dceAuditSvrGroup 8}

aAuditSvrStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this DCE Security Audit server."
    ::= {dceAuditSvrGroup 9}
```

DCE Password Strength Server Table Group

```
pwdStrengthSvrTable OBJECT-TYPE
    SYNTAX            SEQUENCE OF     PwdStrengthSvrEntry
    ACCESS            not-accessible
    STATUS            mandatory
    DESCRIPTION      "A list of DCE Password Strength servers configured
                      on this host."
    ::= {dcemib 21}

pwdStrengthSvrEntry OBJECT-TYPE
    SYNTAX            PwdStrengthSvrEntry
    ACCESS            not-accessible
```

```
    STATUS            mandatory
    DESCRIPTION      ""
    INDEX            {aPWstrengthName}
    ::= {pwdStrengthSvrTable 1}

PwdStrengthSvrEntry ::= SEQUENCE
  {
    aPWstrengthName           DisplayString,
    aPWstrengthPid            INTEGER,
    aPWstrengthUid            INTEGER,
    aPWstrengthGid            INTEGER,
    aPWstrengthInRpcCalls     Counter,
    aPWstrengthOutRpcCalls    Counter,
    aPWstrengthInRpcPkts      Counter,
    aPWstrengthOutRpcPkts     Counter,
    aPWstrengthState          DisplayString,
    aPWstrengthStateValue     INTEGER
  }

aPWstrengthName OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..64))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The name of the DCE Password Strength server
                      on this host."
     --               The string will be "Unavailable" when an error
     --               is encountered trying to retrieve the string.
     --               The string will be "Not applicable" when the
     --               object does not apply.
    ::= {pwdStrengthSvrEntry 1}

aPWstrengthPid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The process ID of this DCE Password Strength server."
    ::= {pwdStrengthSvrEntry 2}

aPWstrengthUid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The user ID of this DCE Password Strength server."
     ::= {pwdStrengthSvrEntry 3}

aPWstrengthGid OBJECT-TYPE
    SYNTAX            INTEGER
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The group ID of this DCE Password Strength server."
    ::= {pwdStrengthSvrEntry 4}

aPWstrengthInRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls received by the server."
    ::= {pwdStrengthSvrEntry 5}

aPWstrengthOutRpcCalls OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC calls initiated by the server."
    ::= {pwdStrengthSvrEntry 6}

aPWstrengthInRpcPkts OBJECT-TYPE
```

```
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets received by the server."
    ::= {pwdStrengthSvrEntry 7}

aPWstrengthOutRpcPkts OBJECT-TYPE
    SYNTAX            Counter
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The number of RPC packets initiated by the server."
    ::= {pwdStrengthSvrEntry 8}

aPWstrengthState OBJECT-TYPE
    SYNTAX            DisplayString (SIZE (0..16))
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state of this DCE Password Strength server."
     --               One of the following strings will be set
     --               based on the state value:
     --               "Unknown"          A problem was encountered while trying to resolve
     --                                  the state of the server.
     --               "Not running"      The server is configured, and is not running.
     --               "Available"        The server is configured and running.
     --               "Not configured"   DCE is not configured.
    ::= {pwdStrengthSvrEntry 9}

aPWstrengthStateValue OBJECT-TYPE
    SYNTAX            INTEGER
    {
         vUnknown                              (0),
         vNotRunning                           (1),
         vAvailable                            (2),
         vNotConfigured                        (3)
    }
    ACCESS            read-only
    STATUS            mandatory
    DESCRIPTION      "The state value of this DCE Password Strength server."
    ::= {pwdStrengthSvrEntry 10}
```

Chapter 12. Event Management Service (EMS)

EMS manages event services in a DCE cell. In EMS, an event is data being transmitted from an event supplier to EMS and from EMS to one or more event consumers. An event consists of an event header and a list of event attributes that contain the event type-specific data.

EMS consists of three major components:

- **The EMS daemon (emsd)** is a server that:
  - Authenticates and authorizes event suppliers and consumers
  - Maintains databases of event types, event filters, and consumers
  - Associates an event filter group with each event consumer
  - Ensures reliable delivery of events to interested consumers.
- **The event supplier** is any DCE-based user application that emits event data.
- **The event consumer** is a requestor that:
  - Queries EMS for supported event types
  - Obtains a list of existing filter names
  - Constructs event filters for each event type
  - Adds event filters to its event filter group.

EMS data structures are grouped into the following functions:

- Event Attributes
- Event Structure
- Event Types
- Event Filters
- Consumer Data Structures
- Server Data Structures

The ability to route events to EMS is integrated with the SVC subsystem through the **ems** serviceability routing specification. You can also use EMS to communicate events through the SVC to the DCE SNMP subagent. For more information on routing messages, see *OSF® DCE Application Development Guide—Core Components*.

DCE administrative functions include management of EMS servers, event queues, and event logs. For more information, see:

- “Logging EMS Events” on page 149
- “Managing EMS Consumers” on page 150
- “Managing EMS Event Filters” on page 151
- “Managing EMS Event Queues” on page 152
- “Managing the EMS Daemon” on page 153
- “Setting Permission for the EMS Server” on page 153
- “Starting the EMS Server” on page 149

DCE Event Management Service

This section describes the DCE Event Management Service (EMS), which provides asynchronous event support for DCE based applications. EMS APIs provide an interface to the suppliers, consumers, and event service administration for use by EMS clients.

In both traditional (SNMP and CMIP) and object system management architectures, communications between the managing and managed systems is bidirectional. One or more managing systems can send requests to query and control various aspects of resources being managed. In addition, the managed resource must be capable of sending asynchronous notifications or events to the managing systems. An event marks a change in state of the managed resource that causes a notification to be sent to interested parties. The routing of these events is usually done through an agent on the managed system.

EMS uses the concepts of event suppliers and event consumers and establishes an event channel between them to support asynchronous communication. In the context of DCE, event suppliers are any DCE-based user application (client or server), and event consumers can be any application with an interest in receiving asynchronous events from one or more DCE processes. An event channel (operating as both a supplier and consumer of events) is a service that decouples the communications between event suppliers and event consumers. EMS also provides a filtering mechanism to allow administrators and consumers control over the events that EMS sends.

EMS provides transparent support for DCE clients and servers using the DCE Serviceability (SVC) and Audit. DCE applications can use the APIs offered in SVC and Audit to become event suppliers.

Functional Highlights

The functional highlights of the DCE EMS are as follows:

- The effort involved in writing a DCE event consumer application is minimized.
- The EMS service improves performance and should minimize network and system load.
- Event consumer applications can locate and register with one or more EMS servers on multiple DCE hosts.

  Note: An EMS server is an EMS daemon (EMSD).
- Events are sent by DCE applications only once through the SVC or Audit interface to EMS.
- Event consumer applications can control the DCE host that is sending events as well as the event types that are sent.
- EMS can transmit events from multiple DCE event supplier applications to one or more DCE event consumer applications, based on a defined set of event filters within a DCE cell in a secure way.
- DCE event supplier applications are not aware of the DCE consumer applications that have registered to receive events.
- EMS provides a remote management API.
- EMS provides reliable delivery of events to consumers and is tolerant of network and machine failures.

- EMS ensures the ability to define and extend events and event contents.

Functional Definition

DCE EMS manages event services in a DCE cell. EMS consists of two parts — the EMSD (EMS daemon) server and the API to access event services.

EMSD is a DCE server that resides on every DCE host in the cell that consumers request events from. The EMS API provides an interface to support event suppliers, event consumers, and EMS server administration.

Event Flow Description

EMS sets up an event, channel to decouple the communications between the supplier and consumer.

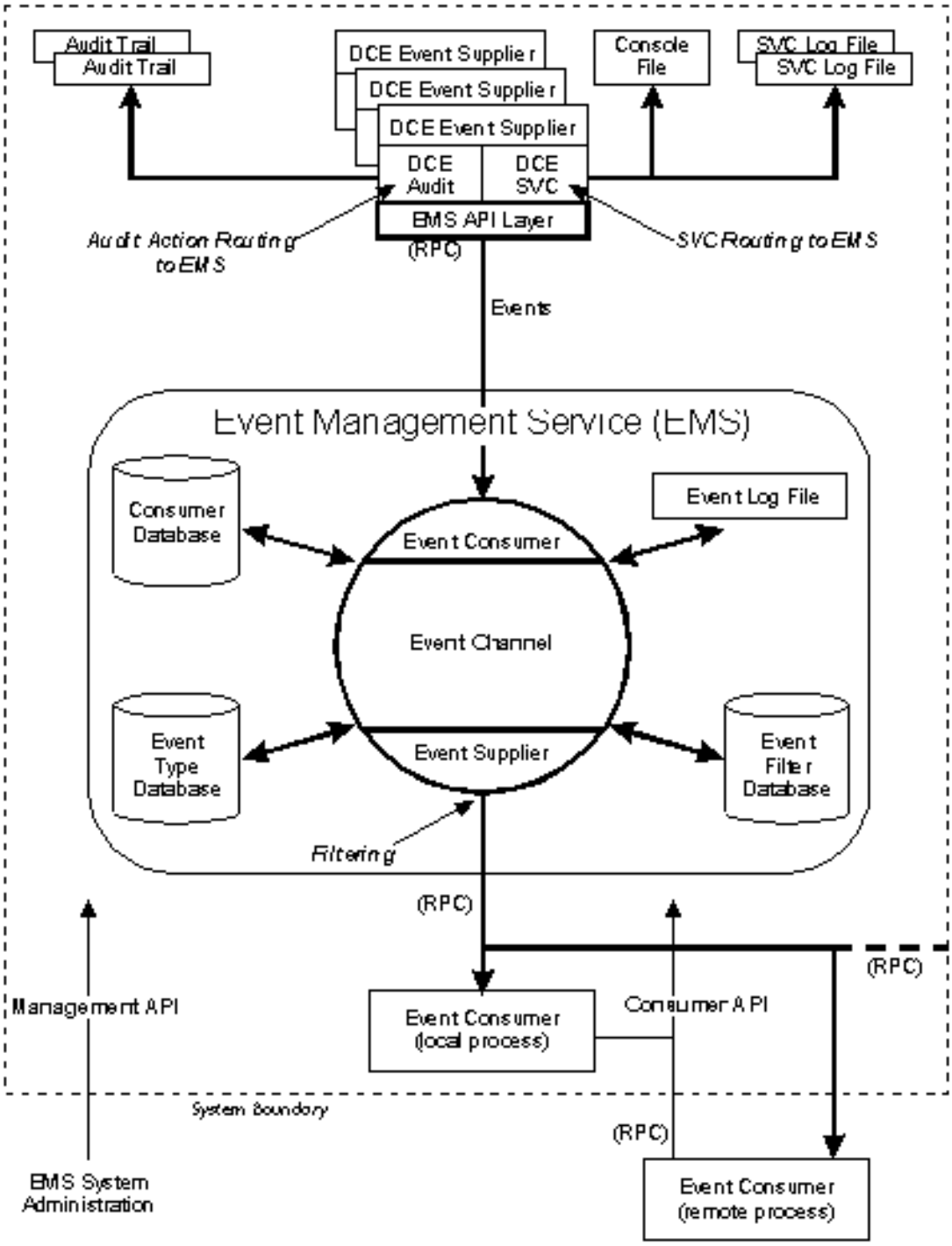
To send events you must enable SVC or audit events to get to EMS.

In order to start receiving events, an event consumer must first register with EMS, and then set up an event filter group to tell EMS the events to forward to that consumer.

After the event is sent to EMS, it is written to the EMS event log as a backup in case the event cannot be delivered immediately.

After the event reaches EMS, it must pass through a consumer event filter before being forwarded to interested consumers. EMS scans the list of registered consumers and uses the event type schema from the event type Database and the consumers event filter group from the consumer Database, and the event filters from the event filter Database to determine if this event passes through to be forwarded for each event consumer. After all appropriate consumers receive the event, the event is removed from the event log.

Relationship of EMS and DCE Subsystems



Event Type Definition

The format of EMS event types are defined by event type schemas and are kept in the EMS event Type Database. The event type schemas consist of a list of attribute names along with the attribute type, which specifies the data format of the data associated with that attribute. Events consist of a fixed header part and a

variable-length data part. The variable-length data part consists of *N* self-defining data items that consist of an attribute type and then the data itself.

The event type schemas are used in several different ways.

A consumer can request a list of supported event types, select the events types it wants to receive by using the event type schemas to construct event filters, and map event data according to attribute names. For example, an event consumer can reconstruct an SVC message by using the attribute names to find the correct data items.

EMS uses the event type schemas to apply event filters to events.

Generic Event Types: EMS supports events with type **Generic**. Generic events do not have an event type schema. The only way to define filters for generic events is to use filter expressions with event header attributes.

Default Event Types: The following defines the SVC event attribute lists that are contained in their event type schemas.

```
#define CNT_SVC_ATTRS (sizeof(svc)/sizeof(ems_attribute_t))

static ems_attribute_t svc}} = {
        {(unsigned char *)″version″,                {ems_c_attr_ulong_int,0} },
        {(unsigned char *)″t″,                          {ems_c_attr_utc,0} },
        {(unsigned char *)″argtypes″,               {ems_c_attr_char_string,0} },
        {(unsigned char *)″table_index″,          {ems_c_attr_ulong_int,0} },
        {(unsigned char *)″attributes″,             {ems_c_attr_ulong_int,0} },
        {(unsigned char *)″message_index″,     {ems_c_attr_ulong_int,0} },
        {(unsigned char *)″format″,                    {ems_c_attr_char_string,0} },
        {(unsigned char *)″file″,                       {ems_c_attr_char_string,0} },
        {(unsigned char *)″progname″,               {ems_c_attr_char_string,0} },
        {(unsigned char *)″line″,                        {ems_c_attr_ulong_int,0} },
        {(unsigned char *)″threadid″,                {ems_c_attr_ulong_int,0} },
        {(unsigned char *)″component_name″, {ems_c_attr_char_string,0} },
        {(unsigned char *)″sc_name″,                  {ems_c_attr_char_string,0} },
        {(unsigned char *)″attribute.debug″,       {ems_c_attr_ushort_int,0} },
        {(unsigned char *)″attribute.severity″,   {ems_c_attr_ushort_int,0} },
        {(unsigned char *)″attribute.actroute″,   {ems_c_attr_ulong_int,0} }
        };

#define CNT_AUDIT_ATTRS (sizeof(audit)/sizeof(ems_attribute_t))

static ems_attribute_t audit** = {
        {(unsigned char*) ″format″,                {ems_c_attr_ushort_int, 0} },
        {(unsigned char*) ″server″,                {ems_c_attr_uuid, 0} },
        {(unsigned char*) ″event″,                 {ems_c_attr_ulong_int, 0} },
        {(unsigned char*) ″outcome″,           {ems_c_attr_ushort_int, 0} },
        {(unsigned char*) ″authz_st″,          {ems_c_attr_ushort_int, 0} },
        {(unsigned char*) ″time″,                  {ems_c_attr_utc, 0} },
        {(unsigned char*) ″addr″,                  {ems_c_attr_char_string, 0} }
        };
```

Several constants have also been defined for matching against the **attribute_severity** attribute. They are:

SVC_C_SEV_FATAL

SVC_C_SEV_ERROR
SVC_C_SEV_WARNING
SVC_C_SEV_NOTICE
SVC_C_SEV_NOTICE_VERBOSE

User-defined Event Types: EMS also allows event suppliers to define new event types. After a new event type is defined, the events of that event type can be sent by the event suppliers. The event type can then be consumed by consumers.

An event supplier defines a new event type by specifying a unique name, a UUID for the event type, and a list of event attributes. For more information on the routines to add, delete, and to get event types, see “EMS Event Type Routines” on page 198.

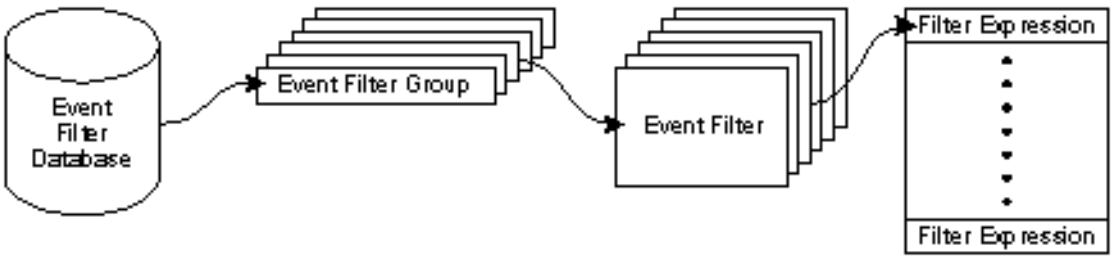
Filtering

EMS supports event filtering for type and generic event types. Only messages that are routed to EMS are sent through the EMS event channel.

An event consumer must add filters to its filter group before it can receive events. This associates an event filter group with each event consumer and that consumer receives only events that pass through one of the entries in the consumer event filter group.

Defining Event Filters: The following is a typical scenario of how an event consumer would start interfacing with EMS. The consumer first queries EMS about the event types that are supported. From that list of event types, the consumer then queries the event type schema in order to construct any event filters for that event type. The event type schemas contain the list of attribute names and attribute types the consumer uses to construct an event filter. Next, the consumer tells EMS to apply the constructed event filter to incoming events by adding it to the consumers event filter group. EMS filters are constructed at several different levels. The lowest level is the filter expression (consisting of an attribute name, attribute operator, and an attribute value that defines a compare operation). All compare operations evaluate to a boolean value. Only certain attribute operators are allowed on certain attribute types.

The Different Levels of an EMS Filter



The following example illustrates a filter expression that evaluates to TRUE if the attribute name **file** of an SVC event type is equal to the string **file.c**.

```
ems_filter_exp_t  xmp_SVC;

xmp_SVC.attr_name  =  ″file″;
xmp_SVC.attr_operator  =  ems_c_attr_op_eq;
```

```
xmp_SVC.attr_value.format = ems_c_attr_char_string;
xmp_SVC.attr_value.tagged_union.char_string = ″file.c″;
```

An event filter is made up of a list of filter expressions that are joined together with **and**. An event filter has both a name and an event type. The event filter name is used to refer to the event filter in event filter create, delete and update operations. The event filter name is also used by consumers and EMS administrators to add and delete event filters to an event filter group.

A consumer event filter group is a collection of one or more event filter names. Before an event can pass through a consumer filter group, the event filter associated with each event filter name specified in the event filter group must be applied to the event. If all the event filter expressions in the named event filter evaluate to TRUE, the event is forwarded by EMS to that consumer. If any of the event filter expressions evaluate to FALSE, the event filter specified by the next event filter name in the event filter group is applied to the event. If none of the event filters in the event filter group evaluate to TRUE, the event is not forwarded to that consumer.

Using the filter API, a consumer can define a new event filter and then add that filter name to its event filter group. A consumer can also get a list of existing event filter names and add one of those event filter names to its event filter group. The following list includes routines for retrieving existing filters:

ems_filter_get
: Gets the contents of an event filter

ems_filter_get_namelist
: Gets a list of the names of all filters in the Event Filter Database

ems_filter_get_list
: Gets a list of all the filters in the Event Filter Database

The consumer can add or delete event filter names from event filter groups, as well as by the administrative interface.

Filtering on Header Information or Event Type: A predefined set of event header attribute name-type pairs have been defined to allow filtering on the information in the event header. Filter expressions with event header attributes can be part of filters with event type specific attributes. Filters with type **Generic**, can only have filter expressions with header attributes (that is, using a generic filter, consumers can filter only on the information contained in an event header).

User Interface Considerations

To simplify filter construction, you can use the **ems_event_types_get_getlist** call to list event type schemas. You can extract those attributes and use them to construct filters for that event type. The design of EMS facilitates developing an event management user interface to consumer applications. This application can use the **ems_event_types_get_getlist** call to get a list of the event type schemas for all the event types that suppliers send to EMS. The event types can then be presented for use in constructing filters.

Writing Consumers

Consumers are not simple clients. They have to be implemented as servers. Therefore, EMS consumers have certain requirements. Consumers must:

1. Call **consumer_start**
2. Register an event handler
3. Register with the various EMSD servers it wants to receive events from
4. Set up event filters
5. Call **rpc_server_listen**.

Using the Sample Supplier and Consumer

The sample consumer is set up to receive events from EMS. The sample supplier is set up to serve as an event supplier. The sample supplier provided by DCE issues messages of all severity types. It uses the **sup.sams** file to define the messages that are issued and sends events to EMS using calls to the **dce_svc_printf** DCE Application Programming Interface (API).

Events reach the sample consumer because it has set up the appropriate registration information, event filtering information, and has successfully started listening for events. Each event consumer is responsible for setting up this information. An event consumer is not a simple client and it has to be implemented as a server by following these steps:

1. Call the **ems_consumer_start** API.

   This API names the instances of the EMS consumer being set up. Event consumer names do not have to be unique. The EMS **ems_consumer_start** API creates an UUID that makes the consumer name identification unique and, therefore, easier to deal with. This sample consumer appends the process ID (converted to ASCII) to the consumer name string. Its name is **sampleConsumer***xxx*, where *xxx* is the process ID.

   This API is called once during consumer initialization. See the sample consumer code in the path to the EMS examples given previously.

2. Call the **ems_consumer_handler_register** API.

   This API registers the consumer event handler with EMS. The events that survive the event filtering defined by the consumer code are passed to the event handler.

   You define the event handler. The sample consumer does nothing but display the event format attribute text. The resulting message displays as *SEVERITY***: message** *%d* **from** *%s***!**, where *SEVERITY* is the severity (for example, **FATAL**), *%d* is a number, and *%s* is the name of the host that sent the message. The sample consumer has code that resolves the replacement text from the SVC event data. This API is called once during consumer initialization and resembles the following in the sample consumer code:

```
ems_consumer_handler_register
      (log_event,                /* name of function that processes event      */
     &status);                   /* address of status - returned from API      */
```

3. Call the **ems_consumer_register** API.

   This API registers the consumer with a host EMS daemon (**emsd**). This API is called once during consumer initialization for each **emsd** being registered. It resembles the following in the sample consumer code:

```
netname.service=ems_ns_dce;      /* DCE CDS name service... indicates the      */
                                 /* name service that recognizes the network */
                                 /* name being defined here...*/
 /* allocate memory for the network address... /*
 netname.netaddr=(ems_netaddr_t *)malloc
```

```
    (sizeof(ems_netaddr_t)+strlen(HOST1)+1);

  /* complete the definition of the network name structure...len, name...*/
  netname.netaddr->len=strlen(HOST1)+1;
    strcpy((char  *)&netname.netaddr->name[0],HOST1);
```

Note: This API must be called for each **emsd** to be monitored by the consumer. The sample consumer has been set up to allow up to two hard-coded host names.

In addition to registering with the **emsd**, the following additional steps are needed for each **emsd** registration:

a. Call the **ems_filter_add** API.

   This API identifies a "named" set of one or more filter expressions. The filter name is used to represent the list of filter expressions. An example of one filter expression may display logically as **severity = fatal**. All associated filter expressions must be TRUE for the filter to be TRUE.

   The sample consumer defines two filters for each **emsd** registered with:

   Filter Name
   Filter Definition

   CompSupAndSevLeError
   component_name = sup AND severity less than or equal to **ERROR**. (CompSup And Sevle Error)

   CompSup
   component_name = sup

   However, the first filter is used to filter events from the first host, and the second filter is used with the second host. The **sup** component is an SVC component defined in the **sup.sams** file.

   The sample code checks to see if the filter already exists. If it does exist, the **ems_filter_add** is not performed. The API resembles the following in the sample consumer:

```
ems_filter_add
   (emsHandle1,                                    /* handle returned by reg
    compSup,                                        /* name of filter group
    ems_c_svc_type,       /* filter event type
    expList1,                                       /* list of filter expressi
     &status);                          /* address of status - returned from
```

   Refer to the sample code for more details on how the list of filter expressions are set up.

b. Call the **ems_add_filter_to_group** API.

   This API is used to group a set of one or more filters defined through the **ems_filter_add** API. All of the filters in a filter group must be true before the event can survive the filtering process. Only one filter has to be TRUE when multiple groups have been defined.

   The API resembles the following in the sample consumer:

```
ems_add_filter_to_group
  (emsHandle1,            /* handle returned by registration                */
```

```
                fnList,                 /* list of filter names
                &status);  /* address of status - returned from API        */
```

Refer to the sample code for more details on how the list of filter names are set up.

4. Call the **rpc_server_listen** API.

   Finally, this API places the consumer into listen mode. It is called, specifying the maximum number of calls it can execute concurrently.

   ```
   rpc_server_listen( 8, &status );
   ```

   Because RPCs can run concurrently, the consumer writer is responsible for ensuring that the event handler routine is thread-safe. If a consumer allows concurrent calls, its remote procedures are responsible for concurrency control. If running a set of remote procedures concurrently requires concurrency control, and a consumer lacks this control, the consumer must allow only one call at a time. This number is set through this API.

   Listen mode continues until the consumer is stopped. Note that this sample consumer has been supplied with code to reauthenticate itself so that its permissions should not expire for as long as it is running. This, however, requires additional setup.

Enabling EMS

At some point, you must enable EMS to receive events. You can control the message severities are sent to EMS for forwarding on to interested event consumers. You do this by setting the local environment either from a command file or by typing, at the command line, the following:

```
set SVC_FATAL=EMS:-;STDERR:-;
set SVC_ERROR=EMS:-;STDERR:-;
set SVC_WARNING=EMS:-;STDERR:-;
set SVC_NOTICE=EMS:-;STDERR:-;
set SVC_NOTICE_VERBOSE=EMS:-;STDERR:-;
```

Instead of issuing the commands from the command line, you can add similar information to a routing file located in **%DCELOC%\dcelocal\var\svc\routing**.

Note: Setting this information in the routing file causes every SVC message (all types of severities) generated by any application on the system to be routed to EMS. This is not recommended, however, because it lowers system performance.

The preceding reference to EMS indicates that all SVC message types are sent to **EMS** and **STDERR**. You can modify this. Minimally, you must enable EMS for the SVC type that you expect to monitor for.

You can also use the **dcecp log** command.

Compiling

See the example in **%DCELOC%\dcelocal\examples\ems** for compiling information. Also, see the **README** file.

Event Consumer Template

The example stored in **%DCELOC%\dcelocal\examples\ems** illustrates the structure of a typical event consumer.

You must write an event handler to log events before you can register it. The sample consumer is written to print out logged events; however, you can modify it to perform in accordance with your needs. For example, you can customize the event handler to call a particular pager number whenever a severity of FATAL displays.

Starting the EMS Server

The **emsd** command starts the EMS daemon. The EMS daemon must be running on the host system in the DCE cell before a consumer can receive events or a supplier can supply events.

The EMS daemon runs under the local host machine principal identity (**host/***hostname***/self**). The DCE Host daemon (**dced**) must be running on the local host when **emsd** is started. The **emsd** command also requires a CDS Advertiser (**cdsadv**).

The **emsd** command has the following optional parameters:

–l*log_directory*
: Specifies where the log file resides.

–q*queue_size*
: Specifies the maximum number of events that are counted by EMS.

–w *svc_route*
: Specifies DCE serviceability routing instructions.

To start the EMS daemon and to specify the queue size and location of the log, type:

```
emsd -q 2048 -l /opt/dcelocal/var/ems
```

To start the EMS daemon and to specify the serviceability routing instructions, and define the maximum queue size, type:

```
emsd -w NOTICE:STDOUT:- -w NOTICE_VERBOSE:STDOUT-:-
```

Logging EMS Events

The EMS event log is used to store events in case of EMS failures. EMS writes all events to the event log and deletes the event record after the event has been transmitted to all consumers that are supposed to receive the event. The event log is kept in a file on the machine where **emsd** is running. Events are stored in a directory specified by the environment variable **EMS_EVENTLOG_DIR**. An API is provided to examine local event logs.

The **emslog** object represents the EMS event log. The **emslog** command is issued in the **dcecp** environment and is followed by one of the following subcommands:

help Returns help information on the object.

operations
Returns a list of operations supported by the object.

show Returns a list of events in the event log file.

To display the general EMS log help information, type:

```
dcecp> emslog help
```

EMS displays:

```
help        Print a summary of command-line options.
operations  Returns the valid operations for command.
show        Returns a list of events in the event log file.
```

To obtain a list of operations supported by the object, type:

```
dcecp> emslog operations
```

EMS displays:

```
show help operations
```

To display a list of events in the event log file, type:

```
dcecp> emslog show

EMS Displays:
--- Start of an EMS event record ---
Type: SVC:Event Id: 8d1b0b00-e9e7-11ce-8af3-10005a890435
Name Service: DCE /.../eagle_dce/hosts/hidalgod.austin.ibm.com
Description Name: EMS_Test_Producer
PID: 565 UID: 0 GID: 0
Severity: NOTICE
Arrival Time: 1995-09-08-14:06:32.970+00:00I-----
Printing 16 items
Item 1: [version] = ulong init 1
Item 2: [t] = 1995-09-08-14:06:32.970+00:00I-----
Item 3: [argtypes] = char string
Item 4: [table_index] = ulong int 0
Item 5: [attributes] = ulong int 64
Item 6: [message_index] = unlon int 389738500
Item 7: [format] = char string Test Supplier starting
Item 8: [file] = char string supplier.c
Item 9: [progname] char string EMS_Test_Producer
Item 10: [line] = ulong int 63
Item 11: [threadid] = ulong int 2
Item 12: [component_name] = char string sup
Item 13: [sc_name] = char string general
Item 14: [attribute.debug] = ushort int 0
Item 15: [attribute.severity] = ushort int 4
Item 16: [attribute.actroute] = ulong int 0
--- End of an EMS event record ---
```

Managing EMS Consumers

EMS consumers register with the event server to receive events. Each consumer has a name, a UUID, a host where it is running, and a list of filter names that make up the filter group.

If a consumer process terminates abnormally, the system administrator may have to delete the consumer using the **emsconsumer** command.

If new filters have been defined in the filter database, the administrator can use the **emsconsumer** command to add those filters to the consumer filter groups to further

refine the events that a consumer receives. Also, if a consumer is not receiving all the events that it should, the administrator can delete filters with the **emsconsumer** command.

DCE provides the **emsconsumer** command and associated subcommands to manage the consumer. This command is issued in the **dcecp** environment and can execute the following subcommands:

catalog
Returns the list of consumers registered with EMS on a host.

delete Deletes a registered consumer from EMS on a host.

help Displays help information on the object.

modify
Modifies the event filter group associated with the given consumer.

operations
Returns a list of operations supported by the object.

show Returns the list of filter names in a consumer filter group.

The following are the required permissions:

- For **emsconsumer catalog** and **emsconsumer show**, you must have **r** permission on **/.:/***hostname***/ems-server/consumers**.
- For **emsconsumer delete**, you must have **d** permission on **/.:/***hostname***/ems-server/consumers**.
- For **emsconsumer modify**, you must have **w** permission on **/.:/***hostname***/ems-server/consumers**.

To obtain the list of consumers registered with EMS, type:

```
dcecp> emsconsumer catalog
```

EMS displays:

```
{consumer1 7e383761-f41f-11ce-9051-08005acd43c6 /.:/hosts/eagle.austin.ibm.com}
{consumer1 a4c7ff26-f449-11ce-a863-10005a4f3556 /.:/hosts/eagle.austin.ibm.com}
{consumer2 283cc40c-f447-11ce-9dd3-10005a4f3556 /.:/hosts/umesh.austin.ibm.com}
```

To add the filter **foo** to the **consumer2** event filter group, type:

```
dcecp> emsconsumer modify
consumer2 -add {filter foo}
```

To display the list of filter names in the **consumer2** filter group, type:

```
dcecp> emsconsumer show consumer2
```

EMS displays:

```
{foo2 foo3 foo4 foo5}
```

Managing EMS Event Filters

EMS event filters are applied by EMS to events received from suppliers to determine if the events are to be forwarded to the consumers.

An EMS event filter is a collection of one or more filter expressions. Each filter expression consists of an attribute name, an attribute operator, and an attribute value.

You can issue the **emsfilter** command with an associated subcommand in the **dcecp** environment to manage event filters on the local host. You can also specify the **–host** option to issue the command to a different host. The format of the DCE host name accepted is either an entire DCE name (for example, **/.:/hosts/jurassic.austin.ibm.com**) or a DCE host name with a domain name (for example, **jurassic.austin.ibm.com**).

The **emsfilter** executes the following subcommands:

catalog
: Returns a list of all filter names in EMS.

delete Deletes a filter and its associated filter expressions from EMS.

help Displays help information on the object.

operations
: Returns a list of operations supported by the object.

show Returns a list of filter expressions in a specified filter.

The following are the required permissions:

- For **emsfilter catalog** and **emsfilter show**, you must have **r** permission on **/.:/***hostname***/ems-server/filters**.
- For **emsfilter delete**, you must have **d** permission on **/.:/***hostname***/ems-server/filters/***filtername*.

To display the filters kept by the EMS daemon, type:

```
dcecp> emsfilter catalog
```

EMS displays:

```
Filter1
Filter2
```

To delete the filter named **Filter1** and its associated filter expressions, type:

```
dcecp> emsfilter delete Filter1
```

To display a list of operations supported by the object, type:

```
dcecp> emsfilter operations
```

EMS displays:

```
catalog delete show help operations
```

To display a list of filter expressions in the **Filter2** filter, type:

```
dcecp> emsfilter show Filter2
```

EMS displays:

```
{event_type == SVC}
{file == file.c}
```

Managing EMS Event Queues

The EMS event queue size can be set at **emsd** startup using the **–q** option or the **EMS_QUEUE_SIZE** environment variable. If EMS starts receiving **queue full** errors, the daemon should be restarted using a larger queue size. See “Starting the EMS Server” on page 149 for more information on **emsd** startup.

Using the configuration GUI, only the environment variable can be used to increase the queue size.

Managing the EMS Daemon

The EMS daemon (**emsd**) is responsible for:

- Managing event ACLs in regards to event suppliers and consumers
- Maintaining databases of event types, event filters, and consumers
- Associating an event filter group with each event consumer
- Ensuring reliable delivery of events to interested consumers

The **ems** command and its associated subcommands manage the EMS daemon on a DCE host. This command is issued in the **dcecp** environment and can execute the following subcommands:

catalog
Returns a list of all hosts the EMS daemon is running on in the current cell.

help Returns help information on the object.

operations
Returns a list of operations supported by the object.

show Returns the attribute list for the EMS daemon.

The **ems show** command requires that you have the **r** permission on */.:/hostname/***ems-server**.

To list all hosts running in the current cell, type:

```
dcecp> ems catalog
```

EMS displays:

```
/.:/hosts/eagle.austin.ibm.com
/.:/hosts/umesh.austin.ibm.com
```

To return a list of operations supported by the object, type:

```
dcecp> ems operations
```

EMS displays:

```
catalog show help operations
```

To display the list of attributes for the EMS daemon, type:

```
dcecp> ems show
```

EMS displays:

```
{eventlog_dir /opt/dcelocal/dce/var/ems}
{queue_size 5000}
```

Setting Permission for the EMS Server

EMS provides for secure manipulation of data in the EMS databases. This includes the Event Filter database, the Event Type database, and the list of consumers in the Consumer database. EMS also provides for supplier and consumer authentication and authorization as well as secure transmission of event data.

All ACLs are associated with names in the DCE namespace and the EMSD server manages the namespace past the junction:

```
/.:/hosts/hostname/ems-server/
```

The ACL associated with this object controls access to the EMSD server registered in this namespace. The permissions associated with *ems-server* are:

Table 3. EMSD Server Permission Bits

Permission bit	Name	Description
c	control	Modify the ACLs on the server
r	read	Read the attributes for this server
s	stop	Stop the EMS server
w	write	Modify the attributes on this server

Three security objects are maintained under the EMS-server junction. The directories and the databases they represent are:

event-types
Event type database

filters Filter database

consumers
Consumer database

Each of these databases has an ACL associated with it.

Event Type Security Management

The Event Type database is represented by the following name in the DCE name space:

```
/.:/hosts/hostname/ems-server/event-types
```

The ACL associated with this object controls access to this database. The permissions associated with *event-types* are:

Table 4. Event Type Database Permission Bits

Permission bit	Name	Description
c	control	Modify the ACLs on the event type
d	delete	Delete an event type schema
i	insert	Add an event type schema
r	read	Read the contents of event type schemas

EMS event data access can be granted per event type. Authority on event data of a given event type can be granted by modifying the ACL on:

```
/.:/hosts/hostname/ems-server/event-types/event_type_name
```

where *event_type_name* is the event type name that appears in the event type schema. The name recognized for SVC events is:

```
/.:/hosts/hostname/ems-server/events/SVC
```

The permissions associated with *event_type_name* are:

Table 5. Event Type Permission bits

Permission bit	Name	Description
c	control	Modify the ACLs on the event type
d	delete	Delete an event type
r	read	Read (consume) an event of this type
w	write	Write (supply) an event of this type

Supplier rights are verified on the first event sent to EMS, and the consumer rights are verified before forwarding events to that consumer. Authenticated RPC is used to access the EMS supplier and consumer remote API.

Event Filter Security Management

The **Filter** database is represented by the following name in the DCE name space:

`/.:/hosts/`*`hostname`*`/ems-server/filters`

The ACL associated with this object controls access to this database. The permissions associated with filters are:

Table 6. Filter Database Permission Bits

Permission bit	Name	Description
c	control	Modify the ACLs on *filters*
d	delete	Delete an event filter
i	insert	Add an event filter
r	read	Get a list of or the contents of event filters

Event filter access control is granted on a per-event-filter basis. Authority on filter access for a given event filter is granted by modifying the ACL on:

`/.:/hosts/`*`hostname`*` /ems-server/filters/`*`filter_name`*

where *filter_name* is the event filter name given the event filter on the call to **ems_filter_add**.

The permissions associated with event filters are:

Table 7. Event Filter Permission Bits

Permission bit	Name	Description
c	control	Modify the ACL on the event filter.
d	delete	Delete the event type filter
w	write	Modify the contents of an event filter

When a consumer creates an event filter, that consumer principal automatically receives **dwc** permissions on the created event filter.

Consumer Security Management

The **Consumer** database is represented by the following name in the DCE name space:

`/.:/hosts/`*`hostname`*`/ems-server/consumers`

The ACL associated with this object controls access to this database. The permissions associated with consumers are:

Table 8. Consumer Database Permission Bits

Permission bit	Name	Description
c	control	Modify the ACLs on consumers
d	delete	Delete a consumer
i	insert	Add (register) a consumer
r	read	List consumer information
w	write	Modify a consumer including his filter group

EMS Security Initialization

When EMS is configured, several security groups are created by default. The groups are **ems-admin**, **ems-consumer**, and **ems-supplier**. The default permissions are:

```
 /.:/hosts/hostname /ems-server
 object acl
           ems-admin              crws
           hosts/hostname/self    rws
           any_other              r

/.:/hosts/hostname/ems-server/event-types
object acl
           ems-admin              cri
           ems-consumer           r
           ems-supplier           ri
           any_other              r

 initial object acl (/.:/hosts/hostname/ems-server/event-types/event_type_name)
           ems-admin              cdw
           ems-consumer           r
           ems-supplier           w

/.:/hosts/hostname/ems-server/filters
object acl
           ems-admin              crdi
           ems-consumer           ir
           any_other              r

initial object acl (/.:/hosts/hostname/ems-server/filters/filter_name)
           ems-admin              cdw

/.:/hosts/hostname/ems-server/consumers
object acl
           ems-admin              cdrw
           ems-consumer           irwd
           any_other              r
```

Because these permissions are set for the **ems_admin** group, each new event filter and event type created automatically inherits the same permissions.

Administrators can add principals to each of these groups to give them access to all EMDs running in a cell. If tighter security is desired, the group can be removed from the respective ACL and principals can be added.

Event Management Service Commands

These commands are issued in the dcecp environment.

- ems catalog
- ems help
- ems operations
- ems show
- emsconsumer commands
- emsconsumer catalog
- emsconsumer delete
- emsconsumer help
- emsconsumer modify
- emsconsumer operations
- emsconsumer show
- emsevent commands
- emsevent catalog
- emsevent delete
- emsevent help
- emsevent operations
- emsevent show
- emsfilter commands
- emsfilter catalog
- emsfilter delete
- emsfilter help
- emsfilter operations
- emsfilter show
- emslog commands
- emslog help
- emslog operations
- emslog show
- emsd

ems commands

Purpose

Manage the EMS daemon on a DCE host.

Format

ems catalog

ems help [*operation* | -verbose]

ems operations

ems show [-host *dce_hostname*]

Argument

operation
: The name of one specific **ems** operation (subcommand) you want to see help information about.

Attributes

eventlog_dir
: Specifies the directory name used where the EMS daemon puts the event log.

queue_size
: Specifies the queue size for the event queues.

Usage

The **ems** object represents the EMS daemon (called emsd) on a host.

This command operates on the EMS daemon on the local host, unless the **-host** option is specified. The format of the host name accepted is either an entire DCE name (/.:/hosts/jurassic.austin.ibm.com) or a host name with domain name (jurassic.austin.ibm.com).

Related Information

Commands:

ems catalog

ems help

ems operations

ems show

emsconsumer commands

emsevent commands

ems catalog

Purpose

Returns the list of all hosts that the EMS daemon is running on in the current cell.

Format

ems catalog

Usage

The **ems catalog** command returns the list of all hosts that the EMS daemon is running on in the current cell.

Privilege Required

No special privileges are needed to use the ems catalog command.

Examples

```
dcecp> ems catalog
/.:/hosts/eagle.austin.ibm.com
/.:/hosts/umesh.austin.ibm.com
```

Related Information

Commands:

- ems help
- ems operations
- ems show
- emsconsumer commands
- emsevent commands

ems help

Purpose

Returns help information on the object.

Format

ems help [*operation* | **-verbose]**

Argument

operation
: The name of one specific **ems** operation (subcommand) you want to see help information about.

Options

-verbose
: Displays detailed information about the DCE Event Management Services object.

Usage

The **ems help** command returns help information on the object. The help operation takes an argument, which may be an operation supported by the object or the **-verbose** switch to return more information.

Privilege Required

No special privileges are needed to use the **ems help** command.

Examples

```
dcecp> ems help
catalog      Returns a list of all hosts that the EMS daemon is running on
help         Prints a summary of command-line options
operations   Returns the valid operations for command
show         Returns the attributes for the EMS daemon
```

Related Information

Commands:

- ems catalog
- ems operations
- ems show
- emsconsumer commands
- emsevent commands

ems operations

Purpose

Returns a list of operations supported by the object.

Format

ems operations

Usage

The **ems operations** command returns a list of operations supported by the object. It takes no arguments, and always returns a TCL list suitable for use in a **foreach** statement. The order of the elements is alphabetical with the exception that help and operations are listed last.

Privilege Required

No special privileges are needed to use the **ems operations** command.

Examples

```
dcecp> ems operations
catalog show help operations
```

Related Information

Commands:

- ems catalog
- ems help
- ems show
- emsconsumer commands
- emsevent commands

ems show

Purpose

Returns the attribute list for the EMS daemon.

Format

ems show [**-host** *dce_hostname*]

Options

-host *dce_hostname*
: Specifies the host where the EMS daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

  Note: The DCE host name is case-sensitive.

Attributes

eventlog_dir
: Specifies the directory name used where the EMS daemon puts the event log

queue_size
: Specifies the queue size for the event queues

Usage

The **ems show** command returns the attribute list for the EMS daemon.

Privilege Required

You must have read (r) permission on /.:/hosts/<*dce_hostname*>/ems-server

Examples

```
dcecp> ems show
{eventlog_dir /opt/dcelocal/dce/var/ems}
{queue_size 5000}
```

Related Information

Commands:

- ems catalog
- ems help
- ems operations
- emsconsumer commands
- emsevent commands

emsconsumer commands

Purpose

Manage EMS consumers and their event filter groups.

Format

emsconsumer catalog [-host *dce_hostname*]

emsconsumer delete *consumer* {-uuid *uuid*} [-host *dce_hostname*]

emsconsumer help [*operation* | -verbose]

emsconsumer modify *consumer* {-uuid *uuid*} {-add|-delete} {filter *filtername*} [-host *dce_hostname*]

emsconsumer operations

emsconsumer show *consumer* {-uuid *uuid*} [-host *dce_hostname*]

Argument

consumer
: A consumer name.

operation
: The name of one specific **emsconsumer** operation (subcommand) that you want to see help information about.

Usage

The emsconsumer object represents an EMS consumer. An EMS consumer registers with EMS to receive event data. It defines event filters to identify the events that should be forwarded to it.

This command operates on the EMS daemon on the local host, unless the **-host** option is specified. The format of the host name accepted is either an entire DCE name (/.:/hosts/jurassic.austin.ibm.com) or a host name with domain name (jurassic.austin.ibm.com).

Related Information

Commands:

emsconsumer catalog

emsconsumer delete

emsconsumer help

emsconsumer modify

emsconsumer operations

emsconsumer show

emsfilter commands

emsconsumer catalog

Purpose

Returns the list of registered consumers with EMS on a host.

Format

emsconsumer catalog [-host *dce_hostname***]**

Options

-host *dce_hostname*

Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

Usage

The **emsconsumer catalog** command returns the list of registered consumers with EMS on a host. The consumer names returned are in an arbitrary order.

Privilege Required

You must have read (r) permission on /.:/hosts/<*dce_hostname*>/ems-server/consumers

Examples

```
dcecp> emsconsumer catalog
{consumer1 7e383761-f41f-11ce-9051-08005acd43c6 /.:/hosts/eagle.austin.ibm.com}
{consumer1 a4c7ff26-f449-11ce-a863-10005a4f3556 /.:/hosts/eagle.austin.ibm.com}
{consumer2 283cc40c-f447-11ce-9dd3-10005a4f3556 /.:/hosts/umesh.austin.ibm.com}
```

Related Information

Commands:

- emsconsumer commands
- emsconsumer delete
- emsconsumer help
- emsconsumer modify
- emsconsumer operations
- emsconsumer show
- emsfilter commands

emsconsumer delete

Purpose

Deletes a registered consumer from EMS on a host.

Format

emsconsumer delete *consumer* {-uuid *uuid*} [-host *dce_hostname*]

Argument

consumer
A consumer name.

Options

-host *dce_hostname*
Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

-uuid *uuid*
Specifies the unique universal identifier (UUID) that is assigned to the consumer.

Usage

The **emsconsumer delete** command deletes a registered consumer from EMS on a host. The argument is the name of the consumer to be deleted or its assigned UUID or, in case of duplicate consumers, a consumer name with its assigned UUID. The command returns an empty string on success.

Privilege Required

You must have delete (d) permission on /.:/hosts/<*dce_hostname*>/ems-server/consumers

Examples

```
dcecp> emsconsumer delete consumer2
dcecp> emsconsumer delete consumer1 -uuid 7e383761-f41f-11ce-9051-08005acd43c6
```

Related Information

Commands:

- emsconsumer commands
- emsconsumer catalog
- emsconsumer help
- emsconsumer modify
- emsconsumer operations
- emsconsumer show
- emsfilter commands

emsconsumer help

Purpose

Returns help information on the object.

Format

emsconsumer help [*operation* **| -verbose]**

Argument

operation
The name of one specific **emsconsumer** operation (subcommand) you want to see help information about.

Options

-verbose
Returns detailed information about the emsconsumer commands object.

Usage

The **emsconsumer help** command returns help information on the object. The help operation takes an argument, which may be an operation supported by the object or the **-verbose** switch to return more information.

Privilege Required

No special privileges are needed to use the emsconsumer help command.

Examples

```
dcecp> emsconsumer help
catalog     Returns the list of registered consumers with EMS on a host.
delete      Deletes a registered consumer from EMS on a host.
modify      Modifies the event filter group associated with a consumer.
show        Shows information about a consumer including the consumers
            filter group.
help        Prints a summary of command-line options.
operations  Returns the valid operations for command.
```

Related Information

Commands:

- emsconsumer commands
- emsconsumer catalog
- emsconsumer delete
- emsconsumer modify
- emsconsumer operations
- emsconsumer show
- emsfilter commands

emsconsumer modify

Purpose

Modifies the event filter group associated with the given consumer.

Format

emsconsumer modify *consumer* {-uuid *uuid*} {-add|-delete} {filter *filtername*} [-host *dce_hostname*]

Argument

consumer
A consumer name.

Options

-add|-delete
Adds or deletes filternames from the consumer filter group.

filter *filtername*
Specifies the name of the consumer filter group.

-host *dce_hostname*
Specifies the host where the EMS daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

-uuid *uuid*
Specifies the unique universal identifier (UUID) that is assigned to the consumer.

Usage

The **emsconsumer modify** command modifies the event filter group associated with the given consumer. Filters can be added or deleted from a consumer event filter group. Added filters are always placed at the end of the consumer event filter group. The command returns an empty string on success.

Privilege Required

You must have write (w) permission on

/.:/hosts/<*dce_hostname*>/ems-server/consumers

Examples

```
dcecp> emsconsumer modify consumer2 -add {filter foo}
```

Argument

Commands:

emsconsumer commands
emsconsumer catalog
emsconsumer delete
emsconsumer help
emsconsumer operations

emsconsumer show
emsfilter commands

emsconsumer operations

Returns a list of operations supported by the object.

Format

emsconsumer operations

Usage

The **emsconsumer operations** command returns a list of operations supported by the object. It takes no arguments and always returns a TCL list suitable for use in a 'foreach' statement. The order of the elements is alphabetical with the exception that help and operations are listed last.

Privilege Required

No special privileges are needed to use the **emsconsumer operations** command.

Examples

```
dcecp> emsconsumer operations
catalog delete modify show help operations
```

Related Information

Commands:

- emsconsumer commands
- emsconsumer catalog
- emsconsumer delete
- emsconsumer help
- emsconsumer modify
- emsconsumer show
- emsfilter commands

emsconsumer show

Purpose

Returns the list of filter names in a consumer filter group.

Format

emsconsumer show *consumer* {-uuid *uuid*} [-host *dce_hostname*]

Argument

consumer
A consumer name.

Options

-host *dce_hostname*
Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

-uuid *uuid*
Specifies the unique universal identifier (UUID) that is assigned to the consumer.

Usage

The **emsconsumer show** command returns the list of filter names in a consumers filter group. This command takes the consumer name as an argument.

Privilege Required

You must have read (r) permission on

/.:/hosts/<*dce_hostname*>/ems-server/consumers

Examples

```
dcecp> emsconsumer show consumer2
{foo2 foo3 foo4 foo5}
```

Related Information

Commands:

- emsconsumer commands
- emsconsumer catalog
- emsconsumer delete
- emsconsumer help
- emsconsumer modify
- emsconsumer operations
- emsfilter commands

emsevent commands

Purpose

Display EMS event types and event type schemas.

Format

emsevent catalog [-host *dce_hostname*]

emsevent delete *event_type_name* [-host *dce_hostname*]

emsevent help [*operation* | -verbose]

emsevent operations

emsevent show *event_type* [-host *dce_hostname*]

Argument

event_type
: Name of the event type.

operation
: The name of one specific **emsevent** operation (subcommand) that you want to see help information about.

Usage

The emsevent object represents the EMS event type, which is a class of events with the same format. This format of the event types are defined by event type schemas. An event type schema consists of a list of attribute name-type pairs that specify the data format of an event.

This command allows for the list of available event types to be displayed, and the event type schema for a particular event type. It operates on the EMS daemon on the local host, unless the **-host** option is specified. The format of the host name accepted is either an entire DCE name (/.:/hosts/jurassic.austin.ibm.com) or a host name with domain name (jurassic.austin.ibm.com).

Related Information

Commands:

emsevent catalog

emsevent delete

emsevent help

emsevent operations

emsevent show

emsevent catalog

Purpose

Returns the list of available event types.

Format

emsevent catalog [-host *dce_hostname*]

Options

-host *dce_hostname*

Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

Usage

The **emsevent catalog** command displays a list of the available event types.

Privilege Required

You must have read (r) permission on /.:/hosts/<*dce_hostname*>/ems-server/event-types.

Examples

```
dcecp> emsevent catalog
SVC
```

Related Information

Commands:

- emsevent commands
- emsevent delete
- emsevent help
- emsevent operations
- emsevent show

emsevent delete

Purpose

Deletes an event type.

Format

emsevent delete event_type_name [-host *dce_hostname*]

Argument

event_type_name
An event type name.

Options

-host dce_hostname
Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

Usage

The **emsevent delete** command deletes an event type. The argument is the name of the event type to be deleted. The command returns an empty string if successful.

Privilege Required

You must have delete permission on:

```
/.:/hosts/<dce_hostname>/ems-server/event-types
   or
/.:/hosts/<dce_hostname>/ems-server/event-types/<event_type_name>
```

Examples

```
dcecp> emsevent delete EventType
dcecp>
```

Related Information

Commands:

- emsevent commands
- emsevent catalog
- emsevent help
- emsevent operations
- emsevent show

emsevent help

Purpose

Returns help information on the object.

Format

emsevent help [*operation* **| -verbose]**

Argument

operation
The name of one specific **emsevent** operation (subcommand) you want to see help information about.

Options

-verbose
Returns detailed information about the **emsevent help** object.

Usage

The **emsevent help** command returns help information on the object. The help operation takes an argument, which may be an operation supported by the object or the **-verbose** switch to return more information.

Privilege Required

No special privileges are needed to use the emsevent help command.

Examples

```
dcecp> emsevent help
catalog     Returns the list of available event types.
delete      Deletes an event type.
help        Prints a summary of command-line options.
operations  Returns the valid operations for command.
show        Returns the event type schema for a event type.
```

Related Information

Commands:

- emsevent commands
- emsevent catalog
- emsevent delete
- emsevent operations
- emsevent show

emsevent operations

Purpose

Returns a list of operations supported by the object.

Format

emsevent operations

Usage

The **emsevent operations** command returns a list of operations supported by the object. It takes no arguments and always returns a TCL list suitable for use in a 'foreach' statement. The order of the elements is alphabetical with the exception that help and operations are listed last.

Privilege Required

No special privileges are needed to use the emsevent operations command.

Examples

```
dcecp> emsevent operations
catalog delete show help operations
```

Related Information

Commands:

- emsevent commands
- emsevent catalog
- emsevent delete
- emsevent help
- emsevent show

emsevent show

Purpose

Returns the event type schema for a event type.

Format

emsevent show *event_type* **[-host** *dce_hostname***]**

Argument

event_type
Name of the event type.

Options

-host *dce_hostname*
Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

Usage

The **emsevent show** command returns the event type schema for a event type. A list of attribute name-type pairs is displayed.

Privilege Required

You must have read (r) permission on /.:/hosts/<*dce_hostname*>/ems-server/event-types/<event_type>

Examples

```
dcecp> emsevent show SVC
        {version ems_c_attr_ulong_int}
        {t ems_c_attr_utc}
        {argtypes ems_c_attr_char_string}
        {table index ems_c_attr_ulong_int}
        {attributes ems_c_attr_ulong_int}
        {message index ems_c_attr_ulong_int}
        {format ems_c_attr_char_string}
        {file ems_c_attr_char_string}
        {progname ems_c_attr_char_string}
        {line ems_c_attr_ulong_int}
        {threadid ems_c_attr_ulong_int}
        {component name ems_c_attr_char_string}
        {sc_name ems_c_attr_char_string}
        {attribute.debug ems_c_attr_ushort_int}
        {attribute.severity ems_c_attr_ushort_int}
        {attribute.actroute ems_c_attr_ulong_int}
```

Related Information

Commands:

emsevent commands
emsevent catalog
emsevent delete
emsevent help

emsevent operations

emsfilter commands

Purpose

Manages EMS event filters on a DCE host.

Format

emsfilter catalog [-host *dce_hostname*]

emsfilter delete *filtername [-host dce_hostname*]

emsfilter help [*operation* | -verbose]

emsfilter operations

emsfilter show *filtername* [-host *dce_hostname*]

Argument

filtername
: A filter name.

operation
: The name of one specific **emsfilter** operation (subcommand) you want to see help information about.

Usage

The emsfilter object represents EMS event filters that are kept by the EMS daemon. The EMS event filters are applied by EMS to events received from suppliers to determine if the events are to be forwarded on to the consumers.

An EMS event filter is a collection of one or more filter expressions. Each filter expression consists of an attribute name, an attribute operator, and an attribute value.

This command operates on the EMS daemon on the local host, unless the -host option is specified. The format of the host name accepted is either an entire DCE name (/.:/hosts/jurassic.austin.ibm.com) or a host name with domain name (jurassic.austin.ibm.com).

Related Information

Commands:

emsconsumer commands

emsfilter catalog

emsfilter delete

emsfilter help

emsfilter operations

emsfilter show

emsfilter catalog

Purpose

Returns a list of names of all filters from EMS on a host.

Format

emsfilter catalog [-host *dce_hostname***]**

Options

-host *dce_hostname*

Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

Usage

The **emsfilter catalog** command returns a list of names of all filters from EMS on a host. The filter names returned are in alphabetical order and not in the order received by EMS.

Privilege Required

You must have read (r) permission on /.:/hosts/<*dce_hostname*>/ems-server/filters.

Examples

In the following example, there are two filters kept by the EMS daemon:

```
dcecp> emsfilter catalog
Filter1
Filter2
```

Related Information

Commands:

- emsconsumer commands
- emsfilter commands
- emsfilter delete
- emsfilter help
- emsfilter operations
- emsfilter show

emsfilter delete

Purpose

Deletes a filter and its associated filter expressions.

Format

emsfilter delete *filtername* **[-host** *dce_hostname***]**

Argument

filtername
A filter name.

Options

-host *dce_hostname*
Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

Usage

The **emsfilter delete** command deletes a filter and its associated filter expressions. The argument is an event filtername to be deleted. If the filter to be deleted is currently being used by at least one consumer, it cannot be deleted and an error message is displayed. The command returns an empty string on success.

Privilege Required

You must have delete (d) permission on /.:/hosts/<*dce_hostname*>/ems-server/filters/<filtername>.

Examples

```
dcecp> emsfilter delete Filter1
dcecp>
```

Related Information

Commands:

- emsconsumer commands
- emsfilter commands
- emsfilter catalog
- emsfilter help
- emsfilter operations
- emsfilter show

emsfilter help

Purpose

Returns help information on the object.

Format

emsfilter help [*operation* **| -verbose]**

Argument

operation

The name of one specific **emsfilter** operation (subcommand) you want to see help information about.

Options

-verbose

Returns detailed information about the emsfilter commandsobject.

Usage

The **emsfilter help** command returns help information on the object. The help operation takes an argument, which may be an operation supported by the object or the -verbose switch to return more information.

Privilege Required

No special privileges are needed to use the emsfilter help command.

Examples

```
dcecp> emsfilter help
catalog      Returns a list of names of all filters from EMS on a host.
delete       Deletes a filter and its associated filter expressions.
help         Prints a summary of command-line options.
operations   Returns the valid operations for command.
show         Returns a list of filter expressions in a specified filter.
```

Related Information

Commands:

- emsconsumer commands
- emsfilter commands
- emsfilter catalog
- emsfilter delete
- emsfilter operations
- emsfilter show

emsfilter operations

Purpose

Returns a list of operations supported by the object.

Format

emsfilter operations

Usage

The **emsfilter operations** command returns a list of operations supported by the object. It takes no arguments, and always returns a TCL list suitable for use in a 'foreach' statement. The order of the elements is alphabetical with the exception that help and operations are listed last.

Privilege Required

No special privileges are needed to use the emsfilter operations command.

Examples

```
dcecp> emsfilter operations
catalog delete show help operations
```

Related Information

Commands:

- emsconsumer commands
- emsfilter commands
- emsfilter catalog
- emsfilter delete
- emsfilter help
- emsfilter show

emsfilter show

Purpose

Returns a list of filter expressions in a specified filter.

Format

emsfilter show *filtername* **[-host** *dce_hostname***]**

Argument

filtername
A filter name.

Options

-host *dce_hostname*
Specifies the host where the EMS Daemon is running. The format of the host name is either an entire DCE name or a host name with a domain name.

Note: The DCE host name is case-sensitive.

Usage

The **emsfilter show** command returns a list of filter expressions in a specified filter. The argument is a filter name to be shown.

Privilege Required

You must have read (r) permission on /.:/hosts/<*dce_hostname*>/ems-server/filters.

Examples

```
dcecp> emsfilter show Filter2
{event_type == SVC}
{file == file.c}
```

Related Information

Commands:

emsconsumer commands
emsfilter commands
emsfilter catalog
emsfilter delete
emsfilter help
emsfilter show

emslog commands

Purpose

Manage EMS log files on the current host.

Format

emslog help [*operation* | -verbose]

emslog operations

emslog show [-dir *directory*] [**-to** *file*]

Argument

operation

The name of one specific **emslog** operation (subcommand) you want to see help information about.

Usage

The emslog object represents the EMS event log, which is used to store events in case of failures of the EMS daemon. The EMS daemon writes all events to the event log and deletes the event record after the event has been transmitted to all the consumers designated to get the event.

The event log is kept in a file on the machine where EMS daemon is running. This command operates on the EMS daemon on the local host.

Related Information

Commands:

emsevent commands

emslog help

emslog operations

emslog show

emslog help

Purpose

Returns help information on the object.

Format

emslog help [*operation* **| -verbose]**

Argument

operation
The name of one specific **emslog** operation (subcommand) you want to see help information about.

Options

-verbose
Displays information about the emslog commands object.

Usage

The **emslog help** command returns help information on the object. The help operation takes an argument, which may be an operation supported by the object or the -verbose switch to return more information.

Privilege Required

No special privileges are needed to use the **emslog** help command.

Examples

```
dcecp> emslog help
help         Prints a summary of command-line options.
operations   Returns the valid operations for command.
show         Returns a list of events in the event log file.
```

Related Information

Commands:

- emsevent commands
- emslog commands
- emslog operations
- emslog show

emslog operations

Purpose

Returns a list of operations supported by the object.

Format

emslog operations

Usage

The **emslog operations** command returns a list of operations supported by the object. It takes no arguments, and always returns a TCL list suitable for use in a ‘foreach’ statement. The order of the elements is alphabetical with the exception that help and operations are listed last.

Privilege Required

No special privileges are needed to use the **emslog operations** command.

Examples

```
dcecp> emslog operations
show help operations
```

Related Information

Commands:

- emsevent commands
- emslog commands
- emslog help
- emslog show

emslog show

Purpose

Returns a list of events in the event log file.

Format

emslog show [-dir *directory***] [-to** *file*]

Options

-dir *directory*
Specifies the directory where the log file is stored.

-to *file* Specifies the name of the file that the output is captured into.

Usage

The **emslog show** command returns a list of events in the event log file.

Privilege Required

No special privileges are needed to use the **emslog show** command.

Examples

```
dcecp> emslog show
--- Start of an EMS event record ---
Type: SVC:Event Id: 8d1b0b00-e9e7-11ce-8af3-10005a890435
Name Service: DCE /.../eagle_dce/hosts.hidalgod.austin.ibm.com
Description Name: EMS_Test_Producer
PID: 565 UID: 0 GID: 0
Severity: NOTICE
Arrival Time: 1995-09-08-14:06:32.970+00:00I-----
Printing 16 items
Item 1: [version] = ulong init 1
Item 2: [t] = 1995-09-08-14:06:32.970+00:00I-----
Item 3: [argtypes] = char string
Item 4: [table_index] = ulong int 0
Item 5: [attributes] = ulong int 64
Item 6: [message_index] = unlon int 389738500
Item 7: [format] = char string Test Supplier starting
Item 8: [file] = char string supplier.c
Item 9: [progname] char string EMS_Test_Producer
Item 10: [line] = ulong int 63
Item 11: [threadid] = ulong int 2
Item 12: [component_name] = char string sup
Item 13: [sc_name] = char string general
Item 14: [attribute.debug] = ushort int 0
Item 15: [attribute.severity] = ushort int 4
Item 16: [attribute.actroute] = ulong int 0
--- End of an EMS event record ---
```

Related Information

Commands:

- emsevent commands
- emslog commands
- emslog help
- emslog show

emsd

Purpose

Starts the DCE Event Management Services Daemon.

Format

emsd [**-l** *log_directory*] [**-q** *queue_size*]
[**-w** *svc_route*...**-w**
svc_route]

Options

-l *log_directory*
: Specifies where the log file resides.

-q *queue_size*
: Specifies the maximum number of events that are queued by EMS. The default size is 512. This value can also be set by setting the EMS_QUEUE_SIZE environment variable. Specifying the **-q** option overrides the environment variable setting.

-w *svc_route*
: Specifies DCE serviceability routing instructions.

Usage

The **emsd** command starts the Event Management Service (EMS) daemon. An EMS daemon must be running in the DCE cell before a consumer can receive events or a supplier can supply events. The EMS daemon runs under the local host machine principal identity (**host/***dce_hostname***/self**). A DCE Host daemon (**dced**) must be running on the local host when **emsd** is started. The **emsd** command also requires a **cdsadvertiser**.

Privilege Required

No special privileges are needed to use the **emsd** command.

Examples

```
emsd -q 2048 -l /opt/dcelocal/var/ems emsd -w NOTICE:STDOUT:-
-w NOTICE_VERBOSE:STDOUT-:-
```

DCE Event Management Service API

The DCE Event Management Service (EMS) manages event services in a DCE cell. EMS consists of three parts:

- **The event supplier interface**

  Provides support for suppliers. A supplier can be any DCE-based user application that emits event data.
- **The EMS daemon (emsd)**

  Performs the following tasks:
  - Authenticating and authorizing event suppliers and consumers
  - Keeping databases of event types, event filters, and consumers
  - Associating an event filter group with each event consumer
  - Ensuring reliable delivery of events to interested consumers
- **The event consumer interface**

  Provides support for the steps required to implement an event consumer. An event consumer performs the following tasks:
  - Query EMS for supported event types
  - Get a list of existing filter names
  - Construct event filters for each event type
  - Add event filters to its event filter group

  Note: The event consumer must be registered with EMS and must set up event filter groups before it can receive events.

The EMS API provides the following structures and interfaces:

EMS Data Structures

EMS Registration Routines

EMS Event Type Routines

EMS Supplier Routine

EMS Event Filter Routines

EMS Consumer Routines

EMS Management Routines

EMS Data Structures

The data structures for EMS are grouped by function. The groups include the following:

- "EMS Event Attributes" on page 190
- "EMS Event Structure" on page 191
- "Event Types" on page 194
- "EMS Event Filters" on page 194
- "EMS Consumer Data Structures" on page 197
- "EMS Server Data Structure" on page 197

EMS Event Attributes

ems_attr_type_t

An **unsigned16** integer that is used to specify the data type of an event attribute. The attribute type specifies the format of the data in the event attribute value union (**ems_attr_value_t**). An event attribute type can be one of those in the following table:

Table 9. Consumer Database Permission Bits

Attribute Type	Data Type	Tagged Union Field Name
ems_c_attr_small_int	idl_small_int	small_int
ems_c_attr_short_int	idl_short_int	short_int
ems_c_attr_long_int	idl_long_int	long_int
ems_c_attr_hyper_int	idl_hyper_int	hyper_int
ems_c_attr_usmail_int	idl_usmall_int	usmall_int
ems_c_attr_ushort_int	idl_ushort_int	ushort_int
ems_c_attr_ulong_int	idl_ulong_int	ulong_int
ems_c_attr_uhyper_int	idl_uhyper_int	uhyper_int
ems_c_attr_short_float	idl_short_float	short_float
ems_c_attr_long_float	idl_long_float	long_float
ems_c_attr_boolean	idl_boolean	bool
ems_c_attr_uuid	uuid_t	uuid
ems_c_attr_utc	utc_t *	utc
ems_c_attr_severity	ems_severity_t	severity
ems_c_attr_acl	sec_acl_t*	acl
ems_c_attr_byte_string	idl_byte*	byte_string
ems_c_attr_char_string	idl_char*	char_string
ems_c_attr_bytes	ems_bytes_t	bytes

Byte strings and character strings are terminated with a 0 (zero) byte. The pickling service of the IDL compiler can be used to encode complex data types into byte strings that are to be included in an EMS event.

ems_bytes_t

A data type to define data stored as bytes. This type contains two fields:

size An integer of type **unsigned32** that indicates the size of the byte data.

data The byte data.

ems_attr_value_t

A self-defining data structure that has an attribute type specifier (*format*) that tells what type of data is in the union, and then appropriate union members to hold the value of the data specified. The *format* field is of type **ems_attr_type_t**, and can contain only one of the tagged union fields described in Table 8 on page 156.

ems_attribute_t

A structure that contains an event attribute name-type pair that defines an event attribute. The **ems_event_t** data type contains an array of **ems_attribute_t** structures. Event attributes can be used in defining the

event types in event type schemas, and in defining event filters in event filter expressions. The **ems_attribute_t** data type contains two fields:

attr_name
A name of type **ems_string** that specifies the attribute name.

attr_type
A value of type **ems_attr_value_t** that specifies the format of the attribute value.

EMS Event Structure

The following data types define an event:

ems_event_type_t.
A variable that defines the type of event. Events can have one of two default types:

ems_c_generic_type
Generic

Events of type **generic** do not have event type schemas associated with them, and can only be filtered by expressions with header attributes in them. This is a **uuid_t** data type. To examine the value in this variable, use the **uuid_compare** routine.

ems_c_svc_type
SVC

ems_eventid_t
A structure that contains the unique identifier for an event. The event identifier contains the following fields:

type An event type of **ems_event_type_t**

id An identifier of type **uuid_t** that is unique to a specific event.

ems_netname_t
A structure containing the network name of a given host machine.

ems_nameservice_t
An enumerated data type that specifies the name service that recognizes the given network name. The possible values are:

ems_ns_other
The name service is other than listed.

ems_ns_dns
DNS name service.

ems_ns_dce
DCE CDS name Service, the only value supported in this release.

ems_ns_x500
X500.

ems_ns_nis
NIS.

ems_ns_sna
SNA network.

ems_netaddr_t
A structure that contains the network name. The name is interpreted

according to the name service specified in **ems_nameservice_t**. The **ems_netaddr_t** structure contains the following fields:

len An unsigned short integer containing the length of the address.

name The name, in an appropriate format. The name is of type **ems_octet_t**, and is of length *len*. The **ems_octet_t** data type is **char**.

For a DCE hostname, the following example sets the **ems_netname_t** structure called `netname`:

```
static char * dce_hostname = "/.:/hosts/eagle.austin.ibm.com";
ems_netname_t  netname;

    netname.service = ems_ns_dce;
    netname.netaddr->len = strlen( dce_hostname )+1;
    netname.netaddr->name = (char *)malloc( netname.netaddr->len );
    strcpy( netname.netaddr->name, dce_hostname );
```

ems_origin_t

A structure that indicates where the event originated; that is, the name of the host where the supplier is running, the name of the supplier, and the supplier process identification. These values may not be valid for all hosts. This structure contains the following fields:

netname
The network name of the originator host, of type **ems_netname_t**.

descname
The descriptive name of the supplier, of type **char ***.

pid The process ID of the originator, of type **unsigned32**. This ID is operating system-dependent.

uid The user ID of the originator, of type **unsigned32**. This ID is operating system-dependent.

gid The group ID of the originator, of type **unsigned32**. This ID is operating system-dependent.

ems_severity_t

An enumerated variable that specifies the severity of the event. The names have a one-to-one correspondence to DCE SVC severity attribute values. The event severity can have one of the following values:

ems_sev_info
Information event.

ems_sev_fatal
Fatal event.

ems_sev_error
Alert event.

ems_sev_warning
Warning event.

ems_sev_notice
Notice event.

ems_sev_notice_verbose
Notice Verbose event.

ems_sev_debug
Debug event.

ems_hdr_t

A structure containing the header of the **ems_event_t** data structure. The header contains the following fields:

eventid
The event identifier, of type **ems_eventid_t**.

origin The event origin, of type **ems_origin_t**.

severity
The event severity, of type **ems_severity_t**.

received
A timestamp indicating the time the event was received. This timestamp is of type **utc_t** and is set by the EMS daemon.

delivered
A timestamp indicating the time the event was delivered to the consumer. This timestamp is of type **utc_t** and is set by the consumer.

A set of filter attributes is provided for event header filtering. The following names can be used for the filter attribute in an event filter expression.

Attribute Name
Attribute Type

eventid.id
ems_c_attr_uuid

eventid.type
ems_c_attr_uuid

origin.netname.service
ems_c_attr_ulong

origin.netname.netaddr
ems_c_attr_bytes

origin.descname
ems_c_attr_char_string

origin.pid
ems_c_attr_ulong

origin.uid
ems_c_attr_ulong

origin.gid
ems_c_attr_ulong

severity
ems_c_attr_ulong

received
ems_c_attr_utc

ems_event_t

A structure containing a fixed header and a variable array. The fields are as follows:

header
The event header, a structure of type **ems_hdr_t**.

count An integer of type **unsigned32**, which contains the number of data items in the *item* array.

item An array of size *count*, containing **ems_attribute_t** attributes. Each data item is a self-defining value that contains an attribute type and attribute data.

Event Types

The EMS event type structures are used to define the EMS event types.

ems_event_type_schema_t
A structure that is used to define an event type. The event type schema specifies only the fixed part of an event. Although the fixed part of an event must match the event type schema, the event can have additional attributes that are unnamed in the schema. The **ems_event_type_schema_t** structure contains the following list of attributes:

type A structure of type **ems_event_type_t** containing an event type ID.

name A pointer to a character string that specifies the name of the event type.

size A long integer that contains the number of attributes in the *attribute* array.

attribute
An array of event type attributes of type **ems_attribute_t** describing the format of this event type. This array has *size* elements.

ems_event_type_list_t
A structure that contains a list of event type schemas. The structure contains the following fields:

size A long integer containing the number of event type schemas.

schema
An array of size *size* of type **ems_schema_ptr_t**, which is defined as:

```
typedef [ptr] ems_event_schema_t  *ems_schema_ptr_t;
```

EMS Event Filters

The event filter data structures allow the definition of both event filters, and event filter lists.

ems_attr_op_t
The attribute operator part of an event filter expression. Attribute operators define the boolean operation to perform on the attribute name and attribute value in an event filter expression. The possible attribute operators are:

Attribute Operator
Description

ems_c_attr_op_eq
The *attr_name* is equal (==) to *attr_value*.

ems_c_attr_op_gt
The *attr_name* is greater than (>) *attr_value*.

ems_c_attr_op_lt
The *attr_name* is less than (<) *attr_value*.

ems_c_attr_op_ge
The *attr_name* is greater than or equal (>=) to *attr_value*.

ems_c_attr_op_le
The *attr_name* is less than or equal (<=) to *attr_value*.

ems_c_attr_op_ne
The *attr_name* is not equal (<>) to *attr_value*.

ems_c_attr_op_substr
The *attr_name* contains the string value specified by *attr_value*.

ems_c_attr_op_bitand
The *attr_name* that is bitwise ANDed with *attr_value* is greater than 0.

ems_filter_exp_t
A structure containing an event filter expression. This structure contains the elements that are used to build an event filter. Event filter expressions contain an attribute name, an operator, and a value that define a boolean filter expression. The fields are:

attr_name
A pointer to a character string that contains the attribute name.

attr_operator
An attribute operator of type **ems_attr_op_t**.

attr_value
An attribute value of type **ems_attr_value_t**.

The following table describes the filter operators that are valid with each attribute type.

Table 10. Filter Expression Operator Table

data type	eq	gt	lt	ge	le	ne	bitand	substr
small int	YES	YES	YES	YES	YES	YES		
short int	YES	YES	YES	YES	YES	YES		
long int	YES	YES	YES	YES	YES	YES		
hyper int	YES	YES	YES	YES	YES	YES		
usmall int	YES	YES	YES	YES	YES	YES		
ushort int	YES	YES	YES	YES	YES	YES		
ulong int	YES	YES	YES	YES	YES	YES		
uhyper int	YES	YES	YES	YES	YES	YES		
short float	YES	YES	YES	YES	YES	YES		
long float	YES	YES	YES	YES	YES	YES		
boolean	YES					YES		
uuid	YES	YES	YES	YES	YES	YES		
utc	YES	YES	YES	YES	YES	YES		
severity	YES					YES		
acl	YES					YES		
byte string	YES					YES	YES	
char string	YES	YES	YES	YES	YES	YES		YES
bytes	YES					YES	YES	

ems_filter_exp_list_t

A structure containing a list of event filter expressions. This structure groups filter expressions together in a list to form an ANDed filter expression used to define an event filter. The structure contains the following fields:

size A long integer indicating the number of filter expressions in the *filter_exps* array.

filter_exps
An array of filter expressions of type **ems_filter_exp_t**.

ems_filter_t

An event filter specifies a series of event filter expressions thatare ANDed together to perform a filter operation. The event filter structure contains the following fields:

filter_name
The event filter name, which is entered in the CDS name space. This name is of type **ems_string_t**.

type A structure of type **ems_event_type_t** that contains the type of event filter.

filter_exp_list
A list of filter expressions of type **ems_filter_exp_list_t**.

Filters with an event type of **generic** can only have filter expressions with header attribute names in them. (See the event header attributes listed in **ems_hdr_t**.)

The following example illustrates how to create a filter:

```
/*-------------------------------------------------*
 * Create a filter that specifies all the events  *
 * received between 1 and 2 AM GMT.               *
 *-------------------------------------------------*/
 ems_filter_exp_list_t *     el = (ems_filter_exp_list_t *)
     malloc(sizeof(ems_filter_exp_list_t)+(1*sizeof(ems_filter_exp_t));
 el->size = 0;
 el->filllter_exps[el->size].attr_name = (unsigned char *)"received.tod";
 el->filter_exps[el->size].attr_operator = ems_c_attr_op_le;
 el->filter_exps[el->size].attr_value.format = ems_c_attr_char_string;
 el->filter_exps[el->size].attr_value.tagged_union.char_string = "0200";
 el->size++;
 el->filter_exps[el->size].attr_name = (unsigned char *)"received.tod";
 el->filter_exps[el->size].attr_operator = ems_c_attr_op_gt;
 el->filter_exps[el->size].attr_value.format = ems_c_attr_char_string;
 el->filter_exps[el->size].attr_value.tagged_union.char_string = "0100";
 el->size++;
```

ems_string_t

A pointer to a character string used to describe filter names.

ems_filtername_list_t

A structure containing a list of event filter names. This event filter list contains the following fields:

size A long integer that contains the number of names in the *filter_names* array.

filter_names
An array containing event filter names of type **ems_string_t**.

ems_filter_list_t
A structure that contains an event filter list. The structure contains the following fields:

size A long integer that contains the number of event filters in the *filters* array.

filters An array of pointers to **ems_filter_t** structures that describe filters.

EMS Consumer Data Structures

These data structures make up the Consumer database in EMS.

ems_consumer_t
A structure that defines an EMS consumer. Each consumer has the following fields:

name A character string containing the DCE name of the consumer, which is entered in CDS.

hostname
The DCE host name where the consumer is running, of type **ems_netname_t**.

uuid A **uuid_t** identifier unique to that consumer.

ems_consumer_list_t
A structure that contains a list of consumer entries. The structure has the following fields:

size A long integer containing the number of entries in the *consumer* array.

consumer
An array of **ems_consumer_t** structures that contain consumer information.

EMS Server Data Structure

ems_attrlist_t
The attribute list data structure defines a list of server attributes. Each attribute is a value maintained by an **emsd** server. The attribute list can be used to query those values. The attribute list contains the following fields:

size A long integer describing the number of attributes in the *attr* array.

attr An array of event type attributes of type **ems_attribute_t**.

EMS Registration Routines

The EMS API allows event suppliers and consumers to register with the EMS daemon. The EMS registration step provides a handle that is used for all future EMS operations. The registration step is required for all event suppliers and management applications. The following routines allow suppliers and management applications to register with the EMS daemon:

ems_register
Obtains an EMS handle for future calls to EMS routines.

ems_unregister
Frees the resources obtained by a call to **ems_register**.

EMS Event Type Routines

The EMS API allows event suppliers and consumers to get a list of event types from the EMS daemon. All events processed by the event service have an event type. Event types can be either generic or defined by an event type schema. The formats of EMS event types are defined by event type schemas and are kept in the EMS Event Type database.

A consumer can request a list of supported event types and select the event types it wants to receive by using the event type schemas to construct event filters and to map event data according to attribute names. For example, an event consumer can reconstruct an SVC message by using the attribute names to find the correct data items.

Suppliers use event type schemas to define new event types that they intend to produce. EMS uses the event type schemas to apply event filters to events.

The event service keeps a database of event types that consists of event type schemas. The following routines allow you to manipulate the event types in the event type database:

ems_event_type_free_list
Frees the list of event type schemas.

ems_event_type_get_list
Gets a list of event type schemas from the Event Type Database.

EMS Supplier Routine

The following routine allows event suppliers to send events to the EMS daemon:

ems_supplier_send
Sends an event to EMS.

EMS Event Filter Routines

Filters are the mechanism used by suppliers and consumers to control which events are sent through the event channel. Filtering is applied by the EMS daemon before forwarding events to consumers. The EMS API supports filtering by allowing event suppliers, consumers, and system administrators to manipulate the EMS Event Filter database. The event filter routines are as follows:

ems_filter_add
Adds a filter to the Event Filter Database.

ems_filter_append
Appends filter expressions to the Event Filter Database.

ems_filter_get
Gets the contents of an event filter.

ems_filter_delete
Deletes a filter from the Event Filter Database.

ems_filter_get_namelist
Gets a list of the names of all filters in the Event Filter Database.

ems_filter_free
Frees an event filter.

ems_filter_free_namelist
Frees a list of event filter names.

ems_filter_get_list
Gets a list of all the filters in the Event Filter Database.

ems_filter_free_list
Frees the list of filters.

EMS Consumer Routines

All event consumers must call the EMS event consumer setup routines before receiving EMS events. In DCE terms, EMS event consumers are both clients and servers. The following steps are required to implement an event consumer:

1. Set up as a DCE server.
2. Register an event handler with the EMS daemon to receive events.
3. Register with the EMS daemon.
4. Create filters to control the events forwarded from the daemon.
5. Start listening for events.

The EMS daemon maintains a consumer database to keep track of all registered consumers. Registering and unregistering with the EMS daemon adds and deletes consumers to and from the database.

The following routines set up the consumers using DCE RPC, and set up the event handler routines.

ems_add_filter_to_group
Adds a filter name to a consumers event filter group.

ems_consumer_handler_register
Registers a consumers event handler.

ems_consumer_register
Registers a consumer with EMS.

ems_consumer_start
Starts an event consumer.

ems_consumer_stop
Stops an event consumer.

ems_consumer_unregister
Unregisters a consumer with EMS.

ems_delete_filter_from_group
Deletes a filter name from a consumers event filter group.

ems_get_filter_group
Gets the list of filter names that comprise a consumers event filter group.

EMS Management Routines

The EMS Management interface provides a means to manage various aspects of EMS. Using this interface, applications can manage event consumers, event filters, and the EMS event log. System administrators can also use **dcecp** to manage the same set of resources.

EMS also offers an interface to the EMS event log. This interface allows management applications to manipulate event logs. The log interface is a local interface only, and can only be run on the machine that runs the **emsd** server.

The EMS event log is used to store events in case of EMS failures. EMS writes all events to the event log, and deletes the event record after the event has been transmitted to all consumers designated to receive the event. The event log is kept in a file on the machine where **emsd** is running. Routines are provided to examine local event logs.

ems_mgmt_free_attributes
Frees a list of **emsd** server attributes.

ems_mgmt_free_consumers
Frees a list of consumers obtained from **ems_mgmt_list_consumers**.

ems_mgmt_free_ems
Frees a list of EMS host names obtained from **ems_mgmt_list_ems**.

ems_mgmt_list_ems
Lists all hosts running **emsd**s.

ems_mgmt_list_attributes
Lists attributes for a specific **emsd**.

ems_mgmt_list_consumers
Lists consumers registered with EMS.

ems_mgmt_delete_consumer
Deletes a consumer from the Consumer Database.

ems_mgmt_delete_filter_from_group
Deletes a filter name from a consumers filter group.

ems_mgmt_add_filter_to_group
Adds a filter name to a consumer's filter group.

ems_mgmt_get_filter_group
Gets the list of names in a consumer's filter group.

ems_log_open
Opens an EMS event log.

ems_log_read
Reads events from an EMS event log.

ems_log_close
Closes an EMS event log.

ems_log_rewind
Rewinds an EMS event log.

ems_add_filter_to_group

Purpose: Adds an event filter to a group.

Format:

```
#include <dce/ems.h>

void ems_add_filter_to_group
       (ems_handle_t handle,
       ems_filtername_list_t  *event_filters,
       error_status_t  *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from **ems_consumer_register**.

event_filters
A pointer to a list of one or more event filter names to add to this consumer event filter group. Consumers can use the names of new event filters after building them with the **ems_filter_add** routine or existing filters that can be obtained by using the **ems_filter_get_namelist** routine.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, the routine returns one of the following error codes:

ems_s_insufficient_permission
The routine does not have permission to perform the operation.

ems_s_filter_list_empty
event_filters contains no event filter names.

ems_s_filtername_exists
An event filter in *event_filters* already exists in the consumer event filter group.

ems_s_no_memory
Error allocating memory.

ems_s_invalid_handle
The handle is not valid.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_add_filter_to_group** routine is used by EMS event consumers to add event filter names to a consumer event filter group. This routine can be called multiple times for each consumer.

Permission Required: (**w**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_consumer_handler_register

Purpose: Registers a consumer event handler.

Format:

```
#include <dce/ems.h>

void ems_consumer_handler_register(
        ems_handler_t hfunc,
        error_status_t  *status);
```

Parameters:

Input

hfunc Specifies the name of the event handler function. The handler signature should be:

```
typedef void (*ems_handler_t) (ems_event_t  *event,
        error_status_t  *status);
```

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, the routine returns one of the following error codes:

ems_s_no_memory
Error allocating memory.

ems_s_mutex_init
Error initializing event queue.

ems_s_cond_variable_init
Error initializing event queue.

ems_s_pthread_create
Error initializing event queue.

ems_s_consumer_not_started
Event consumer has not been started.

Usage: The **ems_consumer_handler_register** routine declares the event consumer event handler. The event consumer developer is responsible for providing the handler to process events.

This routine does not make any RPC calls to EMS.

ems_consumer_register

Purpose: Registers a consumer.

Format:

```
#include <dce/ems.h>

void ems_consumer_register(
        ems_netname_t  *dce_hostname,
        ems_filtername_list_t  *filter_group,
        ems_handle_t  *handle,
        error_status_t  *status);        /* register status    */
```

Parameters:

Input

dce_hostname
A pointer to the name of the DCE host machine where **emsd** is running. If the DCE host name is NULL, then the local host is assumed.

Note: *dce_hostname* is case sensitive.

filter_group
A pointer to a list of event filter names that define this consumer initial event filter group. If *filter_group* is empty, no filter group is specified, and EMS doesl not forward any events to this consumer until the consumer makes a call to **ems_add_event_to_group**.

Output

handle Returns an EMS handle that can be used on subsequent calls to EMS routines.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns one of the following status codes:

ems_s_no_memory
Error allocating memory.

ems_no_consumer_handler
Error calling **ems_consumer_register** before an event handler was registered with **ems_consumer_handler_register**.

ems_s_already_registered
Consumer with this name already registered.

ems_s_mutex_init
Error initializing event queue.

ems_s_cond_variable_init
Error initializing event queue.

ems_s_pthread_create
Error initializing event queue.

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_consumer_not_started
Event consumer has not been started.

ems_s_unsupported_nameservice
Nameservice is not supported.

Usage: The **ems_consumer_register** routine is used by EMS event consumers to register with EMS. This routine should be called once for each DCE host that this consumer wants to receive events from. This routine must be called after a call to **ems_consumer_start**.

Permission Required: (**i**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_consumer_start

Purpose: Starts a consumer.

Format:

```
#include <dce/ems.h>

void ems_consumer_start(
        char  *consumer,
        unsigned32  flags,
        error_status_t  *status);
```

Parameters:

Input

consumer
: A pointer to the consumer name. This name must be unique and is registered in the CDS namespace under **/.:/hosts/***dce_hostname***/ems/consumers**. The name is used by the administrative interface to refer to this consumer.

flags Reserved for future use.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_no_memory
: Error allocating memory.

ems_s_consumer_already_started
: *consumer* already started.

ems_s_invalid_name
: A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_consumer_start** routine should be called at the beginning of each event consumer before making any register calls. It creates an object UUID to uniquely identify this event consumer and register its endpoint so that EMS can send this consumer event data.

This routine does not make any RPCs to EMS.

ems_consumer_stop

Purpose: Stops a consumer.

Format:

```
#include <dce/ems.h>

void ems_consumer_stop(
        error_status_t  *status);
```

Parameters:

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns **ems_s_consumer_not_started**, which indicates that the specified event consumer has not been started.

Usage: The **ems_consumer_stop** routine should be called at the end of each event consumer after a call to **ems_s_consumer_start**. It unregisters the endpoint of this event consumer and breaks the thread that was created by the consumer event handler interface to receive all events from EMS.

This routine does not make any RPCs to EMS.

ems_consumer_unregister

Purpose: Unregisters a consumer.

Format:

```
#include <dce/ems.h>

void ems_consumer_unregister(
        ems_handle_t  *handle,
        error_status_t  *status);
```

Parameters:

Input

handle A handle returned from a call to **ems_consumer_register**. This routine frees memory used by *handle* and sets *handle* to NULL.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_unknown_consumer
Tried to unregister a consumer that was not registered.

ems_s_consumer_not_started
Consumer has not been started.

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_invalid_handle
The handle is not valid.

Usage: The **ems_consumer_unregister** routine is used by EMS event consumers to unregister with EMS. This routine should be called once for each call to **ems_consumer_register**. The event consumer should call this routine before calling the **ems_consumer_stop** routine.

Permission Required: (**d**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_delete_filter_from_group

Purpose: Deletes an event filter from a group.

Format:

```
#include <dce/ems.h>

void ems_delete_filter_from_group(
       ems_handle_thandle,
       ems_filtername_list_t *filter_name,
       error_status_t *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from **ems_consumer_register**.

filter_name
A pointer to the event filter names to delete from the consumer's event filter group.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_filter_list_empty
No filter names were specified for deletion.

ems_s_filtername_not_there
Specified filter name to delete not in group.

ems_s_no_memory
Error allocating memory.

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_invalid_handle
The handle is not valid.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_delete_filter_from_group** routine is used by EMS event consumers to delete event filter names from consumer event filter groups.

Permission Required: (**w**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_event_type_add

Purpose: Adds an event type.

Format:

```
#include <dce/ems.h>

void ems_event_type_add(
        ems_handle_t handle,           /* EMS handle */
        ems_event_schema_t * schema,  /* event type schema to add */
        error_status_t * status);        /* request status */
```

Parameters:

Input

handle A handle returned from a call to ems_register call.

schema
An EMS event type schema that describes the format of an event type.

Output

status Returns the status code from this routine, which indicates whether the routine completed successfully or not. Possible status codes include:

error_status_ok
Indicates success.

ems_s_invalid_handle
Handle parameter is not valid.

ems_s_eventtype_exists
The event type already exists.

ems_s_insufficient_permission
The caller does not have permission to perform this operation.

ems_s_invalid_event_type
The event schema is not valid.

Usage: This routine is used by an event supplier to add new event types to the EMS event type Database. A supplier can add a new event type and then start producing that event type by transmitting events to EMS.

Permission Required: **(i)** on /.:/hosts/*hostname*/ems-server/event-types

ems_event_type_delete

Purpose: Deletes an event type.

Format:

```
#include <dce/ems.h>

void ems_event_type_delete(
        ems_handle_t handle,      /* EMS handle */
        char * type_name,           /* event type name */
        error_status_t * status) ;  /* request status */
```

Parameters:

Input

handle A handle returned from a call to ems_register().

type_name
The name of an EMS event type.

Output

status Returns the status code from this routine, which indicates whether the routine completed successfully or not. Possible status codes include:

error_status_ok
Indicates success.

ems_s_invalid_handle
Handle parameter is not valid.

ems_s_insufficient_permission
The caller does not have permission to perform this operation.

ems_s_event_type_not_found
The specified event type was not found.

ems_s_invalid_name
The event type name specified a name that is not valid.

Usage: This routine is used by an event supplier to delete an event type in the EMS event type Database.

Permission Required: **(d)** on /.:/hosts/*hostname*/ems-server/event-types,

or

(d) on /.:/hosts/*hostname*/ems-server/event-types/*type_name*

ems_event_type_free_list

Purpose: Frees an event type list.

Format:

```
#include <dce/ems.h>

void ems_event_type_free_list(
        ems_event_type_list_t  **type_list,
        error_status_t  *status);
```

Parameters:

Input

type_list
A pointer to an event type list as returned by **ems_event_type_get_list**. This routine sets *type_list* to NULL.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**.

Usage: The **ems_event_type_free_list** routine is used by callers of **ems_event_type_get_list** to free the storage used by an event type list.

ems_event_type_get

Purpose: Gets an event type

Format:

```
#include <dce/ems.h>

void ems_event_type_get(
        ems_handle_t handle,            /* EMS handle */
        char * type_name,                /* event type name */
        ems_event_schema_t** schema,  /* event type schema */
        error_status_t * status);       /* request status */
```

Parameters:

Input

handle Should be the handle returned from a call to *ems_consumer_register* call ().

type_name
The event type name to retrieve from the event type database.

Output

schema
Returns the requested event type schema.

status Returns the status code from this routine, which indicates whether the routine completed successfully or not. Possible status codes include:

error_status_ok
Indicates success.

ems_s_invalid_handle
Handle parameter is not valid.

ems_s_insufficient_permission
The caller does not have permission to perform this operation.

ems_s_invalid_name
The event type name specified a name that is not valid.

ems_s_event_type_not_found
The requested event type was not found.

Usage: This routine is used to retrieve event type schemas from the event type Database.

ems_event_type_get_list

Purpose: Gets an event types list.

Format:

```
#include <dce/ems.h>

void ems_event_type_get_list(
        ems_handle_thandle,
        ems_event_type_list_t  **type_list,
        error_status_t  *status);
```

Parameters:

Input

handle The handle returned from a call to **ems_consumer_register**.

Output

type_list
Returns the list of available event types.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_no_type_list
There is no event type list available.

ems_s_no_memory
Error allocating memory.

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_invalid_handle
The handle is not valid.

Usage: The **ems_event_type_get_list** routine is used by EMS event consumers to find out which event types are available to register for. The consumer can then set up filters for attributes in one of the available event types.

Permission Required: (**r**) on /.:/hosts/*dce_hostname*/ ems-server/event-types

ems_filter_add

Purpose: Adds an event filter.

Format:

```
#include <dce/ems.h>

void ems_filter_add(
        ems_handle_thandle,
        ems_string_t  filter_name,
        ems_event_type_t  type,
        ems_filter_exp_list_t  *exp_list
        error_status_t  *status);
```

Parameters:

Input

handle A handle returned from a call to **ems_consumer_register**.

filter_name
Specifies the event filter name for this event filter. This name can be used to add the event filter to a consumer's event filter group.

type Specifies the event type that this filter is applied against.

exp_list
Pointer to a list of filter expressions that are part of the event filter *filter_name*.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise it returns one of the following error codes:

ems_s_filter_exists
The given filter name already exists.

ems_s_invalid_filter
The input parameter specifies an invalid filter.

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_invalid_handle
The handle is not valid.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_filter_add** routine is used to add a new event filter to the EMS Event Filter Database.

Permission Required: (**i**) on /.:/hosts/*dce_hostname*/ ems-server/filters

ems_filter_append

Purpose: Appends to an event filter.

Format:

```
#include <dce/ems.h>

void ems_filter_append(
        ems_handle_t handle,
        ems_string_t  filter_name,
        ems_filter_exp_list_t  *exp_list,
        error_status_t  *status);
```

Parameters:

Input

handle The handle returned from a call to **ems_consumer_register**.

filter_name
Specifies the name of the event filter to add the filter expressions to.

exp_list
A list of filter expressions that are added to the end of event filter *filter_name*.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_filter
The input parameter specifies an invalid filter.

ems_s_filter_not_found
The specified filter was not found.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_handle
The handle is not valid.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_filter_append** routine is used to add filter expressions to an event filter. The filter expressions are added to the end of the current list of filter expressions in the event filter.

Processing: (**w**) on /.:/hosts/*dce_hostname*/ ems-server/filters/*filter_name*

ems_filter_delete

Purpose: Deletes an event filter.

Format:

```
#include <dce/ems.h>

void ems_filter_delete(
        ems_handle_t handle,
        ems_string_t  filter_name,
        error_status_t  *status);
```

Parameters:

Input

handle The handle returned from a call to **ems_consumer_register**.

filter_name
Specifies the name of the event filter to delete.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_filter_not_found
The specified filter name was not found in the filter database.

ems_s_filter_in_use
The specified filter name appears in a consumer's event filter group.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_handle
The handle is not valid.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_filter_delete** routine is used to delete an event filter from the Event Filter Database. The name *filter_name* cannot be contained in any consumer event filter group when this routine is called.

Processing: (**d**) on /.:/hosts/*dce_hostname*/ ems-server/filters/*filter_name*, or (**d**) on /.:/hosts/*dce_hostname*/ ems-server/filters

ems_filter_free

Purpose: Frees an event filter.

Format:

```
#include <dce/ems.h>

void ems_filter_free(
        ems_filter_exp_list_t **exp_list,
        error_status_t  *status);
```

Parameters:

Input

exp_list
A pointer to a list of filter expressions to free.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns an error.

Usage: The **ems_filter_free** routine is used to free a list of event filter expressions obtained by a call to the **ems_filter_get** routine.

ems_filter_free_list

Purpose: Frees an event filter list.

Format:

```
#include <dce/ems.h>

void ems_filter_free_list(
        ems_filter_list_t  **filter_list,
        error_status_t  *status);
```

Parameters:

Input/Output

filter_list
: A pointer to a list of event filters that make up the Event Filter Database as returned by the routine **ems_filter_get_list**. On output, *filter_list* is set to NULL.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**.

Usage: The **ems_filter_free_list** routine is used by callers of **ems_get_event_filter_database** to free the storage used by an Event Filter Database (**ems_filter_db_t**) structure.

ems_filter_free_namelist

Purpose: Frees a list of event filter names.

Format:

```
#include <dce/ems.h>

void ems_filter_free_namelist(
        ems_filtername_list_t  **name_list,
        error_status_t  *status);
```

Parameters:

Input

name_list
A pointer to a list of filter names to free.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns an error.

Usage: The **ems_filter_free_namelist** routine is used to free a list of filter names returned by various routines. The routines that return a list of filter names are:

ems_filter_get_namelist

ems_get_filter_group

ems_mgmt_get_filter_group

ems_filter_get

Purpose: Gets an event filter.

Format:

```
#include <dce/ems.h>

void ems_filter_get(
        ems_handle_t handle,
        ems_string_t filter_name,
        ems_event_type_t  *filter_type,
        ems_filter_exp_list_t  **exp_list,
        error_status_t  *status);
```

Parameters:

Input

handle The handle returned from a call to **ems_consumer_register**.

filter_name
Specifies the name of the event filter to get.

Output

filter_type
Specifies the event type of the filter.

exp_list
A pointer to the list of filter expressions that are part of event filter *filter_name*.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_filter_not_found
The specified filter name was not found in the filter database.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_handle
The handle is not valid.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_filter_get** routine is used to get the filter expressions in an event filter.

Processing: (**r**) on /.:/hosts/*dce_hostname*/ ems-server/filters

ems_filter_get_list

Purpose: Gets event filter list.

Format:

```
#include <dce/ems.h>

void ems_filter_get_list(
        ems_handle_thandle,
        ems_filter_list_t  **filter_list,
        error_status_t  *status);
```

Parameters:

Input

handle A handle returned from a call to **ems_consumer_register**.

Output

filter_list
A pointer to a list of all the event filters in the Event Filter Database.

status A pointer to the completion code. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_empty_filter_db
No filters in database.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_handle
The handle is not valid.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_filter_get_list** routine is used to get a list of the event filters in the Event Filter Database. This list should be freed using **ems_filter_free_list**.

Processing: (**r**) on /.:/hosts/*dce_hostname*/ ems-server/filters/*filter_name*

ems_filter_get_namelist

Purpose: Lists event filter names.

Format:

```
#include <dce/ems.h>

void ems_filter_get_namelist(
        ems_handle_t handle,
        ems_filtername_list_t  **name_list,
        error_status_t  *status);
```

Parameters:

Input

handle The handle returned from a call to **ems_consumer_register**.

Output

name_list
A pointer to a list of all the event filter names in the Event Filter Database. The routine **ems_event_filter_get** can be used to show the contents of each event filter.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_empty_filter_db
No filters in database.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_handle
The handle is not valid.

Usage: The **ems_filter_get_namelist** routine is used to get a list of the names of the event filters in the Event Filter Database.

Processing: (**r**) on /.:/hosts/*dce_hostname*/ ems-server/filters

ems_get_filter_group

Purpose: Gets a filter group.

Format:

```
#include <dce/ems.h>

void ems_get_filter_group(
        ems_handle_t handle,
        ems_filtername_list_t  **filter_group,
        error_status_t  *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from **ems_consumer_register**.

Output

filter_group
A pointer to the list of event filter names that are in the consumer event filter group.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_handle
Handle that is not valid.

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_consumer_not_found
The specified consumer is not registered.

Usage: The **ems_get_filter_group** routine returns a list of event filter names that comprise the consumer event filter group. The requesting consumer must free the storage allocated for *filter_group*.

Processing: (**r**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_log_close

Purpose: Closes the event log.

Format:

```
#include <dce/ems.h>

void ems_log_close(
        ems_log_file_t *handle,
        error_status_t  *status);
```

Parameters:

Input/Output

handle Specifies the event log file to close. On output *handle* is set to NULL.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns **ems_s_invalid_log_handle**, which indicates that an invalid log file handle was passed in.

Usage: The **ems_log_close** routine closes an event log file.

ems_log_open

Purpose: Opens the event log.

Format:

```
#include <dce/ems.h>

void ems_log_open(
        ems_log_file_t *log_file,
        char  *log_dir,
        error_status_t  *status);
```

Parameters:

Input

log_dir Directory where log directory is located. If NULL, the environment variable **EMS_EVENTLOG_DIR** is checked. If **EMS_EVENTLOG_DIR** is not set, the default directory is used.

Output

log_file
Log handle to use in other **ems_log_*** routines.

status A pointer to the completion code. If the routine completes successfully it returns **error_status_ok**. Otherwise it returns one the following error codes:

ems_s_no_event_log
Event log not found.

ems_s_no_log_entries
No event log entries.

ems_s_no_memory
Error allocating memory.

Usage: The **ems_log_open** routine opens an EMS event log and locks the event log database until the **ems_log_close** routine is called.

ems_log_read

Purpose: Reads the event log.

Format:

```
#include <dce/ems.h>

void ems_log_read(
        ems_log_file_t handle,
        ems_event_t  **event,
        error_status_t  *status);
```

Parameters:

Input

handle Specifies the open event log to read from.

Output

event A pointer to the next event in the event log.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_log_handle
Invalid log file handle passed in.

ems_s_no_more_events
No more events to read in log file.

Usage: The **ems_log_read** routine reads an event from the EMS event log.

ems_log_rewind

Purpose: Rewinds the event log.

Format:

```
#include <dce/ems.h>

void ems_log_rewind(
        ems_log_file_t handle,
        error_status_t  *status);
```

Parameters:

Input

handle Specifies the event log file to rewind.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_log_handle
An log file handle that is not valid was passed in.

ems_s_no_log_entries
No event log entries.

Usage: The **ems_log_rewind** routine rewinds an event log. This allows the event log to be rewound to the beginning. This function is equivalent to calling **ems_log_close** and then calling **ems_log_open** again.

ems_mgmt_add_filter_to_group

Purpose: Adds a list of event filter names to an event filter group.

Format:

```
#include <dce/ems.h>

 void ems_mgmt_add_filter_to_group(
        ems_handle_t handle,
        char  *consumer,
        uuid_t  *uuid,
        ems_filtername_list_t  *filter_name,
        error_status_t  *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from the **ems_register** routine.

consumer
Specifies the consumer whose event filter group is being updated.

uuid Specifies the consumer UUID that uniquely identifies the consumer to clear. If this parameter is NULL, then only one consumer can exist with the name *consumer*.

filter_name
Specifies the list of event filter names to add.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_invalid_handle
An invalid handle was passed.

ems_s_consumer_not_found
The specified consumer is not registered.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_mgmt_add_filter_to_group** routine adds event filter names to a consumer event filter group.

Processing: (**i**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_mgmt_delete_consumer

Purpose: Clears the consumers.

Format:

```
#include <dce/ems.h>

void ems_mgmt_delete_consumer(
        ems_handle_thandle,
        char  *consumer,
        uuid_t  *uuid,
        error_status_t  *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from the **ems_register** routine.

consumer
A pointer to the consumer name to clear. This name is the name returned in the **ems_consumer_list_t** data structure after calling **ems_mgmt_list_consumers** or the name used on the **ems_consumer_start** routine.

uuid Specifies the consumer UUID that uniquely identifies the consumer to clear. If this parameter is NULL, only one consumer can exist with the name *consumer*.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_handle
A handle that is not used was passed.

ems_s_consumer_not_found
The specified consumer is not registered.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_mgmt_delete_consumer** routine clears all information stored in EMS about the specified consumer. This means clearing the consumer filter and then unregistering the consumer. The consumer receives notification that it is being deleted.

Processing: (**d**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_mgmt_delete_filter_from_group

Purpose: Deletes the event filter name from the event filter group.

Format:

```
#include <dce/ems.h>

void ems_mgmt_delete_filter_from_group(
        ems_handle_t handle,
        char  *consumer,
        uuid_t  *uuid,
        ems_filtername_list_t  *filter_name,
        error_status_t  *status);
```

Parameters:

Output

handle Must contain a valid consumer handle obtained from the **ems_register** routine.

consumer
A pointer to the consumer whose event filter group is being updated.

uuid A pointer to the consumer UUID that uniquely identifies the consumer to clear. If this parameter is NULL, only one consumer can exist with the name **consumer**.

filter_name
A pointer to the names of the filters to delete from the consumer filter group.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_handle
An invalid handle was passed.

ems_s_consumer_not_found
The specified consumer is not registered.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_mgmt_delete_filter_from_group** routine deletes the specified event filter names from a consumer event filter group.

Processing: (**w**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_mgmt_free_attributes

Purpose: Frees a list of **emsd** server attributes.

Format:

```
#include <dce/ems.h>**

void ems_mgmt_free_attributes(
        ems_attrlist_t  list,
        error_status_t  *status);
```

Parameters:

Input

list A pointer to the list of attributes to free.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns an error.

Usage: The **ems_mgmt_free_attributes** routine frees a list of **emsd** server attributes obtained by the **ems_mgmt_list_attributes** routine.

ems_mgmt_free_consumers

Purpose: Frees a list of consumers obtained from **ems_mgmt_list_consumers**.

Format:

```
#include <dce/ems.h>

void ems_mgmt_free_consumers(
        ems_consumer_list_t  **list,
        error_status_t  *status);
```

Parameters:

Input

list A pointer to a list of consumers to free.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns an error.

Usage: The **ems_mgmt_free_consumers** routine frees a list of consumers obtained from **ems_mgmt_list_consumers**.

ems_mgmt_free_ems

Purpose: Frees a list of hosts obtained from **ems_mgmt_list_ems**.

Format:

```
#include <dce/ems.h>

void ems_mgmt_free_ems(
    char  ***host_list,
    error_status_t  *status);
```

Parameters:

Input

host_list
A pointer to a list of hosts to free.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns an error.

Usage: The **ems_mgmt_free_ems** routine frees a list of consumers obtained from **ems_mgmt_list_ems**.

ems_mgmt_get_filter_group

Purpose: Gets a list of event filter names in an event filter group.

Format:

```
#include <dce/ems.h>

void ems_mgmt_get_filter_group(
        ems_handle_thandle,
        char  *consumer,
        uuid_t  *uuid,
        ems_filtername_list_t  **filter_group,
        error_status_t  *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from the **ems_register** routine.

consumer
A pointer to the consumer event filter group to return. The consumer name is the name given to the **ems_start_consumer** routine or the name returned in the **ems_consumer_list_t** data structure from the routine **ems_mgmt_list_consumers**.

uuid A pointer to the consumer UUID that uniquely identifies the consumer to clear. If this parameter is NULL, only one consumer can exist with the name *consumer*.

Output

filter_group
A pointer to the list of event filter names in the specified consumer's event filter group.

status A pointer to the completion code. On successful completion this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_handle
An invalid handle was passed.

ems_s_consumer_not_found
The specified consumer is not registered.

ems_s_insufficient_permission
No permission for the requested operation.

ems_s_invalid_name
A filter, consumer, or filter attribute name that contains characters that are not valid was specified.

Usage: The **ems_mgmt_get_filter_group** routine returns a list of event filter names in a cnsumer event filter group.

Processing: (**i**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_mgmt_list_attributes

Purpose: Lists **emsd** attributes.

Format:

```
#include <dce/ems.h>

void ems_mgmt_list_attributes(
        ems_handle_t handle,
        ems_attrlist_t  **list,
        error_status_t  *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from the **ems_register** routine.

Output

list A pointer to the list of **emsd** attributes.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_invalid_handle
An invalid handle was passed.

ems_s_no_memory
Error allocating memory.

Usage: The **ems_mgmt_list_attributes** routine lists **emsd** server attributes. Free this list using the **ems_mgmt_free_attributes** routine.

Processing: (**r**) on /.:/hosts/*dce&ushostname*/ems-server

ems_mgmt_list_consumers

Purpose: Lists consumers.

Format:

```
#include <dce/ems.h>

void ems_mgmt_list_consumers(
        ems_handle_t handle,
        ems_consumer_list_t  **list,
        error_status_t  *status);
```

Parameters:

Input

handle Must contain a valid consumer handle obtained from the **ems_register** routine.

Output

list A pointer to the list of consumers.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_no_memory
Error allocating memory.

ems_s_no_consumers
No consumers registered.

ems_s_insufficient_permission
No permission to perform the requested operation.

ems_s_invalid_handle
An invalid handle was passed.

Usage: The **ems_mgmt_list_consumers** routine lists consumers registered with EMS.

Processing: (**r**) on /.:/hosts/*dce_hostname*/ ems-server/consumers

ems_mgmt_list_ems

Purpose: Lists EMS hosts.

Format:

```
#include <dce/ems.h>

void ems_mgmt_list_ems(
        char  ***host_list,
        error_status_t  *status);
```

Parameters:

Output

host_list
A pointer to the list of hosts running **emsd**.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns **ems_no_memory**, which indicates that there was an error allocating memory.

Usage: The **ems_mgmt_list_ems** routine lists hosts running **emsd**. Use **free** to free memory used by *host_list*.

ems_register

Purpose: Registers with EMS.

Format:

```
#include <dce/ems.h>

void ems_register(
        ems_netname_t  *dce_hostname,
        ems_handle_t  *handle,
        error_status_t  *status);
```

Parameters:

Input

dce_hostname
A pointer to the name of the DCE host machine where **emsd** is running. If the DCE host name is NULL, then the local host is assumed.

Note: *dce_hostname* is case sensitive.

Output

handle Returns an EMS handle to use for future calls to EMS routines.

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_no_memory
An EMS handle cannot be allocated.

ems_s_unsupported_nameservice
The *dce_hostname* contains an unsupported name service.

Usage: The **ems_register** routine registers with EMS and obtains an EMS binding handle. This routine can be used by a management application that uses the EMS Management API or by event suppliers that want to add new event types.

ems_supplier_send

Purpose: Sends supplier events to EMS.

Format:

```
#include <dce/ems.h>

void ems_supplier_send(
        ems_handle_t handle,
        ems_event_t *event,
        error_status_t *status);
```

Parameters:

Input

handle Should be the handle returned from a call to **ems_register**.

event A pointer to the event data. For the content of the event messages, see "EMS Data Structures" on page 189.

Output

status Pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns one of the following error codes:

ems_s_invalid_handle
A handle that has not been initialized or that is invalid was used.

ems_s_no_memory
The EMS server received an error allocating memory.

ems_s_insufficient_permission
This supplier does not have permission to supply events.

Usage: The **ems_supplier_send** routine is called by event suppliers to send events to EMS.

Processing: (**w**) on /.:/hosts/*dce_hostname*/ ems-server/event-types/*type_name*

ems_svc_connect_push_supplier

Purpose: Connects a push supplier to an event consumer so it can receive serviceability events.

Format:

```
void
ems_svc_connect_push_supplier(
            char*             bin_log,
            char*             filter_file,
            char*             time_file
            int               time_interval,
            error_status_t*   status)
```

Parameters:

Input

bin_log
Specifies the name of the Serviceability binary log file to monitor.

filter_file
Specifies an event filter file. This file can contain zero or more severity filters and zero or more message identifiers. See "Event Management Using the Direct Supplier/Consumer Model" on page 243 for the correct syntax.

time_file
Specifies a unique timestamp file name.

time_interval
After reaching the end of the *bin_log* file, this API waits for *time_interval* seconds before it checks for any new entries in the log file. If the *time_interval* is set to 0, the API returns immediately.

Output

status A pointer to the completion status. This API never returns unless one of the following error conditions occurs:

ems_s_no_memory
An error occurred while allocating memory.

svc_s_cantopen
Permission is denied or the file does not exist.

svc_s_at_end
No more entries in the *bin_log* file (only when the *time_interval* is 0).

Usage: The **ems_svc_connect_push_supplier** routine is called by event consumers to receive Serviceability (SVC) events previously routed to the **bin_log** file.

This routine operates independently of the EMS daemon, and uses the direct supplier/consumer model. In this model, the consumer sets up the consumer environment. The consumer then calls this API to directly connect to an event supplier that delivers all events from the **bin_log** file.

The event consumer provides the **filter_file** to specify the messages that are of interest. Each event in the binary log file is examined and the filters in the event filter file are applied; if the event passes the filters, it is forwarded to the event consumer.

After each event is processed from the SVC *bin_log* file, the timestamp of that event is recorded in the *time_file* file. If the consumer reinvokes this API, this timestamp is checked to prevent resending of events that were already sent. If multiple consumers are to be used, it is important to specify a unique file name. If a non-unique timestamp file is used, only those events that come in time after the most recent timestamp are sent to all consumers. This results in the loss of some events. If a NULL timestamp file is specified, no record is kept of the events that were already sent, and all events are then sent again.

After all events from the *bin_log* file have been read, this API waits for *time_interval* seconds before it checks the log file for any new entries. After the *time_interval* has passed, the API processes the new entries until it reaches the end of the file again. Therefore, the API never returns. If the user prefers that the API return after the end of the *bin_log* file has been reached, the *time_interval* should be set to 0.

The **ems_consumer_handler_register** routine must be called prior to calling this routine to register an event handler and to enable the processing of events.

ems_unregister

Purpose: Unregisters with EMS.

Format:

```
#include <dce/ems.h>

void ems_unregister(
        ems_handle_t *handle,
        error_status_t *status);
```

Parameters:

Input/Output

handle An EMS handle obtained from the **ems_register** routine. On output the value of *handle* is set to NULL.

Output

status A pointer to the completion status. On successful completion, this routine returns **error_status_ok**. Otherwise, it returns **ems_s_invalid_handle**, which indicates that a handle that is not valid was passed.

Usage: The **ems_unregister** routine unregisters and frees the resources used by an EMS handle. This routine should be called with a handle obtained by the **ems_register** routine.

Event Management Using the Direct Supplier/Consumer Model

The DCE Event Management Service (EMS) supports push suppliers using both an event channel model and the direct push supplier to push consumer model. In the event channel model, all events are sent to the EMS daemon (emsd), which acts as the event channel; it routes events to any registered consumers that are interested in those events. Event transmission between the supplier and the consumer is decoupled by using an event channel. In the direct supplier-consumer model, a connection is made between one supplier and one consumer to send events.

Some system management applications may not want to depend on DCE RPC to deliver Serviceability (SVC) events from supplier to consumer. For those applications, the direct supplier-consumer model is more suitable. A specific API is provided to support the direct supplier-consumer model. In this model, the consumer sets up the consumer environment and then calls the *ems_svc_connect_push_supplier* API to directly connect to an event supplier that delivers all events from a standard DCE SVC binary log file. This event supplier does not use DCE RPC to deliver the events and the EMS daemon (or event channel) is not used in the event delivery process.

The following setup is required to take full advantage of this model:

1. **Routing DCE Serviceability messages to a binary log file**

   DCE Serviceability allows messages to be routed to a binary log file by changing the DCE Serviceability routing file. This routing file resides in /opt/dcelocal/var/svc/routing. The following are examples of two routing file entries:

   ```
   FATAL:BINFILE:/opt/dcelocal/var/svc/bin.log
   ERROR:BINFILE:/opt/dcelocal/var/svc/bin.log
   ```

   Including these lines in the routing file causes all fatal and error messages to be routed to the specified binary log file. To monitor warning, notice, or notice verbose messages, corresponding entries are required.

   In order to enable an event consumer to receive SVC events, all interesting messages must be routed to the same binary file as shown above. The event consumer calls the *ems_svc_connect_push_supplier* API, which monitors the binary file for new messages. When a new message is received, it goes through a filtering mechanism using the Event Filter file (see the next section). If the event passes, the supplier passes the event to the consumer and the appropriate event handler is invoked.

   Note: The preceding setup actually enables all DCE applications using Serviceability to route their FATAL and ERROR messages to the *bin.log* file, which in turn, allows all DCE applications to be monitored.

2. **Using the Event Filter file**

   The Event Filter file is used as input by the *ems_svc_connect_push_supplier* API to identify the type of messages the event consumer wants to receive. The required syntax of this file consists of zero or more severity filters and zero or more message identifiers. The following syntax applies:

   - One of the following strings: ″ALL_FATAL″, ″ALL_ERROR″, ″ALL_WARNING″, ″ALL_NOTICE″, and ″ALL_NOTICE_VERBOSE″. You can

use one or more of these to indicate that you want to monitor all messages of that severity. Any information following one of these strings on the same line are ignored.

- A message identifier specified as *0xhhhhhhhh*. This is a message ID generated by SAMS. A message ID identifies a SVC message that a DCE application issues. Use this to monitor specific messages.
- If a line does not begin with a severity filter or a message identifier, the line is treated as a comment line and is ignored. Leading white spaces (blanks) are ignored.

  Note: This message identifier must be the OSF message ID, and not the NT DCE message ID.
- The characters are NOT case-sensitive. For example, the file may contain the following:

```
#  This is an example of an event filter file
#  Different severities can be specified
ALL_FATAL
all_error

#  Also, specific message identifiers can be specified
0x12121212
```

Summary

″Interesting″ messages are enabled using both the routing file specifications and the filtering capabilities of the Event Filter file.

Each message is verified to check if it meets either the severity requirement or if its message IDs is one of the message ids in the Event Filter file. If the message passes, it is forwarded. Otherwise, the message is discarded.

It is important to carefully complete the routing file specification, because if the ″interesting″ messages are not routed to the binary log file, regardless of the contents of the Event Filter file, no messages are sent to the consumer.

Other Notes:

- No Event Filter file

  If the Event Filter file does not exist, the *ems_svc_connect_push_supplier* API assumes that the event consumer is at the least interested in ALL_FATAL and ALL_ERROR messages.
- Event Filter file without severities or message identifiers

  If the Event Filter file exists and there are no severities or message identifiers specified, there is no monitoring of messages.
- Changing the Event Filter file

  If the Event Filter file is modified while the event consumer is active, the event consumer must be stopped and restarted for the changes to take effect.
- Maintaining the Binary Log file

  If the binary file is expected to grow over time, it needs to be cleaned periodically. The current cleanup method is to remove the file. This file can be removed only when DCE is stopped.
- Time Stamp file

  To keep track of the events that have already been read from the binary file and forwarded, a file recording the timestamp is stored with a unique user-defined timestamp file. If the event consumer is stopped, messages that have previously

been forwarded are skipped. Deleting this file causes all messages from the beginning of the binary file to be reread and reforwarded.

- Application Server versus. Core Server management

  Event Management Service (EMS), and DCE SNMP Subagent are still used to monitor DCE application servers. EMS provides dynamic filter support and ACL support using persistent databases.

  The direct supplier-consumer model has been provided to allow DCE core servers to be monitored without depending on DCE RPC. This support does not have the dynamic EMS filter and ACL support.

  If a user sets up both EMS and the previously mentioned binary log file monitoring, duplicate messages can be received. When an application server issues a DCE Serviceability message, it is sent to EMS and is logged in the binary log file. EMS forwards the message and the subagent (or another event consumer) that monitors the binary log file reads and forwards the message from the log file.

Chapter 13. IDL Compiler Enhancements

DCE for Windows NT includes an enhanced IDL compiler that provides value-added functions that are not documented in the *OSF DCE Application Development Guide* shipped with this product.

For more information see:

The -standard Build Option

The **-standard** IDL compiler command option allows you to specify portable or extended features of the OSF DCE. This NT-specific option is useful when you perform builds.

The *standard_type* argument specifies the IDL features you want to enable. If you do not specify this argument, the compiler generates warning messages for all features that are not available in the previous version of OSF DCE.

You can specify one of the following values for the *standard_type* argument:

portable
: Allows only the features available in OSF DCE Release 1.0.2.

dce_v10
: Synonymous with the **portable** argument.

dec_v10
: Allows all language features supported by the **-**standard dce_v10 argument, plus a set of extensions to its products based on OSF DCE Release 1.0.

extended
: Allows all language features supported in the current version of the compiler. This is the default.

dce_v11
: Allows only the language features available in OSF DCE Release 1.1. Currently, synonymous with the **extended** argument.

The following example command line compiles the IDL interface **test.idl** and enables extended features of the OSF DCE:

```
C:\> idl test.idl -standard extended
```

The -filename Build Option

The **-filename** IDL compiler command option provides backward compatibility support for stubs named with the short filename format.

The *filename_format* argument specifies the type of filename format you want to use. If you do not specify this argument, the compiler generates long filenames.

You can specify one of the following values for the *filename_format* argument:

short Generates stub files with the following format: *file* _c.c and *file* _s.c.

long Generates stub files with the following format: *file* _cstub.c and *file* _sstub.c.

The following example command line compiles the IDL interface **test.idl** and generates stub files using the short filename format:

```
C:\> idl test.idl -filename short
```

Stub Auxiliary Files

By default, IDL compilers in OSF DCE IDL Release 1.1 and later do not generate the **-caux** and **-saux** files that would have been generated when an IDL file was compiled with earlier releases. However, if you want to use build procedures that were designed to work with earlier compilers, you can cause the Release 1.1 (and later) IDL compiler to generate empty auxiliary files. To do this, define the environment variable **IDL_GEN_AUX_FILES** with the following command:

```
C:\> SET IDL_GEN_AUX_FILES =1
```

You should only need to do this if you are using **-DMIA**.

Garbage Collection Support for Distributed Objects

If you use the Interface Definition Language (IDL) in C++ mode (-lang cxx), garbage collection is supported for distributed objects. For client applications that use a large number of servers or objects, there are two environment variables that can be set to increase the performance of garbage collection by reducing the number of RPC pings to its servers.

If a client application uses more than 20 different servers, you can set the **RPC_RECLAIM_MAX_SERVER** environment variable to a higher value (20 is the default) before starting the application.

For a client application that uses more than 100 distributed objects per server, you can set the **RPC_RECLAIM_MAX_OBJECT** environment variable to a higher value (100 is the default) before starting the application.

For a typical application, the default values provided should be adequate. Increase these values only if necessary since they will increase the use of system resources (that is, memory).

By default, distributed objects are pinged every 5 minutes by a client, and are reclaimed by the server if not pinged for over 1 day. These periods can be tuned by applying the **cxx_reclaim** attribute to the interface. For example:

```
[cxx_reclaim(2,20)]  interface  interface_name
{
...
}
```

This sets the ping period to 20 minutes, and only reclaims an object after 2 days of activity. It is anticipated that the default, implicit attribute of **cxx_reclaim(1,5)** is reasonable in most cases.

Garbage collection can also be suppressed by applying an attribute of **[cxx_reclaim(0,0)]** to the interface.

Chapter 14. Modifications to Internationalization

The Internationalization function in DCE for Windows NT differs from the OSF DCE function in that DCE for Windows NT extends the character set that is allowed in PGO names.

The RPC code set conversion feature supports a revised version of the architecture. See “RPC Code Set Conversion” on page 257 for additional information.

OSF DCE uses the XPG4 programming model to implement the messaging and serviceability API’s. The Windows NT operating system does not support this model, but DCE for Windows NT provides the underlying XPG4 support for the messaging and serviceability API’s.

For more information see:

Naming Considerations

Serviceability Component Names

Installing Application Message Catalogs

Synchronization of XPG4 Internationalization with the Country Environment

dce_setlocale

Naming Considerations

According to standard (OSF) DCE, entries in the Security namespace, such as principal names, can consist of only characters in the DCE portable character set. DCE for Windows NT provides an override capability which enables the use of non-portable characters. This override capability is provided primarily for migration to DCE Security and should be used only in environments that are homogeneous with respect to both platforms and code set.

Security namespace entries that use non-portable characters are guaranteed to work correctly only with DCE for Windows NT and only when the code set of the entire enterprise is the same as that of the process under which the names are created. To enable non-portable Security names, set this environment variable:

```
DCE_USE_NONPORTABLE_NAMES=1
```

Using standard (OSF) DCE, certain entries in the cell directory services (CDS) namespace, such as directory names, can be composed of characters from outside of the DCE portable character set. Because DCE for Windows NT does not perform code set conversions on CDS names, you should use non-portable characters only in environments that are, and will remain homogeneous with respect to the code set.

Subject to the restrictions above and to the additional naming rules documented elsewhere, the following names can contain characters outside of the Portable Character Set:

```
CDS Object
CDS Directory
CDS Attribute
CDS Link

RPC idl_byte data
```

```
RPC full name

Principal
Group
Organization
ERA
```

Note: Windows NT is extremely heterogeneous with respect to code sets. Unless, you have a very clear understanding of the variety of code sets conventions that exist simultaneously in a Windows NT environment, you should restrict system data such as principal group organization (PGO) names to the DCE portable character set.

Messaging and Serviceability Considerations

Select from the following topics for information on component names and message catalogs, and instructions on how to install the message catalogs.

Serviceability Component Names

Installing Application Message Catalogs

Serviceability Component Names

The OSF Application Development Guide discusses how to create sams files and use the sams command to generate XPG4-style message catalogs for your DCE application. The example program uses the component name **hel**. This component name is used in the names of the various files that are generated by the sams command, and is also used to generate globally-unique message IDs for the component.

When your program calls a DCE Messaging or Serviceability function using a message ID number, DCE derives the component name from the number and looks for the message in a message catalog named **dcexxx.cat**, where xxx is the component name. For the example program, DCE will look for a message catalog named **dcehel.cat**. If DCE does not find the message in a catalog, it will attempt to find it in a default message table, using the message ID as the index.

The association of message numbers with message catalog names allows DCE to locate the appropriate message catalog using only the message ID. Likewise, globally-unique message numbers enable DCE to efficiently search internal message tables. This scheme reduces the complexity of messaging for DCE and for the DCE application programmer, but in order for it to work, the serviceability component names must be unique.

When you choose serviceability component names for your applications, you must first make sure that you don't use a name that is already being used by DCE itself. You can see the complete list of DCE components by looking at the names of the message catalogs in the directory %DCELOC%\dcelocal\nls\msg\enus1252.

In addition, if you want to ensure that the names you use will not subsequently be used by DCE or by another registered DCE application you can register your serviceability component names by sending e-mail to *dce-registry@osf.org*.

Installing Application Message Catalogs

After using sams files to create a new message catalog you must copy the catalog to a location in the NLSPATH. You can put your application message catalogs with the DCE system catalogs. If you put them somewhere else, you must add the path to the NLSPATH environment variable.

Windows NT Code Page Considerations

There are two types of code pages, OEM and ANSI. Whether data is entered from a command line or DCE prompt determines which code page it is encoded.

OEM Versus ANSI Code Pages

Code Page Conversion or Command Line Parameters

OEM Versus ANSI Code Pages

Windows NT defines two types of code pages. The OEM code pages are the traditional MS-DOS/IBM-PC code pages, such as code page 437 and 850. The ANSI code pages, such as code page 1252, are more similar to the ISO standard code sets. For Asian code pages, such as code page 932, there is no difference between the OEM and ANSI code pages.

Windows NT generally uses the OEM code pages in console sessions and the ANSI code pages in Windows.

Code Page Conversion or Command Line Parameters

When you start a program from the command line in a console session, Windows NT automatically converts any command line parameters from the OEM code page encoding to an ANSI code page encoding.

If you are using non-English characters with DCE commands such as **dce_login** and **dcecp**, you must be aware that data which is passed to DCE will be encoded in ANSI code pages if it is entered on the command line. It will be encoded in OEM code pages if it is entered on DCE prompts.

For example, if you are using the extended character set for PGO names, and you create a registry which contains ANSI-encoded principal names, you should use **dce_login** with the principal name on the command line. If you create a registry with OEM-encoded names, you should allow **dce_login** to prompt you for the principal name.

These considerations do not apply if you are using Asian data. The OEM and ANSI code pages are equivalent.

XPG4 Internationalization

DCE uses the XPG4 internationalization model, which is based on locales. A particular locale specifies a human language, a set of cultural conventions associated with various types of numeric and character data, and the code page in which character data is encoded.

You can write locale names in any of the following ways:

xx_yy.ZZZZ
xxyyZZZZ
xx_yy
xxyy

where xx is a language abbreviation, yy is a country abbreviation, and ZZZZ is an optional 3- or 4-digit code page number. Case is ignored. The code page number is optional. If it is left off the number of the currently active code page will be appended to the locale name.

For more information, see:

Setting the Locale Using LANG

Locale and Message Catalogs, NLSPATH Environment Variable

Supported Locale Names

Synchronization with the Country Environment

Setting the Locale Using LANG

You can use the environment variable LANG to set the locale for a process (or globally via the Registry). Each DCE locale is implemented as a DLL, and the DLL names are the valid locale names. The DCE locales are installed in directory %DCELOC%\dcelocal\locale\. For the meaning of each name, see Supported Locale Names.

For example, since DCE has a locale DLL named enus437.dll, you can set the locale for English (language), US (cultural conventions), and 437 (code page), by typing:

```
set LANG=enus437
```

Because a code page is optional, you could also type:

```
set LANG=en_us
```

and ″**.437** would be appended to the locale name on a typical US installation of Windows NT.

Locale and Message Catalogs, NLSPATH Environment Variable

The locale names are used to locate the message catalogs. DCE's message catalogs are installed in directories of the form %DCELOC%\dcelocal\nls\msg\xxyyZZZZ\, where xxyyZZZZ is a locale name. For example, the message catalogs for Japanese are located in the directory %DCELOC%\dcelocal\nls\msg\jajp932\.

The environment variable NLSPATH specifies the path (or a series of paths, separated by ″;″) which is searched when DCE looks for a message catalog. You can add additional paths to the NLSPATH, but you should preserve the path that is set by DCE at install time:

```
NLSPATH=%DCELOC%\dcelocal\nls\msg\%L\%N.
```

This path uses two standard substitution variables, %L and %N. Both must be uppercase. The locale name is substituted for %L and a message catalog name is substituted for %N.

For example, if your application provides German message catalogs, you can put them in the directory %DCELOC%\dcelocal\nls\msg\dede850. The DCE Messaging and Serviceability APIs will search this path if LANG is set to **dede850** or if **dede850** is passed to **dce_setlocale()**.

Note: There is a slight potential for conflict with non-DCE applications and other XPG4 implementations, if they manipulate or interpret the contents of the NLSPATH variable in a different way. For example, DCE for Windows NT uses the semicolon instead of the colon for the NLSPATH separator.

Supported Locale Names

The following are the locale names that DCE supports.

```
″ARAA1256″,     /* Arabic in Arabic area ANSI CP                */
″ARAA864″,       /* Arabic in Arabic area OEM CP                  */
″BGBG1251″,     /* Bulgarian in Bulgaria  ANSI CP               */
″CAES1252″,      /* Catalan in Catalia ANSI CP                     */
″CSCZ1250″,      /* Czech in Czech Republic  ANSI CP            */
″CSCZ852″,        /* Czech in Czech Republic     OEM CP         */
″DADK1252″,     /* Danish in Denmark ANSI CP                    */
″DADK1252″,     /* Danish in Denmark OEM CP                      */
″DECH1252″,     /* German in Switzerland  ANSI CP              */
″DECH850″        /* German in Switzerland  OEM CP                */
″DEDE1252″,     /* German in Germany ANSI CP                     */
″DEDE850″,       /* German in Germany OEM CP                     */
″ELGR1253″,     /* Greek in Greece ANSI CP                        */
″ELGR869″,       /* Greek in Greece OEM CP                          */
″ENGB1252″,     /* English in U.K. ANSI CP                       */
″ENGB850″,      /* English in U.K. OEM CP                         */
″ENUS1252″,     /* English in US  ANSI CP                          */
″ENUS437″,      /* English in US OEM CP                            */
″ENUS850″,      /* English in US OEM CP                             */
″ESES1252″,     /* Spanish in Spain ANSI CP          */
″ESES850″,       /* Spanish in Spain OEM CP           */
″FIFI1252″,      /* Finnish in Finland ANSI CP                    */
″FIFI850″,       /* Finnish in Finland OEM CP                     */
″FRBE1252″,     /* French in Belgium ANSI CP                     */
″FRBE850″,       /* French in Belgium OEM CP                      */
″FRCA1252″,     /* French in Canada ANSI CP                      */
″FRCA850″,       /* French in Canada OEM CP                       */
″FRCH1252″,     /* French in Switzerland ANSI CP                */
″FRCH850″,       /* French in Switzerland OEM CP                  */
″FRFR1252″,      /* French in France ANSI CP                      */
″FRFR850″,       /* French in France OEM CP                       */
″HRHR1250″,     /* Croatian in Croatia ANSI CP                   */
″HRHR852″,       /* Croatian in Croatia OEM CP                    */
″HUHU1250″,     /* Hungarian in Hungary ANSI CP                 */
″HUHU852″,       /* Hungarian in Hungary OEM CP                  */
″ISIS1252″,       /* Icelandic in Iceland ANSI CP                 */
″ISIS850″,        /* Icelandic in Iceland OEM CP                   */
″ITIT1252″,       /* Italian in Italy ANSI CP                      */
″ITIT850″,        /* Italian in Italy OEM CP                        */
″IWIL1255″,      /* Hebrew in Israel ANSI CP                      */
″IWIL862″,        /* Hebrew in Israel OEM CP                       */
″JAJP932″,         /* Japanese in Japan                                  */
```

```
"KOKR949",          /* Korean in Korea                                    */
"MKMK1251",        /* Macedonian in Macedonia                            */
"NLBE1252",         /* Flemish in Belgium ANSI CP                        */
"NLBE850",          /* Flemish in Belgium OEM CP                         */
"NLNL1252",         /* Dutch in Netherlands ANSI CP                      */
"NLNL850",          /* Dutch in Netherlands OEM CP                       */
"NONO1252",         /* Norwegian in Norway ANSI CP                       */
"NONO850",          /* Norwegian in Norway OEM CP                        */
"PLPL1250",         /* Polish in Poland ANSI CP                          */
"PLPL852",          /* Polish in Poland OEM CP                           */
"PTBR1252",         /* Portuguese in Brazil ANSI CP                      */
"PTBR850",          /* Portuguese in Brazil OEM CP                       */
"PTPT1252",         /* Portuguese in Portugal ANSI CP                    */
"PTPT850",          /* Portuguese in Portugal OEM CP                     */
"RORO1250",         /* Romanian in Romania ANSI CP                       */
"RORO852",          /* Romanian in Romania OEM CP                        */
"RURU1251",         /* Russian in Russia ANSI CP                         */
"RURU866",          /* Russian in Russia OEM CP                          */
"SKSK1250",         /* Slovak in Slovakia ANSI CP                        */
"SKSK852",          /* Slovak in Slovakia OEM CP                         */
"SLSI1250",         /* Slovenian in Slovenia ANSI CP                     */
"SLSI852",          /* Slovenian in Slovenia OEM CP                      */
"SQAL1252",         /* Albanian in Albania                               */
"SHSP1250",         /* Latin Serbian in Serbia ANSI CP                   */
"SRSP1251",         /* Cyrillic Serbian in Serbia ANSI CP                */
"SVSE1252",         /* Swedish in Sweden ANSI CP                         */
"SVSE850",          /* Swedish in Sweden OEM CP                          */
"THTH874",          /* Thai in Thailand                                  */
"TRTR1254",         /* Turkish in Turkey ANSI CP                         */
"TRTR857",          /* Turkish in Turkey OEM CP                          */
"ZHCN936",          /* Simplified Chinese in China                       */
"ZHTW950",          /* Traditional Chinese in Taiwan                     */
```

Synchronization with the Country Environment

DCE uses the X/Open XPG4 locale programming model for internationalization. Internationalized Windows NT applications may use this model, or they may use the traditional Windows NT country environment model. Although DCE is designed to use the locale model, it accommodates the Windows NT internationalization programming and operating environments. You do not have to change existing internationalized Windows NT applications when they are running on DCE.

DCE internationalization works properly even when the DCE application program does not establish a usable XPG4 internationalization environment, either because the program is not using XPG4 or because it is using an incompatible version of XPG4 internationalization routines. Although the XPG4 programming model requires the application to set up the XPG4 environment, DCE internationalization works correctly without the XPG4 programming requirement.

DCE NT applications can use either the XPG4 locale programming model or the NT proprietary internationalization programming models.DCE applications can be divided into four classes with respect to internationalization:

- **Windows NT country environment programming model; single-locale**

Implements internationalization using the Windows NT operating system interfaces, such as **GetLocaleInfo**, or other proprietary Windows NT techniques. This model does not support multiple languages or code sets during a single invocation of the application.

- **Windows NT internationalization programming model; multiple-locale**

  Implements internationalization using the Windows NT operating system interfaces, such as SetLocale, SetThreadLocale, GetLocaleInfo, and other proprietary Windows NT techniques. It supports multiple languages and code sets during a single invocation of the application.

- **XPG4 internationalization programming model; single-locale**

  Implements internationalization on some variant of XPG4, from a third party software package or from a compiler such as IBM VisualAge C++ NT version, which offers XPG4 support. It does not support multiple locales during a single invocation of the application.

- **XPG4 internationalization programming model; multiple-locale**

  Implements internationalization on some variant of XPG4, from a third party software package or from a compiler such as IBM VisualAge C++ NT version, which offers XPG4 support. It supports multiple locales during a single invocation of the application.

The DCE Messaging and Serviceability functions and the RPC code set conversion functions are designed to use the XPG4 programming model. However, it is not necessary for the NT DCE application to use this model, because NT DCE automatically synchronizes DCE's XPG4 locale with the NT country environment or locale of the application program.

This synchronization takes place when the application program starts to run. If the program changes its code page, or if it explicitly sets the XPG locale, the **dce_setlocale()** function must be called to force the re-synchronization of DCE's internationalization environment with that of the application. Calling the **dce_setlocale()** function has the effect of switching the DCE locale to the one requested by the program.

NT DCE Default Locale

If the DCE application does not explicitly set the locale by calling **dce_setlocale(),** DCE uses XPG4 environment variables or NT country information to establish the locale. If the LANG environment variable is set, the DCE locale is based on its value and on the code page of the process. If LANG is not set, the DCE locale is based on the Windows NT country code and the code page of the process. If the derived locale is not one of those supported by DCE, DCE uses the closest locale which preserves the code page setting.

Note the following:

- Changing to a locale for which DCE does not ship message catalogs changes DCE messages to English.
- For existing multiple-locale applications and new ones that do not use **dce_setlocale(),** the XPG4 synchronization occurs only when the program is started. Any changes to NT country environment or to XPG4 locale will not be seen by DCE unless the **dce_setlocale()** function is called when the locale is changed in the application program.

- Because the underlying implementation of an XPG4 application's locale may be different from that of DCE, the XPG4 application must both call **setlocale()** and **dce_setlocale()** when explicitly setting the locale.

Special Notes Regarding Internationalization

- The control programs which have been superseded by **dcecp** in this DCE release (**rgy_edit**, **acl_edit**, **cdscp**, **rpccp**, **sec_admin**, and **dtscp**) have not been enabled to support multibyte characters. If you are working in a multibyte environment, you should use dcecp.
- The DCE-supported locales are located in **%DCELOC%\dcelocal\locale** . These DLL names, such as **enus437**, can be used as the second parameter of the **dce_setlocale()** function, or as the value of the LANG environment variable.

dce_setlocale

This DCE serviceability routine explicitly sets the DCE locale.

Format

#include <dce\dce_msg. h>

char *dce_setlocale(DCE_LC_ALL, const char **locale);*

Parameters

Input

DCE_LC_ALL Set all locale categories.

locale Locale name expressed as a quoted string.

Usage

The **dce_setlocale** routine**:**

Is a wrapper function. The **dce_setlocale** routine calls the runtime version of the **setlocale** routine which is used by the DCE code.

- Supports a subset of the standard **setlocale** routine.
- Supports the setting of all locale categories to a single locale (but does not support the setting of individual locale categories).
- Returns a pointer to a string that lists the values of the locale categories. These values match the *locale* parameter of the call.

Return Values

On error **dce_setlocale** returns NULL, and the locale is not changed.

Note: DCE programs that do not use the locale model do not need to call **dce_setlocale** because DCE will initialize a locale that matches the country and code page environment. DCE programs that call **setlocale,** however, should also call **dce_setlocale** to ensure that DCE's locale matches the locale of the program.

Because **dce_setlocale** is not supported on all implementations of DCE, make sure that any calls to **dce_setlocale** are within the bounds of an #ifdef for portability.

The following code fragment shows how to use the **dce_setlocale routine**.

```
#include <dce_msg.h>
char *string;
void main(void)
{
        /*Set the DCE locale to jajp932.  Note that unless the
      **            DCE Japanese message files are present, DCE messages
      **            will be displayed or logged in English.
       */
      #ifdef WIN32
               string = dce_setlocale(DCE_LC_ALL, "jajp932") ;
               if (string != NULL)
               {
                if (string != NULL)
                }
       #endif
}
```

The output is:

JAJP932 JAJP932 JAJP932 JAJP932 JAJP932 JAJP932

RPC Code Set Conversion

The support in DCE for NT 2.2 varies in some key ways from the original OSF implementation. The codesets.idl interface has been modified to provide improved cross-platform support. The version number has been increased to 2.0, which is the version supported in the other IBM implementations of DCE. The following differences are discussed, along with important programming information and a revised version of the sample RPC program discussed in the *Application Development Guide*.

In this release, the code sets attribute should be accessed only through the RPC NSI interface to the Directory Services. Although the code sets attribute has an ISO Object Identifier (OID), it should not be referenced in the current release.

The primary interface change to the code set data structure is discussed in the rpc_intro section of the *Application Development Reference* ″RPC Data Types and Structures″.

The revised code set data structure contains the following fields:

c_set A 32-bit hexadecimal value assigned by OSF that uniquely identifies the code set.

c_max_bytes
A 16-bit decimal value that indicates the maximum number of bytes this code set uses to encode one character.

ch_sets_num
A 16-bit decimal value that indicates the number of character sets supported by the code set.

ch_sets
A 32-bit pointer to a dynamically allocated array of OSF-assigned character set identifiers.

The following routines require a code set value:

- cs_byte_from_netcs
- cs_byte_local_size
- cs_byte_net_size
- cs_byte_to_netcs
- dce_cs_loc_to_rgy
- dce_cs_rgy_to_loc
- rpc_cs_get_tags
- rpc_cs_binding_set_tags
- rpc_rgy_get_max_bytes

In these routines, the code set value has a data type of unsigned32.

Note: The wchar_t* routines are not supported in this release of DCE.

The RPC stub buffer sizing routines **xxx_net_size** and **xxx_local_size** use the value of c_max_bytes to calculate the size of a buffer for code set conversion.

The RPC character set compatibility evaluation routine **rpc_cs_char_set_compat_check** uses the value of **ch_sets_num** and values pointed to by **ch_sets** to evaluate character set compatibility between a client and a server.

The revised C language representation of the code set data structure is as follows:

```
typedef struct {
            long   c_set;
            short  c_max_bytes;
            short  ch_sets_num;
            short  *ch_sets;
    } rpc_cs_c_set_t;
```

The code set data structure is a member of the code sets array, which is discussed in ″Data Types and Structures″.

Additional Implementation Details and Restrictions

The following provide implementation details and restrictions for the Code Sets array, the **cs_char** attribute and a sample program.

Code Sets Array

DCE for NT 2.2 does not support use of the **csrc** command to modify the code sets array that is shipped with DCE. The use of an intermediate code set other than ISO 10646 (UCS-2, Level 1) is not supported.

The DCE-supplied stub buffer sizing routines do not support the **idl_cs_in_place_convert** conversion type. The conversion method is determined at runtime and there is no guarantee that the conversion can be performed in a single storage area.

You can find the DCE for NT 2.2 code sets array database (text and binary) at **%DCELOC%\dcelocal\nls\csr**. The text file was used to generate the binary version, which is used by RPC at runtime. The text file contains OSF-assigned values and values unique to the DCE for NT which may be required by the application programmer.

There are individual entries for each registered code set. Each entry has this format:

```
description  text
loc_name     text
rgy_value    unsigned32
char_values  unsigned16:...
max_bytes    unsigned16
```

loc_name
The local code set name. It must be enclosed in quotes when used as a parameter in an RPC API.

rgy_value
The OSF-registered unique code set identifier.

char_values
The value of the OSF-registered character set IDs for the code page.

max_byte
The maximum number of bytes for a character in the code page.

cs_char Attribute

Arrays of **cs_char** can be fixed, varying, conformant, or conformant varying. The treatment of a scalar **cs_char** is similar to that of a fixed array of one element.

In this release, only conformant or conformant varying arrays can be used without restrictions, because they are designed to allow the data expansion and contraction which can occur during code set conversion.

For fixed or varying arrays, the array size is fixed and can not be modified during the RPC marshalling or unmarshalling. For fixed arrays, the number of bytes of data on the client, the server, and the network must be exactly equal to the number defined in the IDL file.

The following are the additional restrictions for fixed or varying arrays.

Fixed Arrays:

- The number of array elements in the local (client and server) and network representations of the data must be the same as the array size defined in the IDL.
- Because the array size is the input length used by the code set conversion, the complete array must be populated with valid data.
- You must write your own stub buffer sizing routines and code set conversion routines. The routines provided by DCE RPC do not support the **idl_cs_in_place_convert** conversion type.
- You may write your own stub tag-setting routines or use the DCE RPC tag-setting routine **rpc_cs_get_tags()** to set the sending tag value and the receiving tag value. You must ensure that the code set conversion between server and client will not result in data expansion or contraction.
- You may write your own character and code sets compatibility evaluation routines. You must not use the DCE RPC **rpc_cs_eval_with_universal()** because universal conversion may cause data expansion. You may use the **rpc_cs_eval_without_universal()** but keep in mind that the conversion model selection used by this routine is: use RMIR if possible, else use SMIR if possible, then CMIR. You must make sure that the conversion can be performed without data expansion or contraction.

Varying Arrays:

- Neither the number of array elements in the local representation nor the number of array elements in the network representation may exceed the array size in the IDL.
- The value of **length_is** is the input length used by the code set conversion routine.
- Expansion and contraction of data is allowed within the array size defined in the IDL file.

Revised Sample Program

```
/*
This is the sample server code
*/

 #include <stdio.h>
 #include <stdlib.h>
 #include <dce/rpc.h>
 #include <dce/dce.h>
 #include <dce/nsattrid.h>
 #include <dce/dce_error.h>
 #include <locale.h>
 #include <pthread.h>
 #include <dce/codesets.h>
 #include <dce/dce_msg.h>
 #include "sample.h"      // IDL generated header

 /*
  * Macro for result checking
  */

 #define msg1 "FAILED %s()\nresult: expected:   %s\n\n"
 #define msg3 "Listening for remote procedure calls...\n"
 #define CHECK_STATUS(t, func, returned_st, expected_st) \
 { \
      if (returned_st == expected_st) { \
      } else { \
           dce_error_inq_text(returned_st,\
            (unsigned char *)unexpected, &dce_status); \
           dce_error_inq_text(expected_st, \
           (unsigned char *)expected, &dce_status); \
           printf("FAILED %s()\nresult: %s\nexpected:  %s \n\n", \
                     func, unexpected, expected); \
      } \
 } \

 static unsigned char     unexpected[dce_c_error_string_len];
 static unsigned char     expected[dce_c_error_string_len];
 static int          dce_status;

 int
 main(int argc, char *argv[])
 {
      error_status_t          status;
      int               i;
      rpc_ns_handle_t          inq_contxt;
      rpc_binding_vector_t     *binding_vector;
      rpc_codeset_mgmt_p_t      arr;
      pthread_t           this_thread = pthread_self();
      char                *nsi_entry_name;
      char                *server_locale_name;
      error_status_t           expected = rpc_s_ok;
      int               server_pid;

      /* The environment variable I18N_SERVER_ENTRY needs
```

```
 * to be set before running this program. This is
 * not a DCE environment variable, so you can set up
 * your own environment variable if you like.
 */

nsi_entry_name = getenv("I18N_SERVER_ENTRY");


/* If the XPG/POSIX programming model is being used, set
 * the locale. In this way, the current locale
 * information is extracted from XPG/POSIX defined
 * environment variable LANG or LC_ALL.
 * Call dce_setlocale() to synchronize the program locale
 * with DCE's locale.
 */

setlocale(LC_ALL, "");
dce_setlocale(DCE_LC_ALL, "");

/*
 * Get supported code sets.
 */
rpc_rgy_get_codesets (
     &arr,
     &status );

CHECK_STATUS(TRUE, "rpc_rgy_get_codesets", status, expected);

rpc_server_register_if (
     cs_test_v1_0_s_ifspec,
     NULL,
     NULL,
     &status );

CHECK_STATUS(TRUE, "rpc_server_register_if", status, expected);

rpc_server_use_all_protseqs (
     rpc_c_protseq_max_reqs_default,
     &status );

CHECK_STATUS(TRUE, "rpc_server_use_all_protseqs", status, expected);

rpc_server_inq_bindings (
     &binding_vector,
     &status );

CHECK_STATUS(TRUE, "rpc_server_inq_bindings", status, expected);

rpc_ns_binding_export (
     rpc_c_ns_syntax_default,
     (unsigned_char_p_t)nsi_entry_name,
     cs_test_v1_0_s_ifspec,
     binding_vector,
     NULL,
     &status );

CHECK_STATUS(TRUE, "rpc_ns_binding_export", status, expected);

rpc_ep_register (
     cs_test_v1_0_s_ifspec,
     binding_vector,
     NULL,
     NULL,
     &status );

CHECK_STATUS(TRUE, "rpc_ep_register", status, expected);
```

```
/*
 * Register the server's supported code sets into the name space.
 */
rpc_ns_mgmt_set_attribute (
     rpc_c_ns_syntax_default,
     (unsigned_char_p_t)nsi_entry_name,
     rpc_c_attr_codesets,
     (void *)arr,
     &status );

CHECK_STATUS(TRUE, "rpc_ns_mgmt_set_attribute", status, expected);

/*
 * Free memory allocated by getting code sets.
 */
rpc_ns_mgmt_free_codesets (&arr, &status);

CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codeset", status, expected);

TRY
{

     printf(msg3);

     rpc_server_listen (
          rpc_c_listen_max_calls_default,
          &status );

     CHECK_STATUS(TRUE, "rpc_server_listen", status, expected);

     /*
      * Remove code set attributes from namespace on return.
      */

     rpc_ns_mgmt_remove_attribute (
          rpc_c_ns_syntax_default,
          (unsigned_char_p_t)nsi_entry_name,
          rpc_c_attr_codesets,
          &status );

     CHECK_STATUS(TRUE, "rpc_ns_mgmt_remove_attribute", status, expected);

     rpc_ns_binding_unexport (
          rpc_c_ns_syntax_default,
          (unsigned_char_p_t)nsi_entry_name,
          cs_test_v1_0_s_ifspec,
          (uuid_vector_p_t)NULL,
          &status );

     CHECK_STATUS(TRUE, "rpc_ns_binding_unexport", status, expected);

     rpc_ep_unregister (
          cs_test_v1_0_s_ifspec,
          binding_vector,
          NULL,
          &status );

     CHECK_STATUS(TRUE, "rpc_ep_unregister", status, expected);


     rpc_binding_vector_free (
          &binding_vector,
          &status );

     CHECK_STATUS(TRUE, "rpc_binding_vector_free", status, expected);
```

```
            rpc_server_unregister_if (
                 cs_test_v1_0_s_ifspec,
                 NULL,
                 &status );

            CHECK_STATUS(TRUE, "rpc_server_unregister_if", status, expected);

       }
       CATCH_ALL
       {
            /*
             * Remove code set attribute from namespace on a signal.
             */

            rpc_ns_mgmt_remove_attribute (
                 rpc_c_ns_syntax_default,
                 (unsigned_char_p_t)nsi_entry_name,
                 rpc_c_attr_codesets,
                 &status );

            CHECK_STATUS(TRUE, "rpc_ns_mgmt_remove_attribute", status, expected);

            rpc_ns_binding_unexport (
                 rpc_c_ns_syntax_default,
                 (unsigned_char_p_t)nsi_entry_name,
                 cs_test_v1_0_s_ifspec,
                 (uuid_vector_p_t)NULL,
                 &status );

            CHECK_STATUS(TRUE, "rpc_ns_binding_unexport", status, expected);

            rpc_ep_unregister (
                 cs_test_v1_0_s_ifspec,
                 binding_vector,
                 NULL,
                 &status );
            CHECK_STATUS(TRUE, "rpc_ep_unregister", status, expected);


            rpc_binding_vector_free (
                 &binding_vector,
                 &status );

            CHECK_STATUS(TRUE, "rpc_binding_vector_free", status, expected);


            rpc_server_unregister_if (
                 cs_test_v1_0_s_ifspec,
                 NULL,
                 &status );

            CHECK_STATUS(TRUE, "rpc_server_unregister_if", status, expected);

       }
       ENDTRY;
 }


error_status_t cs_conf_trans (
        handle_t        h,
        idl_long_int    in_size,
        net_byte        *in_string,
        net_byte        *out_string
)
{
        int i, length, k, s_size;
        wchar_t *wcs;
```

```
        wchar_t *wcs_rev;
        int   w_size;

        s_size = (int)in_size;

        w_size = sizeof(wchar_t);
        wcs     = (wchar_t *)malloc((s_size + 1) * MB_CUR_MAX * w_size);
        wcs_rev = (wchar_t *)malloc((s_size + 1) * MB_CUR_MAX * w_size);


        length = (int)mbstowcs(wcs, (char *)in_string, s_size);
        if (length == -1) {
                fprintf(stderr, "Error occurred in mbstowcs!!\n");
        }

        k = length - 1;
        for (i = 0; i<length; i++)
                wcs_rev[i] = wcs[k--];
        wcs_rev[i] = L'\0';

        k = (int)wcstombs((char *)out_string, wcs_rev, s_size);
        if (k == -1) {
                fprintf(stderr, "Error occurred in wcstombs!!\n");
                return(0);
        }

        free(wcs);
        free(wcs_rev);

        return(rpc_s_ok);
}

/*
This is the sample client code
*/

 #include <stdio.h>
 #include <string.h>
 #include <locale.h>
 #include <dce/rpc.h>
 #include <dce/rpcsts.h>
 #include <dce/dce_error.h>
 #include <dce/dce_msg.h>

 #include "sample.h"    // IDL generated header

 /*
  * Result check MACRO
  */
 #define msg1 "FAILED %s()\nresult: %s\nexpected:  %s \n\n"
 #define msg2 "is_server_listening error -> %s\n"
 #define msg3 "i18n_input_data open failed\n"
 #define msg4 "i18n_result_file open failed\n"
 #define msg5 "FAILED %ld  MSG: %s\n"
 #define msg6 "PASSED rpc #%d\n"


 #define CHECK_STATUS(t, func, returned_st, expected_st) \
 { \
      if (returned_st == expected_st) { \
      } else { \
           dce_error_inq_text(returned_st,\
            (unsigned char *)unexpected, &dce_status); \
           dce_error_inq_text(expected_st, \
           (unsigned char *)expected, &dce_status); \
           printf("FAILED %s()\nresult: %s\nexpected:  %s \n\n", \
                     func, unexpected, expected); \
```

```
    } \
} \

static unsigned char     unexpected[dce_c_error_string_len];
static unsigned char     expected[dce_c_error_string_len];
static int          dce_status;

void
main(void)
{
     rpc_binding_handle_t    bind_handle;
     rpc_ns_handle_t         import_context;
     error_status_t          status;
     error_status_t          temp_status;
     cs_byte                net_string[SIZE];
     cs_byte                loc_string[SIZE];
     unsigned char           err_buf[256];
     char               *nsi_entry_name;
     char               *client_locale_name;
     int                i, rpc_num;
     FILE               *fp_in, *fp_out;
     long               in_str_len; /* in_str_len is the actual size in bytes of the input stri


     /* The environment variable I18N_SERVER_ENTRY needs
      * to be set before running this program. This is
      * not a DCE environment variable, so you can set up
      * your own environment variable if you like.
      */

     nsi_entry_name = getenv("I18N_SERVER_ENTRY");


     /* If the XPG/POSIX programming model is being used, set
      * the locale. In this way, the current locale
      * information is extracted from XPG/POSIX defined
      * environment variable LANG or LC_ALL.
      * Call dce_setlocale() to synchronize the program locale
      * with DCE's locale.
      */

     setlocale(LC_ALL, "");
     dce_setlocale(DCE_LC_ALL, "");

     rpc_ns_binding_import_begin (
               rpc_c_ns_syntax_default,
          (unsigned_char_p_t)nsi_entry_name,
          cs_test_v1_0_c_ifspec,
               NULL,
               &import_context,
               &status );

     CHECK_STATUS(TRUE, "rpc_ns_binding_import_begin", status, rpc_s_ok);

     /*
      *  Add code set compatibility checking logic to the context.
      */
     rpc_ns_import_ctx_add_eval (
               &import_context,
          rpc_c_eval_type_codesets,
          (void *)nsi_entry_name,
               rpc_cs_eval_with_universal,
          NULL,
               &status );

     CHECK_STATUS(TRUE, "rpc_ns_import_ctx_add_eval", status, rpc_s_ok);
```

```
while (1) {
        rpc_ns_binding_import_next (
                    import_context,
                    &bind_handle,
                    &status );

     CHECK_STATUS(TRUE, "rpc_ns_binding_import_next", status, rpc_s_ok);

     if (status == rpc_s_ok)
          break;
           else
           {
                   return;
           }
}

   rpc_ns_binding_import_done (
           &import_context,
           &status );

CHECK_STATUS(TRUE, "rpc_ns_binding_import_done", status, rpc_s_ok);

   rpc_ep_resolve_binding (bind_handle,
           cs_test_v1_0_c_ifspec,
           &temp_status);

CHECK_STATUS(TRUE, "rpc_ep_resolve_binding", temp_status, rpc_s_ok);

   if(rpc_mgmt_is_server_listening(bind_handle, &status)
           && temp_status == rpc_s_ok)
   {
           ; /* Do nothing. */
   }
   else
   {
           dce_error_inq_text ((unsigned long)status,
      err_buf, (int *)&temp_status);
           printf("is_server_listening error ->  %s \n", err_buf);
   }

/*
 *  This program reads the data from a file.
 */

fp_in = fopen("./i18n_input_data", "r");

if (fp_in == NULL)
{
     printf(msg3);
     return;
}

fp_out = fopen("./i18n_method_conf_result_file", "w");

if (fp_out == NULL)
{
     printf(msg4);
     fclose(fp_in);
     return;
}

(void)fgets((char *)net_string, SIZE, fp_in);
while (!feof(fp_in))
{


     in_str_len = (long)strlen(net_string) + 1;
```

```
          temp_status = cs_conf_trans(bind_handle, in_str_len, net_string, loc_string);

          if (temp_status != rpc_s_ok)
          {
               dce_error_inq_text(temp_status, err_buf,
                (int *)&status);

                printf(msg5);
                ((unsigned long)temp_status, err_buf);
          }
          else
          {
               printf(msg6, rpc_num++);
               (void)fputs((char *)loc_string, fp_out);
               (void)fputs("\n", fp_out);
          }
        (void)fgets((char *)net_string, SIZE, fp_in);
      }

      fclose(fp_in);
      fclose(fp_out);

      return;
 }


/*
This is the IDL file for sample code
*/

[
uuid(b076a320-4d8f-11cd-b453-08000925d3fe),
version(1.0)
]

interface cs_test
  {
       const unsigned short SIZE = 100;
       typedef byte net_byte;

          error_status_t cs_conf_trans (
                  [in] handle_t     IDL_handle,
            [in] unsigned     long stag,
            [in] unsigned     long drtag,
            [out] unsigned     long *p_rtag,
            [in] long          arr_size,
            [in, size_is(arr_size)]       net_byte  a[*],
            [out, size_is(arr_size)]      net_byte  b[*]
          );
}

/*
This is the ACF file for sample program
*/

  [
  explicit_handle
  ]
  interface cs_test
  {
       include "dce/codesets_stub";

       typedef [cs_char(cs_byte)] net_byte;

       [comm_status, cs_tag_rtn(rpc_cs_get_tags)] cs_conf_trans (
            [cs_stag] stag,
```

```
            [cs_drtag] drtag,
            [cs_rtag] p_rtag );
}
```

Here is an example client program of the cs_test interface that provides its own character set compatibility evaluation and conversion model selection. This example client uses the rpc_cs_binding_set_tags() routine to set the code set tags within the client code rather than using a tag-setting routine to set them within the stub code.

```
#include <stdio.h>
#include <locale.h>
#include <dce/rpc.h>
#include <dce/rpcsts.h>
#include <dce/dce_error.h>

#include "cs_test.h"          /* IDL generated include file */

/*
 * Result check MACRO
 */
#define CHECK_STATUS(t, func, returned_st, expected_st) \
{
        if (returned_st == expected_st) { \
     ;  /* No operation */

        } else { \
                dce_error_inq_text(returned_st,\
           (unsigned char *)unexpected,  &dce_status); \
                dce_error_inq_text(expected_st,\
           (unsigned char *)expected, &dce_status); \
          printf("FAILED %s()\nresult:   %s\nexpected:   %s\n\n", \
                                func, unexpected, expected); \
        } \
} \

static unsigned char    unexpected[dce_c_error_string_len];
static unsigned char    expected[dce_c_error_string_len];
static int              dce_status;

void
main(void)
{
     rpc_binding_handle_t     bind_handle;
     rpc_ns_handle_t          lookup_context;
     rpc_binding_vector_p_t     bind_vec_p;
     unsigned_char_t          *entry_name;
     unsigned32          binding_count;
     cs_byte               net_string[SIZE];
     cs_byte               loc_string[SIZE];
     int               i, k, rpc_num;
     int               model_found, smir_true, cmir_true;
     rpc_codeset_mgmt_p_t     client, server;
     unsigned32          stag;
     unsigned32          drtag;
     unsigned16          stag_max_bytes;
     error_status_t          status;
     error_status_t          temp_status;
     unsigned char          err_buf[256];
     char               *nsi_entry_name;
     char               *client_locale_name;
     FILE               *fp_in, *fp_out;
     long                in_str_len; /* size in bytes of net_string */
```

```
nsi_entry_name = getenv("I18N_SERVER_ENTRY");


/* If the XPG/POSIX programming model is being used, set
 * the locale. In this way, the current locale
 * information is extracted from XPG/POSIX defined
 * environment variable LANG or LC_ALL.
 * Call dce_setlocale() to synchronize the program locale
 * with DCE's locale.
 */

setlocale(LC_ALL, "");
dce_setlocale(DCE_LC_ALL, "");

rpc_ns_binding_lookup_begin (
           rpc_c_ns_syntax_default,
     (unsigned_char_p_t)nsi_entry_name,
     cs_test_v1_0_c_ifspec,
           NULL,
     rpc_c_binding_max_count_default,
           &lookup_context,
           &status );

CHECK_STATUS(TRUE, "rpc_ns_binding_lookup_begin", status, rpc_s_ok);

rpc_ns_binding_lookup_next (
     lookup_context,
     &bind_vec_p,
     &status );

CHECK_STATUS(TRUE, "rpc_ns_binding_lookup_next", status, rpc_s_ok);

   rpc_ns_binding_lookup_done (
           &lookup_context,
           &status );

CHECK_STATUS(TRUE, "rpc_ns_binding_lookup_done", status, rpc_s_ok);

/*
 *  Get the client's supported code sets
 */
rpc_rgy_get_codesets (
     &client,
     &status );

CHECK_STATUS(TRUE, "rpc_rgy_get_codesets", status, rpc_s_ok);

binding_count = (bind_vec_p)->count;
for (i=0; i < binding_count; i++)
{
     if ((bind_vec_p)->binding_h[i] == NULL)
          continue;

     rpc_ns_binding_select (
          bind_vec_p,
          &bind_handle,
          &status );

     CHECK_STATUS(FALSE, "rpc_ns_binding_select", status, rpc_s_ok);

     if (status != rpc_s_ok)
     {
          rpc_ns_mgmt_free_codesets(&client, &status);
          CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codesets",
           status, rpc_s_ok);
     }
```

```
rpc_ns_binding_inq_entry_name (
     bind_handle,
     rpc_c_ns_syntax_default,
     &entry_name,
     &status );

CHECK_STATUS(TRUE, "rpc_ns_binding_inq_entry_name", status, rpc_s_ok);
if (status != rpc_s_ok)
{
     rpc_ns_mgmt_free_codesets(&client, &status);
     CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codesets",
      status, rpc_s_ok);
}

/*
 *  Get the server's supported code sets from NSI
 */
rpc_ns_mgmt_read_codesets (
     rpc_c_ns_syntax_default,
     entry_name,
     &server,
     &status );

CHECK_STATUS(FALSE, "rpc_ns_mgmt_read_codesets", status, rpc_s_ok);

if (status != rpc_s_ok)
{
     rpc_ns_mgmt_free_codesets(&client, &status);
     CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codesets",
      status, rpc_s_ok);
}

/*
 *  Start evaluation
 */
if (client->codesets[0].c_set == server->codesets[0].c_set)
{
     /*
      *  client and server are using the same code set
      */
     stag = client->codesets[0].c_set;
     drtag = server->codesets[0].c_set;
     break;
}

/*
 *  check character set compatibility first
 */
rpc_cs_char_set_compat_check (
     client
     server
     &status );

CHECK_STATUS(FALSE, "rpc_cs_char_set_compat_check",
      status, rpc_s_ok);

if (status != rpc_s_ok)
{
     rpc_ns_mgmt_free_codesets(&server, &status);
     CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codesets",
      status, rpc_s_ok);
}

smir_true = cmir_true = model_found = 0;

for (k = 1; k < server->count; k++)
{
```

```
        if (smir_true)
             break;
        if (client->codesets[0].c_set
             == server->codesets[k].c_set)
        {
             smir_true = 1;
             model_found = 1;
        }
   }

   for (k = 1; k < client->count; k++)
   {
        if (cmir_true)
             break;
        if (server->codesets[0].c_set
             == client->codesets[k].c_set)
        {
             cmir_true = 1;
             model_found = 1;
        }
   }

   if (model_found)
   {
        if (smir_true && cmir_true)
        {
             /* RMIR model works */
             stag = client->codesets[0].c_set;
             drtag = server->codesets[0].c_set;
             stag_max_bytes
                  = client->codesets[0].c_max_bytes;
        }
        else if (smir_true)
        {
             /* SMIR model */
             stag = client->codesets[0].c_set;
             drtag = client->codesets[0].c_set;
             stag_max_bytes
                  = client->codesets[0].c_max_bytes;
        }
        else
        {
             /* CMIR model */
             stag = server->codesets[0].c_set;
             drtag = server->codesets[0].c_set;
             stag_max_bytes
                  = server->codesets[0].c_max_bytes;
        }
   }
   else

   {

        /*
         *  We use UNIVERSAL code set
         */
             stag = UCS2_L1;
             drtag = UCS2_L1;
             rpc_rgy_get_max_bytes (
                     UCS2_L1,
                     &stag_max_bytes,
                     &status
              );
              if (status != rpc_s_ok)
              {

                  rpc_ns_mgmt_free_codesets(&server, &status);
```

```
                    CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codesets", status, rpc_s_ok);
                }
    }

          /*
           *  set tags value to the binding
           */
          rpc_cs_binding_set_tags (
               &bind_handle,
               stag,
               drtag,
               stag_max_bytes,
               &status );

          CHECK_STATUS(FALSE, "rpc_cs_binding_set_tags",
           status, rpc_s_ok);
          if (status != rpc_s_ok)
          {
               rpc_ns_mgmt_free_codesets(&server, &status);
               CHECK_STATUS(FALSE, "rpc_ns_mgmt_free_codesets",
                status, rpc_s_ok);
               rpc_ns_mgmt_free_codesets(&client, &status);
               CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codesets",
                status, rpc_s_ok);
          }
          else
               break;
}

rpc_ns_mgmt_free_codesets(&server, &status);
CHECK_STATUS(FALSE, "rpc_ns_mgmt_free_codesets", status, rpc_s_ok);

rpc_ns_mgmt_free_codesets(&client, &status);
CHECK_STATUS(TRUE, "rpc_ns_mgmt_free_codesets", status, rpc_s_ok);


rpc_ep_resolve_binding (bind_handle,
     cs_test_v1_0_c_ifspec,
     &temp_status);

CHECK_STATUS(TRUE, "rpc_ep_resolve_binding", temp_status, rpc_s_ok);

if(rpc_mgmt_is_server_listening(bind_handle, &status)
     && temp_status == rpc_s_ok)
{
     printf("PASSED rpc_mgmt_is_server_listening()");
}
else
{
     dce_error_inq_text ((unsigned long)status, err_buf,
      (int *)&temp_status);
     printf("is_server_listening error -> %s\n", err_buf);
}

fp_in = fopen("./i18n_input_data", "r");

if (fp_in == NULL)
{
     printf("i18n_input_data open failed\n");
}

fp_out = fopen("./i18n_tags_fixed_result_file", "w");

if (fp_out == NULL)
{
     printf("i18n_result_file open failed\n");
}
```

```
        (void)fgets((char *)net_string, SIZE, fp_in);
        while (!feof(fp_in))
        {
             temp_status = cs_conf_trans(bind_handle, in_str_len, net_string, loc_string);
             if (temp_status != rpc_s_ok)
             {
                  dce_error_inq_text(temp_status, err_buf, (int *)&status);
                  printf("FAILED %ld  MSG: %s\n", (unsigned long)temp_status, err_buf);
             }
             else
             {
                  printf("PASSED rpc #%d\n", rpc_num++);
                  (void)fputs((char *)loc_string, fp_out);
                  (void)fputs("", fp_out);
             }
        (void)fgets((char *)net_string, SIZE, fp_in);
        }

        fclose(fp_in);
        fclose(fp_out);

        return;
    }
```

Related Information

For more information, see:

- *OSF DCE Application Development Guide — Core Components*
- *OSF DCE Application Development Reference*

Unsupported Functions

RPC code set conversion is described in:

- *OSF Application Development Guide*— ″Writing Internationalized RPC Applications″
- *OSF Introduction to DCE*— ″Ensuring Character and Code Sets Interoperability″

DCE for Windows NT supports a more advanced version of code set conversion, that is compatible with other IBM DCE implementations.

The following are the DCE RPC API's that are not supported.

- **wchar_t_from_netcs**
- **wchar_t_local_size**
- **wchar_t_net_size**
- **wchar_t_to_netcs**

Chapter 15. Application Debugging with the RPC Event Logger

The DCE for Windows NT IDL compiler includes enhanced application debugging support beyond the support provided with OSF DCE. The IDL compiler includes the RPC Event Logger, a software utility that records information about operations relating to the processing of an application. Operational information about the program state at a specific point during processing, called an **event**, is recorded, as an event log, in a file or to a terminal screen.

The terms *event log* and, *log* refer to the stream of logging output captured in the event log file or displayed on the screen.

Event logging provides a detailed, low-level view of the processing of your RPC application. If development of your RPC application is proceeding well, this level of detail may not be necessary. However, when you are in the debugging phase of application development, the continuous processing information provided by the Event Logger and the ability to change the type and timing of logging can be valuable.

For more information, see:

Overview of the RPC Event Logging Facility

Generating RPC Event Logs

Controlling Log Information: Using Environment Variables and the Log Manager

Using the -trace Option, Environment Variables, and the Log Manager Together

Debugging Your Application: Using Event Logs

Event Names and Descriptions

Overview of the RPC Event Logging Facility

When you enable event logging, the Event Logger creates one log for each client and server process. To enable the RPC Event Logger, specify an IDL compiler option that traces events (as described in Enabling Event Logging).

Enabling event logging when compiling allows you the option of generating logs at runtime without rebuilding the application. Once logging is enabled, you can use environment variables and the RPC Log Manager (**rpclm**) to control logging operations. The Log Manager provides a command line interface for changing logging operations while the application is running.

The RPC Event Logger records events about application calls, context handles, errors, miscellaneous events, and logging operations. These are called event types. Typical RPC events include the following:

call_start
: A client application made a call to a server.

call_failure
: A client stub ended abnormally either through an exception or failing status.

exception
: An exception was detected in the server stub, and the exception caused the call to stop.

context_rundown
A context handle on a server was freed by the context rundown procedure.

The Event Logger generated events are as follow:

application calls
To signal call activation, the call start and end, attempts to rebind to a server, and stopping of a server thread.

context handles
To signal context handle creation and deletion by the client and server, and context handle modification, removal, and rundown.

errors To signal call and receive failure from the client, exceptions, server failure, and call transmission failure from the server.

miscellaneous events
To provide information about the application manager routine, and input and output argument processing events.

The logging operation itself generates events that display the logging output device, and that signal modification of logging parameters, and event log start and stop.

As a result of using the **-trace** option in the IDL compile command, **idl**, RPC events are generated by code in the client and server stub modules created by the compiler. Some events are generated at selected points in the RPC runtime library. For this reason, certain events, such as those relating to the logging operation, are always generated into the application code in addition to the event types you specify.

The events generated in each of these areas are shown in the following table. The first column lists events that can be generated, and the second column indicates whether the client or server, or both, can generate the event. For a complete description of each event see “Event Names and Descriptions” on page 290.

Event Types	
Event Name	**Origin**
Call Events	
activate	server
call_end	client
call_start	client
rebind	client
terminate	server
Context Handle Events	
client_ctx_created	client
client_ctx_deleted	client
client_ctx_destroyed	client
context_created	server
context_deleted	server
context_modified	server
context_rundown	server

Error Events	
call_failure	client
exception	server
receive_fault	client
status_fail	server
transmit_fault	server

Miscellaneous Events	
await_reply	client
manager_call	server
manager_return	server
receive	client

Logging Events	
internal_error	client, server
listening	client, server
log_events	client, server
log_file	client, server
log_start	client, server
log_stop	client, server

In the event log, each event is described on a single line divided into five fields. The five fields are defined in the following table.

Event Log Fields	
Field	**Field Description**
Event Time	The system clock at the time of the event. Events are listed chronologically in the log.
Thread Identity	The hostname, process ID, and thread ID.
Operation Name	The interface and operation name (if available).
Event Name	The name of the event.
Event Data	Data related to the event. This field contains either specific information about logging operations or a string binding that uniquely identifies the client process, server process, or Log Manager process.

The following is an example of an event log generated for an RPC client. The log contains five columns. To improve readability, columns four and five are shown below the first three columns. In addition, the field names have been added to identify the events; the names do not appear in an actual event log. (In subsequent event log examples, the field names are occasionally used instead of actual data to improve readability where necessary.)

```
EVENT  TIME                                          THREAD  IDENTITY       OPE
1993-02-07:11:48:18.31.160-5:00I0.121      ifdef:8710/1                binopwk.binopwk_
1993-02-07:11:48:18.32.170-5:00I0.121      ifdef:8710/1                binopwk.binopwk_
1993-02-07:11:48:18.65.180-5:00I0.121      ifdef:8710/1                binopwk.binopwk_
EVENT  NAME          EVENT  DATA
```

```
log_start                    all
call_start                   ncacn_ip_tcp:16.31.48.109[1821]
call_end
```

This small event log indicates that the following events occurred:

1. The **log_start** event indicates that logging started on February 7, 1993, at 11:48 a.m. on the host named **ifdef**, in process number 8710, and in thread number 1. Event logging was enabled when the **binopwk** interface was compiled with the IDL **-trace** option. The RPC call to the **binopwk_add** operation in the **binopwk** interface caused logging to begin and is the first event logged. The Event Data field indicates that all events are being logged.
2. The **call_start** event indicates an attempt to make a call to a server. The string binding in the Event Data field shows that the call was made over the TCP/IP transport to host 16.31.48.109 with endpoint 1821. This string binding identifies the server being contacted.
3. The **call_end** event indicates that the RPC call is completed, and control has returned to the caller of **binopwk_add**.

This log indicates that the RPC call to the **binopwk_add** interface was successful because no error events occurred.

Generating RPC Event Logs

To create an event log you must follow these four basic steps:

1. Specify the -**trace** option in your **idl** command line to enable event logging.
2. Compile and link the application.
3. Assign the event log to a filename or to the screen.
4. Run the application.

For information on how to use the **-trace** option see:

Enabling Event Logging
Using the -trace Option
Combining Event Logs
Disabling Event Logging

Enabling Event Logging

To enable event logging, specify the **-trace** option when you use the **idl** command to compile an interface. The syntax of the **idl** command with the **-trace** option is as follows:

```
C:\>idl filename -trace value
```

Event types are specified as a value of **-trace**. Valid values and the event types they denote are listed in the following table.

Event Values and Types

Value	Event Type
all	Log all events.
none	Disable all previously specified trace options.

Value	Event Type
calls	Log events relating to start and end of all RPC calls.
context	Log events relating to context handle creation, deletion, and rundown.
errors	Log errors.
misc	Log all miscellaneous events.
log_manager	Enable command interface support which allows modification at runtime of event logging options.

Using the -trace Option

The Event Logger generates a large volume of information for your analysis. Because the Event Logger continues to record information into the log files, the log files continue to grow until the disk is full; therefore, make sure you discard any unneeded log files.

To help reduce the generation of unwanted information, you can use the **-trace** options to enable event logging on only a subset of events. For example, instead of specifying the **all** option, specify only **calls** or only **context_handles**. The subset you specify depends on the part of your application you are debugging. Although the **-trace** option provides logging control on a per-compilation basis, the interface must be rebuilt to enable or disable logging of different event types. The **-trace** options offer the ability to select different event types for the various IDL interfaces that might make up a single application.

You can use the **-trace** option to request logging of a single type of event, such as **errors**, with a command similar to the following:

```
C:\> idl binopwk.idl -trace errors
```

You can also use the **-trace** option to request logging of multiple event types, such as **errors** and **calls** as shown below:

```
C:\> idl binopwk.idl -trace errors -trace calls
```

The above command enables the Event Logger, specifying error and call event logging.

To enable event logging to trace the RPC calls within a process, do the following:

1. Enable event logging by specifying the -**trace** option in the **idl** command you use to compile each interface definition. The following example specifies the **-trace all** option:

   ```
   C:\> idl binopwk.idltrace all
   ```

2. Build and link the client and server portions of the application.
3. Use the environment variable **RPC_LOG_FILE** to direct the log output for both the server and client processes. To store Event Logger output in a file, assign the environment variables to a filename.

   In the window where the server portion of the application will be running, direct logging for the server to a file with the following syntax:

   ```
   C:\> set rpc_log_file =server.log
   ```

   Or, to direct logging for the server to the screen (standard output), use the following syntax:

   ```
   C:\> set rpc_log_file=
   ```

where a specific space character is included immediately after the equal sign (=).

4. In the window where the client portion of the application will be running, direct logging for the client to a file using the following syntax:

   ```
   C:\> set rpc_log_file =client.log
   ```

   Or, to direct logging for the client to the screen, use the following syntax:

   ```
   C:\> set rpc_log_file=
   ```

   where a specific space character is included immediately after the equal sign (=).

Now you can invoke the client and server processes. The event log is recorded in the specified file or displayed on your screen when you run the application.

Combining Event Logs

Although event logs are generated locally for each process, you can combine event log files to give you a broader view of how the applications are running. This topic does not provide examples of each step in the application development process.

The syntax of the **sort** command is as follows:

```
C:\ > sort-m server-filename.log client-filename.log>client_and_server-filename.log
```

The **-m** option is specified, indicating that the files are already sorted and prevents reordering of events that occurred at the same time.

If two events have the same timestamp, a warning message displays after the sort is completed.

The following example illustrates how to combine logs from two different systems.

1. The server process command sequence is as follows:

   ```
   C:\> idl fpeserv.idl-trace calls-trace errors
   C:\> set rpc_log_file =server.log
   C:\> server
   ```

2. The client process command sequence is as follows:

   ```
   C:\> idl fpeserv.idl -trace calls -trace errors
   C:\> set rpc_log_file=client.log
   C:\> server
   ```

   These command sequences result in two log files: **server.log** and **client.log**, shown below. In the following example log files, the Event Data field is replaced by the word <data> to improve readability of the log.

   The example SERVER.LOG file is:

   ```
   1993-03-03:20:37:03.170-5:00I0.121  murp:17924/15  fpe.setup  log_start  <data>
   1993-03-03:20:37:03.170-5:00I0.121  murp:17924/15  RPC  Log  Mgr   listening  <data>
    1993-03-03:20:37:03.180-5:00I0.121  murp:17924/15  fpe.setup   activate   <data>
    1993-03-03:20:37:03.180-5:00I0.121  murp:17924/15  fpe.setup   terminate  <data>
    1993-03-03:20:37:03.200-5:00I0.121  murp:17924/15  fpe.float  <data>
    1993-03-03:20:37:03.200-5:00I0.121  murp:17924/15   transmit_fault  <data>
    1993-03-03:20:37:03.200-5:00I0.121  murp:17924/15  fpe.float     terminate   <data>
   ```

   The example CLIENT.LOG file is:

```
1993-03-03:20:37:02.850-5:00I0.121  ifdef:28168/1  fpe.stup  log_start  <data>
1993-03-03:20:37:02.880-5:00I0.121  ifdef:28168/1  fpe.stup   call_start  <data>
1993-03-03:20:37:03.190-5:00I0.121  ifdef:28168/1  fpe.stup    call_end   <data>
1993-03-03:20:37:03.190-5:00I0.121  ifdef:28168/1  fpe.flt      call_start      <data>
1993-03-03:20:37:03.210-5:00I0.121  ifdef:28168/1   receive_fault  <data>
1993-03-03:20:37:03.210-5:00I0.121  ifdef:28168/1   call_failure   <data>
```

3. Next, the two log files are combined and sorted with the **sort** command.

   ```
   C:\> sort -m client.log server.log> cli_serv.log
   ```

   The resulting CLI_SERV.LOG file is:

   ```
   1993-03-03:20:37:02.850-5:00I0.121  ifdef:28168/1  fpe.setup  log_start  <data>
   1993-03-03:20:37:02.880-5:00I0.121  ifdef:28168/1  fpe.setup  call_start  <data>
   1993-03-03:20:37:03.170-5:00I0.121  murp:17924/15  fpe.setup  log_start  <data>
   1993-03-03:20:37:03.170-5:00I0.121  murp:17924/15  RPC  Log  Mgr  listening  <data>
   1993-03-03:20:37:03.180-5:00I0.121  murp:17924/15  fpe.setup  terminate  <data>
   1993-03-03:20:37:03.190-5:00I0.121  ifdef:28168/1  fpe.setup   call_end   <data>
   1993-03-03:20:37:03.190-5:00I0.121  ifdef:28168/1  fpe.float    call_start  <data>
   1993-03-03:20:37:03.200-5:00I0.121  murp:17924/15  fpe.float   activate   <data>
   1993-03-03:20:37:03.200-5:00I0.121  murp:17924/15  fpe.float  exception  <data>
   1993-03-03:20:37:03.200-5:00I0.121  murp:17924/15  transmit_fault  <data>
   1993-03-03:20:37:03.200-5:00I0.121  murp:17924/15  fpe.float  terminate  <data>
   1993-03-03:20:37:03.210-5:00I0.121  ifdef:28168/1  receive_fault  <data>
   1993-03-03:20:37:03.210-5:00I0.121  ifdef:28168/1  call_failure  <data>
   ```

For the combined output to be accurate, the system clocks on all hosts on which event logs are generated must be closely synchronized. The Distributed Time Service (DTS) component of DCE for Windows NT provides such a service. Once the clocks are synchronized, the ordering of events in a combined log file is valid only if the difference between timestamps made on different machines is greater than the inaccuracy field in those timestamps. For more information about timestamps, see the DTS documentation in *OSF DCE Administration Guide*.

In the preceding CLI_SERV.LOG file example, consider the event with the timestamp 1993-03-03:20:37:03.180-5:00I0.121 and the event that follows it (these two event lines are separated from the rest of the log by a blank line on either side). The timestamps indicate that the terminate event precedes the **call_end** event.

However, you cannot determine this sequence of events by comparing timestamps because the inaccuracy value at the end of the timestamp is greater than the difference between the timestamps. That is, the difference in time between these events is only 10 milliseconds (the difference between 180 and 190 milliseconds). However, the inaccuracy in the timestamps is 121 milliseconds (I0.121). Therefore, the log is not a definitive indicator of which event occurred first. Because of the simplicity of the example and the single thread of control, you can assume that the terminate event preceded the **call_end** event.

Disabling Event Logging

To disable event logging, compile your interface without specifying the **-trace** option. For example:

```
C:\> idl binopwk.idl
```

Controlling Log Information,Using Environment Variables and the Log Manager

In addition to the **-trace** options, the Event Logger offers two other methods for controlling information in the event log. Each facility is advantageous in different circumstances, depending on the type of processes with which you are working and the type of events you need to log. The two methods are as follows:

- Controlling Logged Events with Environment Variables

  Select a subset of event types specified previously with the**-trace** option by creating the environment variable **RPC_EVENTS** . You assign the environment variable to the required event types before running the process. This method allows you to use event logging without rebuilding the interface; however, you must first stop the process or assign the environment variable before starting it. This method is also useful in cases where you specified all-inclusive event logging (such as with the **-trace all** option) but you determine while the application is running that you need only a subset of events.
- Controlling Logged Events with the RPC Log Manager

  Select a subset of event types specified previously with the **-trace** option by using the RPC Log Manager command interface. This method allows you to modify event logging parameters for a running image - there is no need to rebuild the interface or to stop and restart the process. In addition, you can use the Log Manager to modify event types specified with the environment variable **RPC_EVENTS**.

Controlling Logged Events with Environment Variables

One way to control the type of events logged is by assigning the environment variable **RPC_EVENTS**. This method is ideal for an application that contains a single RPC interface because environment variables provide control at the process level, rather than at the interface-by-interface level. However, to enable the environment variable you must first stop the client or server process.

To use environment variables to control event logging, first use the IDL **-trace** option in your **idl** compile command and then assign the log file with RPC_LOG_FILE. You can then use the environment variable **RPC_EVENTS** to reduce the number of events currently being logged. For example, if you used the **-trace errors** option to request error event logging, you can subsequently use only the environment variable to request logging of **errors** or **none**. You cannot use the environment variable to increase the number of event types to be logged. To do this, you must recompile the interface with the required **-trace** options.

The value of RPC_EVENTS is a list of event types separated by commas. The list identifies the event types to be monitored. Valid values are the same as those for **-trace** (except **log_manager**). These values are **all**, **none**, **calls**, **context**, **errors**, and **misc**.

An example command line follows:

```
C:\> set rpc_events =calls,errors
```

If the environment variable **RPC_EVENTS** was not assigned, then by default all of the events specified with the **-trace** option are written into the event log.

Controlling Logged Events with the RPC Log Manager

During application development, certain problems occur only after a server has processed some number of calls. Other problems may require more information than anticipated to debug. These problems can be addressed by enabling the RPC Log Manager in your application image. The Log Manager offers a command line interface (**rpclm**) for manipulating logging operations when the application is running. When you use the **rpclm** command line interface, you do not need to rebuild your interface or stop and restart your server or client process to manipulate logging operations.

The **rpclm** commands are shown in the following table:

Command Interface to rpclm	
Commands	**Description**
inquire	Inquire about the currently logged events and determine the name of the active log file.
log	Specify additional events to log. Valid values are **all**, **none**, **calls**, **context**, **errors**, and **misc**.
unlog	Disable logging of the specified event types. Valid values are **all**, **none**, **calls**, **context** , **errors**, and **misc** .
file	Change the output device or file to which events are logged.
quit	Terminate the **rpclm** session.
help	Display a description of **rpclm** commands.

Follow these steps to enable the RPC Log Manager to control event logging:

1. Use the **-trace log_manager** option in your **idl** compile command.
2. Create the **RPC_LOG_FILE** environment variable and assign it to a filename or to screen output.
3. Run the client or server process, or both.
4. When the first call is made to an interface compiled with the **-trace** option, a listening event is generated into the event log. Invoke the **rpclm** command interface (as specified in step 4 below) by specifying the string binding from the **listening** event.

Note: Only string bindings from a listening event can be used to invoke **rpclm**.

The **rpclm** command interface allows you to control event logging parameters from your keyboard. You can use the command interface to reduce the events currently being logged as well as to manipulate logging operations. You can enable or disable logging of different event types (within the set selected with the **-trace** option), store event logging in a file or display it on the screen, inquire about the current event types being logged, and display the name of the current log file.

The following procedure illustrates how to use the Log Manager:

1. When you compile your interface with the **idl** compile option, include the **-trace log_manager** option. For example:

   ```
   C:\> idl binopwk.idl-trace all-trace log_manager
   ```

2. Assign the **RPC_LOG_FILE** environment variable to a filename. For example:

   ```
   C:\>set rpc_log_file =client.log
   ```

3. Run the client or server process, or both.
4. Upon the first remote procedure call to an interface compiled with the **-trace log_manager** option, a **listening** event is generated into the log. Examine the Event Data field of the **listening** event in the log to determine the Log Manager string binding. (The RPC Event Logger is itself a client/server application: the Log Manager is a server process, and **rpclm** is its client. The **rpclm** client uses the string binding of the **listening** event to communicate with the Log Manager server.) Start **rpclm** and specify the Log Manager string binding. For example, consider the following event:

   <time> murp:17868/15 RPC Log Mgr listening ncacn_ip_tcp:16.31.48.144[3820]

   The **listening** event indicates that the RPC Log Manager is waiting for commands from **rpclm**. (In the example, the Time field is replaced by the word <time> to improve readability of the log.) To invoke **rpclm**, type the **listening** event string binding for this server process from the Event Data field as follows:

   ```
   C:\> rpclm "ncacn_ip_tcp:16.31.48.144[3820]"
   ```

   Note: You must enclose the string binding in double quotation marks (″ ″)
5. As you process **rpclm** commands, the Log Manager displays current logging parameters that indicate the changes made to event logging for this process. For example:

   ```
   rpclm> unlog all

          Event types:
          Events logged to terminal rpclm
   rpclm> log calls

          Event types: calls
          Events logged to terminal
   ```

   The log for this server process will have corresponding events logged as follows:

   ```
   <time>  murp:17868/15  RPC Log Mgr  log_events  none
   <time>  murp:17868/15  RPC Log Mgr  log_events  calls
   ```

   The following example illustrates a command dialog between the user and **rpclm**. The dialog begins when the user specifies a string binding from a listening event to **rpclm**.

   ```
   C:\> rpclm "ncacn_ip_tcp:cltdce[1821]"
   rpclm> help

          rpclm Commands:
          inquire- Display logged events and log filename
          log- Specify additional events to log
          unlog- Specify events that should no longer be logged
          file- Change file into which events are logged
          quit- Exit log manager
   rpclm> inquire

          Event Types: calls
          Events logged to terminal
   rpclm> log errors
   ```

```
        Event Types: calls errors
        Events logged to terminal
rpclm> file server.log

        Event Types: calls errors
        Events logged to file 'server.log'
rpclm> quit
```

In this dialog, typing the **help** command displays the **rpclm** commands and command descriptions.

Typing the **inquire** command displays the types of events being logged and the log filename. In this example, errors are being logged to the screen.

Typing the **log calls** command specifies that the Log Manager should start logging all events relating to calls, in addition to error events.

The user then types the **file** command and specifies a filename. This command requests that **rpclm** change its output device from the terminal screen to a file named **server.log**.

Typing **quit** ends the **rpclm** session.

Using the -trace Option, Environment Variables, and the Log Manager Together

This topic describes a few different ways to use the **-trace** options, environment variables, and the Log Manager together. When you are learning to use the Event Logger, one possible approach is to specify all-inclusive event logging with the **-trace all** IDL compilation option, and then examine the event log to get an understanding of typical output. You can then use the environment variable **RPC_EVENTS** to log only those events needed, such as **calls** or **errors**.

In the case of a running process that you do not want to stop, use a different method.

1. Enable the Event Logger, specifying logging of all events, and enable the Log Manager. Type:

   ```
   C:\> idl filename -trace all-trace log_manager
   ```

2. Set the event log to display on the screen. Type:

   ```
   C:\> set rpc_log_file  where a specific space character is
                                                   included immediately after the
   ```

3. Assign the **RPC_EVENTS** environment variable so it will not log any event types. Type:

   ```
   C:\>set rpc_events =none
   ```

With these parameters set, the only event that is displayed is the **listening** event once the first call is made to a server interface compiled with the **-trace log_manager** option. You can then obtain the string binding for the process and use it later, if needed. Once you start the process, if an error occurs, use the string binding to invoke the **rpclm** command interface and log the needed events. Any **rpclm** commands issued at this point will modify the **RPC_EVENTS** environment variable assignment. For example, if you assign the environment variable **RPC_EVENTS** to **calls** and then issue a command to **rpclm** to **log errors**, errors as well as calls are logged.

Once you are familiar with Event Logger output, consider using the command interface regularly to enable or disable subsets of event types as needed.

This topic provides an example of common tasks you may need to perform during event logging. In this particular example, a distributed server process provides a mathematical calculation service. The client process passes data to be calculated to the server process. This type of processing often generates exception events such as those in the example event log. That is, some operations are interrupted by floating point overflow and integer division by zero exceptions, as well as others. This example uses **rpclm** to control logging of a server process; however, you can also use **rpclm** to control event logging for a client process.

The following processes are shown in three windows: a server process window, a client process window, and an **rpclm** window.

1. **Server Window:**

   The user enables the RPC Event Logger by specifying the **-trace all**and**-trace log_manager** options in the **idl**command line:

   ```
   C:\>idl server.calc -trace all -trace log_manager
   ```

2. **Server Window:**

   The user starts the server process. The server receives a client call and initializes the RPC Log Manager. The environment variables were assigned to enable event logging with no event types selected, so only Log Manager events are output, as shown. (The endpoint displayed for the **listening** event is the endpoint of the Log Manager.)

   ```
   C:\> set rpc_log_file=
   ```

   where a specific space character is included immediately after the equal sign (=).

   ```
   C:\> set rpc_events =none
   C:\> server ncacn_ip_tcp
   ```

   ```
   <time> murp:17868/15 fpe.setup log_start none
   <time> murp:17868/15 RPC Log Mgr listening ncacn_ip_tcp:16.31.48.144[3820]
   ```

3. **Client Window:**

   The user invokes the client process. The specified string binding is used to find the server. The client process displays the output PASS 1 upon completion.

   ```
   C:\> client ncacn_ip_tcp 16.31.48.86 [3123]
   PASS 1
   ```

4. **rpclm Window:**

   The user invokes **rpclm** and specifies the string binding displayed in the **listening** event output by the server process, shown in step 2. The string binding must be enclosed in double quotation marks (″ ″). The user issues the **inquire** command, and the event logging parameters for the server process are displayed. The Log Manager reply indicates that no event types are enabled and that the event log is being displayed on the screen from which the server process was started. The user issues the **log errors** command to enable logging of error events for the server process.

   ```
   C:\> rpclm "ncacn_ip_tcp:16.31.48.144[3820]"
   rpclm> inquire

          Event types:
          Events logged to terminal
   rpclm> log errors
   ```

Event types: **errors**
Events logged to terminal

5. **Client Window**

   The user invokes the client process a second time. The error events that occur when the server is running, are logged to the server window. The client process displays the output PASS 2 upon completion.

   ```
   C:\> client ncacn_ip_tcp 16.31.48.86 [3123]
   CPASS 2
   ```

6. **Server Window**

   The server process receives the command from **rpclm** to start logging errors. Any errors that occur in the server process are logged.

   ```
   <time> murp:17868/15 RPC Log Mgr        log_events                errors
   <time> murp:17868/15 fpe.flt_overflw    exception                 Floating point

   <time> murp:17868/15                                  transmit_fault          rpc_s_fau
   <time> murp:17868/15 fpe.flt_underflw   exception                 Floating point und
   <time> murp:17868/15                                  transmit_fault          rpc_s_fa
   <time> murp:17868/15 fpe.flt_divbyzer   exception                 Floating point/


   <time> murp:17868/15                                  transmit_fault          rpc_s_fau
   <time> murp:17868/15 fpe.dble_overflw   exception                 Floating point

   <time> murp:17868/15                                  transmit_fault          rpc_s_fau
   <time> murp:17868/15 fpe.dble_underflw  exception                 Floating point

   <time> murp:17868/15                                  transmit_fault          rpc_s_fau
   <time> murp:17868/15 fpe.dble_divbyzer  exception                 Floating point/decim

   <time> murp:17868/15                                  transmit_fault          rpc_s_fa
   ```

7. **rpclm Window**

   The user issues the **unlog all** command to disable logging of all previously specified event types

   ```
   rpclm> unlog all
   ```

   Event types:
   Events logged to terminal

8. **Server Window**

   The event log now contains an entry that indicates the Event Logger will stop logging previously specified events.

   ```
   <time>  murp:17868/15  RPC Log Mgr  log_events  none
   ```

9. **rpclm Window**

   The user issues a **log calls** command to enable logging of call events.

   ```
   rpclm> log calls
   ```

   Event types: **calls**
   Events logged to terminal

10. **Server Window:**

    The newest event log entry indicates that the Event Logger will start logging call events.

<time> murp:17868/15 RPC Log Mgr log_events calls

11. **rpclm Window:**

    Because logging output will increase now that call events are being logged, the user issues an **rpclm** command to redirect logging output to a file named **servcalc.log**. When the application stops and logging is complete, the user can use a text editor to view and search for entries in the log. This log file contains only those call events from the server process.

```
rpclm> file servcalc.log

        Event types: calls
        Events logged to file 'servcalc.log'
```

12. **Server Window:**

    The newest event log entry indicates that the logger starts redirecting logging information to file **servcalc.log**.

<time> murp:17868/15 RPC Log Mgr log_file servcalc.log

13. **Client Window:**

    The user invokes the client process a third time. The call events that occur when the server is running are logged to **servcalc.log** file. The client process displays the output PASS 3 upon completion.

```
C:\> client ncacn_ip_tcp 16.31.48.86 [3123]
PASS 3
```

14. **Server Log:**

    This is log file **servcalc.log**

<time> murp:17868/15 RPC Log Mgr log_start server_calc.log
<time> murp:17868/15 fpe.setup activate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.setup terminate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.flt_overflw activate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.flt_overflw terminate
<time> murp:17868/15 fpe.flt_underflw activate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.flt_underflw terminate
<time> murp:17868/15 fpe.flt_divbyzer activate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.flt_divbyzer terminate
<time> murp:17868/15 fpe.dble_overflw activate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.dble_overflw terminate
<time> murp:17868/15 fpe.dble_underflw activate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.dble_underflw terminate
<time> murp:17868/15 fpe.dble_divbyzer activate ncacn_ip_tcp:16.31.48.109[2905]
<time> murp:17868/15 fpe.dble_divbyzer terminate

15. **rpclm Window**

    The user issues a **file** command to redirect event logging output from **server_calc.log** to the terminal screen. To do this, press the Return key without specifying a filename when the Log Manager prompts for one.

```
rpclm> file

        New File Name: <Return>
        Event types: calls
        Events logged to terminal

rpclm>
```

16. **Server Window**

The final event in the **servcalc.log** file is a log_file event, which indicates that event logging output is being redirected, in this case to the terminal screen. Therefore, no filename is displayed to the right of the event name.

<time> murp:17868/15 RPC Log Mgr log_file

Debugging Your Application, Using Event Logs

The RPC Event Logger is designed to help you debug your distributed application and is an enhancement over the basic diagnostics in the RPC product. The diagnostics alone provide minimal information. For example, the sample program called **test2**, which is provided with the DCE software kit, generates the rpc_x_no_more_bindings exception when the client fails to contact the server. Without the aid of RPC event logging, this is the only diagnostic information available.

The following example shows the basic RPC diagnostic information that an application displays when an error occurs.

```
C:\> test2
*** Unable to obtain server binding information
Make sure environment variable RPC_DEFAULT_ENTRY = .:/test2_server
Exception: no more bindings (dce / rpc)
IOT trap (core dumped)
```

If you enable RPC event logging by defining the environment variable **RPC_LOG_FILE**, the details of client execution are captured in a file. From the event log, you can determine which servers the client tried to contact and the reason each attempt failed.

In the following event log example, the Event Data field on the rebind events indicates that the interface is not registered in the endpoint map and that a communications failure occurred. This information indicates that the server either is not running or it failed to register properly with the endpoint mapper.

The final event, **call_failure**, indicates that the call ended with the no more bindings status. This event indicates that the client tried all available servers but failed to communicate with any of them. (In the example, the word <time> represents the actual value for time.)

```
C:\>test2

<time> ko:11436/1 test2.test2_add      log_start    all
<time> ko:11436/1 test2.test2_add      call_start       ncacn_ip_tcp:16.20.16.27[ ]
<time> ko:11436/1 test2.test2_add      rebind             not registered in endpoint map(c
<time> ko:11436/1 test2.test2_add      call_start       ncacn_dnet_nsp:4.262[]
<time> ko:11436/1 test2.test2_add        rebind           not registered in endpoint

<time> ko:11436/1 test2.test2_add        call_start       ncadg_ip_udp:16.20.16.27[]
<time> ko:11436/1 test2.test2_add        rebind           communications failure (dce/rpc)
<time> ko:11436/1 call_failure                                  no more bindings (dce/rp
*** Unable to obtain server binding information
Make sure environment variable RPC_DEFAULT_ENTRY = .:/test2_server
Exception: no more bindings (dce / rpc)
IOT trap (core dumped)
```

Event Names and Descriptions

This topic lists and describes RPC events. See the Overview of the RPC Event Logging Facility table for a list of events by type (calls, context handles, errors, miscellaneous, and logging) and their origin (client or server).

RPC Events	Description
activate	A thread was assigned to process an RPC call on a server, and the server stub has started processing input arguments. The Event Data field of the event log contains the string binding of the client application making the call.
await_reply	The transmission of input arguments in a call from a client application to a server is completed. The event is generated by the client stub. The client application is waiting for output arguments from the server.
call_end	A call from a client application is complete and the client stub is returning to the caller.
call_failure	A client stub terminated abnormally because either an exception occurred or a failing status was returned. The Event Data field of the event log contains the error text associated with the exception or RPC status code.
call_start	A client application attempted to make a call to a server. The event is generated by the stub within the client application. The Event Data field of the event log displays the string binding of the server being contacted.
client_ctx_created	A client application has allocated a context handle on a particular server. The Event Data field of the event log contains the following information about this event: the address representing the context handle in the client address space (an opaque pointer), the UUID which can be used to identify the corresponding context handle on the server, and the string binding of the server on which the actual context resided.
client_ctx_deleted	The client application representation of a context handle is being deleted to reflect the deletion of the context handle on the server. The Event Data field of the event log contains the following information about this event: the address representing the context handle in the client address space (an opaque pointer), the UUID which can be used to identify the corresponding context handle on the server, and the string binding of the server on which the actual context resided
client_ctx_destroyed	A client application has destroyed the client representation of a context handle through the **rpc_ss_destroy_client_context()** routine. The Event Data field of the event log contains the following information about this event: the address representing the context handle in the client address space (an opaque pointer), the UUID, which can be used to identify the corresponding context handle on the server, and the string binding of the server on which the actual context resided
context_created	A new context handle was created on a server and returned from the application manager routine. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.
context_deleted	A context handle on a server has been deleted by the application manager routine. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.
context_modified	A context handle on a server was returned from the application manager routine with a value that is different from its previous value. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.

context_rundown	A context handle on a server was freed by the context rundown procedure. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.
exception	An exception was detected in the server stub, and the exception caused the call to terminate. The Event Data field of the event log contains a text description of the exception.
internal_error	A failure occurred in the support routines that manage the Event Logger. Check the Event Data field of the event log for a description of the cause of the event. If the error does not seem to indicate a transient network problem or an environmental failure, report the failure in a Software Performance Report (SPR).
listening	The RPC Log Manager has started to listen for **rpclm** commands. The listening event is generated by the portion of the RPC Log Manager built into your application by the RPC runtime when you specify the **-trace log_manager** option on your IDL compilation. The RPC Log Manager services the requests generated by the **rpclm** command. You use one of the string bindings from a listening event to invoke the **rpclm** command interface.
log_events	Event logging was modified through the Log Manager command interface **rpclm**. The Event Data field of the event log contains the new set of events being logged.
log_file	Event logging was modified through the Log Manager command interface **rpclm**. The Event Data field of the event log contains the new filename for the event log. If no filename is displayed, events are being logged to the screen.
log_start	A new event log was created or event logging was resumed after being suspended by a user command to the Log Manager command interface **rpclm**. The Event Data field in the event log contains a list of event types being logged.
log_stop	Event logging was stopped through the Log Manager command interface **rpclm**.
manager_call	The server stub is about to call the application manager routine.
manager_return	Control has just returned from the application manager routine to the server stub.
rebind	A call from a client application to a server failed. The Event Data field in the event log shows the reason for the failure to contact the server. The event is generated by the stub within the client application. The call failed on an auto_handle operation and the client is attempting to rebind to the next server.
receive	Following the transmission of input arguments from a client application call to a server, the client received a reply and has started processing output arguments.
receive_fault	The client received a fault indicating a failure on the server. The Event Data field of the event log contains the RPC status that identifies the failure. All failures have fault codes which you can find in the file NCASTAT.IDL. If the fault code in the NCASTAT.IDL file is too general (such as unspecified fault), examine the server event log for precise failure information.
status_fail	A failure status was encountered in the server stub. The Event Data field of the event log describes the failure.
terminate	The server thread has completed processing the call and has ended.

transmit_fault	The server runtime is sending fault information to the client application. The Event Data field of the event log indicates the name of the fault being sent. The fault information in this field is listed in the NCASTAT.IDL file. The fault information in this field may be less descriptive than the information logged about the actual error. To obtain precise failure information see the exception or **status_fail** events in the event log.

Summary: RPC Event Logger

The RPC Event Logger is a developer's aid for debugging DCE RPC applications. The RPC Event Logger allows you to modify IDL-generated stub routines in order to generate event logs of runtime execution of RPC calls on the screen or in a file. In addition, the RPC Log Manager command interface **(rpclm)** provides command line access to event logging parameters, allowing you to enable and disable debugging support of clients and servers as they execute.

The DCE RPC application development environment is designed to create applications that are portable to other DCE platforms and that can interoperate with other DCE applications. Use of the DCE RPC Event Logger does not affect code portability or interoperability. Because the Event Logger does not modify the application, you can take advantage of event logging without affecting application portability to other hardware or software platforms.

In addition, use of the DCE RPC Event Logger does not limit interoperability with other DCE implementations. Because event logs are generated only in the local application, communication protocols are not modified. You can, for example, use the event logging facility with any server process running under DCE or with any client process communicating with an RPC server on any hardware or software platform.

Chapter 16. Using DTS Time Providers

DCE for Windows NT includes DTS time provider applications that work in conjunction with a distributed time service to provide an accurate time source. When a time provider is started, it becomes the DTS service's sole time provider.

DTS time providers are provided as source code on the DCE kit in the %DCELOC%\dcelocal\examples\dtss directory.

For more information, see:

"Building DTS Time Providers"

"Null Time Provider"

"Starting the Null Time Provider as a Foreground Process"

"Starting the Null Time Provider as a Native NT Service" on page 294

"NTP Time Provider" on page 294

Building DTS Time Providers

To build the DTS time providers, type:

```
x:\> cd %DCELOC%\dcelocal\examples\dtss
x:\> nmake -f providers.mak
```

Null Time Provider

The ″null″ time provider (**dts_null_provider.exe**) implements a time source that assumes the host's time is already synchronized in some manner (for example, through the use of NTP or specialized hardware).

Starting the Null Time Provider as a Foreground Process

To run the program interactively, type:

```
%DCELOC%\dcelocal\examples\dtss> dts_null_provider.exe
```

The provider asks for values for the following parameters:

Poll period
: The number of seconds between time service queries. For example, a value of 300 causes the provider to supply the inaccuracy to the time service every 5 minutes.

Base inaccuracy
: The number of milliseconds of systematic inaccuracy in the timestamps delivered by the host. The default value is 100 milliseconds.

Output trace flag
: Enables or disables the logging of TP tracing information to standard output. Setting the output trace flag to TRUE causes information to be logged.

To view the valid command line arguments, start the provider interactively, and then exit.

Starting the Null Time Provider as a Native NT Service

To run the DTS Null Time Provider Service as a native NT service, type:

```
C:\> net start dts_null_provider
```

To stop the service, type:

```
C:\> net stop dts_null_provider
```

NTP Time Provider

The Network Time Protocol (NTP) is an Internet recommended standard for distributing time. The NTP time provider (**dts_ntp_provider.exe**) assumes that the user is familiar with the NTP protocol and has an NTP server available as a time source.

To run the program interactively, type:

```
%DCELOC\dcelocal\examples\dtss> dts_ntp_provider.exe
```

The provider asks for values for the following parameters:

Internet host name of the NTP server to query
: This can be specified as a name or an address (decwet.dec.com or 16.1.0.4).

Poll period
: This is the number of seconds between NTP queries. For example, a value of 300 causes the provider to query the NTP server every 5 minutes.

Base inaccuracy
: This is the number of milliseconds of systematic inaccuracy in the timestamps delivered by NTP. The default value is 30 milliseconds. If the times returned by the provider are systematically incorrect, the base inaccuracy may need to be increased.

Number of timestamps read at each synchronization
: The range is 1 to 6 readings and the default is 4.

Disallow clock set flag
: This causes the service to enable or disable clock adjustments that DTS would otherwise cause. This is useful during TP development. Setting the disallow clock set flag to TRUE disables clock adjustments.

Output trace flag
: This enables or disables the logging of TP tracing information to standard output. Setting the output trace flag to TRUE causes information to be logged.

To view the valid command line arguments, start the provider interactively, and then exit.

Chapter 17. Using the Name Service Interface Daemon

The Name Service Gateway, also known as the **nsid**, provides access to DCE name services from native Microsoft RPC applications. The **nsid** runs on one or more DCE systems in the cell and performs the operations of the DCE RPC name service interface (NSI). It does this on behalf of a PC client (MS-DOS, Windows, and Windows NT) or a PC server (Windows NT) that has RPC services only and no other DCE services. Through a hidden level of indirection, the **nsid** allows the PC to appear as if it is directly involved in the broader cell namespace.

For more information, see:

How nsid Works

An application on a PC running Microsoft Windows makes a call to familiar name-service procedures, such as **rpc_ns_binding_export** (or, in the Microsoft native format, **RpcNsBindingExport**). Within these procedures, the parameters are passed using RPC to the nsid. The nsid receives the parameters from the PC, converts them to native DCE format, and makes a call to the native **rpc_ns_binding_export** procedure that corresponds to the procedure called on the PC.

The CDS server receives the parameters, performs the requested operation, and returns the results to the nsid. The nsid converts the results back into a format the PC caller can understand and returns them to the PC using RPC. The PC client now has the results of the call and can take appropriate action.

The system where the nsid is running can be a different system from where the CDS server is running, because any operation defined by the NSI can be called from any member of the cell.

Configuring and Starting the nsid

Use the DCEsetup utility to configure and start the **nsid**. After it is configured, the **nsid** is added to your DCE configuration and is started along with all other DCE components.

Note: In a split server configuration, the CDS master has to be configured before **nsid** can be configured.

Security Considerations

RPC communication between the client system (using the services of the **nsid**) and the system running the **nsid** uses unauthenticated Microsoft RPC. The **nsid** runs under the fixed principal, **pc-user**. Communication between the system running the **nsid** and the DCE Cell Directory Service is authenticated under this principal. In order for the **nsid** to access entries in the DCE namespace on behalf of the client system, you must modify the access control lists (ACLs) on the namespace entries to authorize access by the **nsid** principal. However, if the namespace entry **.:/subsys/DCE/pc** is used by the client system, you do not need to modify the ACLs. The ACLs are preset with authorized access for the **nsid** principle pc-user. For example, a MSRPC server can export an interface named ″foo″ with the cds entry name **.:/subsys/DCE/pc/foo** without modifying the ACLs. A MSRPC client can then import a binding to that interface using the same cds entry name.

The Microsoft Locator and the nsid

The Locator is Microsoft's simple, flat-namespace directory service. The Locator exports the Microsoft version of the RPC name service interface (NSI) and makes an association between entry-name strings and string bindings.

The Locator exports the identical interface as the nsid. The caller of the Microsoft NSI makes a remote procedure call to either the Locator or the nsid based entirely on the string-binding components defined at the time in the Registry.

The Microsoft Registry and the nsid

The Registry defines the name server that is queried when any of the **rpc_ns_*** procedures are called. The name server can be either the Microsoft Locator or the nsid; after installation of Windows NT, the default setting is the Locator.

Modifying the Windows NT Registry Using the Windows NT Control Panel

The RPC Name Service Provider is usually changed through the Network applet in the Control Panel. To perform this task, do the following:

1. Highlight **RPC Name Service Provider** under **Installed Network Software**, and click on **Configure**.
2. Select **DCE Cell Directory Service** under **Name Service Provider**, and supply the IP address (or the DECnet address) in the Network Address box.
3. Click on OK to complete the operation, unless you want to supply a default name-service entry (see “Modifying the Windows NT Registry Using the Registry Editor” on page 297).

Modifying the Windows NT Registry Using the Registry Editor

If users enable nsid using DCE Setup, the Windows NT Registry is updated automatically for nsid operation. Otherwise you can modify the Registry using the registry editor **(regedit.exe)**. You must use this method to add the default name-service entry. On DCE platforms, the default name-service entry is specified as an environment variable.

To set up the Registry on Windows NT to use the nsid, do the following:

1. From a command window, type `regedt32` to display the Registry Editor window.
2. Click on the HKEY_LOCAL_MACHINE window.
3. Double-click **SOFTWARE**, **Microsoft**, **Rpc**, and **NameService**. The right half of the HKEY_LOCAL_MACHINE window now lists the parameters for the RPC Name Service Provider.
4. Change the settings as follows:

   Setting
   New Value

   DefaultSyntax
   Either 0 or 3

   Endpoint
   No value

   Protocol
   One of the protocol sequences that the NT system uses to communicate with the **nsid**

   NetworkAddress
   Network address of the system where the **nsid** is running

   ServerNetworkAddress
   Network address of the system where the **nsid** is running

   You can change all of the values by double-clicking on them; a window is displayed that allows you to change the value.
5. If you want to provide a default name-service entry, on the **Edit** menu, click **Add Value**.
6. For Value Name, type `DefaultEntry` in the syntax shown here. The default data type, REG_SZ, is correct.
7. Click **OK**, then in the **Next** box provide your default name-service entry.

After the task is completed, the right half of the HKEY_LOCAL_MACHINE window displays parameter information similar to the following:

```
DefaultEntry:REG_SZ:/.:/Foobar
DefaultSyntax:REG_SZ:3
Endpoint:REG_SZ:
NetworkAddress:REG_SZ:16.64.0.79
Protocol:REG_SZ:ncacn_ip_tcp
ServerNetworkAddress:REG_SZ:16.64.0.79
```

To finish setting up the Registry, on the **Registry** menu, click **Exit**.

APIs Supported by the nsid

All of the NSI is supported by the **nsid**, but not on all platforms. For instance, because DOS and Windows provide only client RPC services, server-only APIs are not supported.

The following list of APIs is in the DCE style. The native Microsoft style is slightly different. For example, **rpc_ns_group_delete** appears as **RpcNsGroupDelete** in the Microsoft native style. A **dceport.h** include file is provided to enhance application portability on Microsoft platforms.

rpc_ns_binding_import_begin
rpc_ns_binding_import_next
rpc_ns_binding_import_done
rpc_ns_binding_lookup_begin
rpc_ns_binding_lookup_next
rpc_ns_binding_lookup_done
rpc_ns_binding_export (not supported on DOS or Windows)
rpc_ns_binding_unexport(not supported on DOS or Windows)
rpc_ns_group_mbr_add
rpc_ns_group_mbr_remove
rpc_ns_group_delete
rpc_ns_mgmt_inq_exp_age
rpc_ns_mgmt_set_exp_age
rpc_ns_profile_elt_add
rpc_ns_profile_elt_remove
rpc_ns_profile_eelete
rpc_ns_mgmt_entry_create
rpc_ns_mgmt_entry_delete
rpc_ns_mgmt_entry_inq_if_ids
rpc_ns_mgmt_binding_unexport
rpc_ns_entry_expand_name
rpc_ns_group_mbr_inq_begin
rpc_ns_group_mbr_inq_next
rpc_ns_group_mbr_inq_done
rpc_ns_profile_elt_inq_begin
rpc_ns_profile_elt_inq_next
rpc_ns_profile_elt_inq_done
rpc_ns_entry_object_inq_begin
rpc_ns_entry_object_inq_next
rpc_ns_entry_object_inq_done

Chapter 18. Using the Example Programs

The DCE for Windows NT Application Development Kit provides a full range of example programs, including those designed specifically for a Windows NT environment. Also included are programs written in C++ that illustrate use of distributed objects.

Several example programs are supplied with the DCE for Windows NT Application Development Kit. These programs are located in directories under %dceloc%\dcelocal\examples. In addition to the information provided here, each example program includes an online **readme** file located in the same directory as the program. The following table shows the different features of each example program.

Example Program	Description
RPC Test Program #1	Makes minimal use of DCE services. Server does not register endpoints; binding information not exported to namespace.
RPC Test Program #2	Makes minimal use of DCE services. Server registers endpoints; binding informa exported to namespace; uses security.
RPC Test Program #3	Makes minimal use of DCE services. Server registers endpoints; binding informa exported to namespace.
Bank Program	Simulates an automated teller machine. Uses all DCE services, including securit
Timop Program	Calculates the span of time it takes a server to perform an operation. Uses all D services, including security and threads.
PC Phone book Program	Looks up employee contact information that resides with the **phnbk** server.Uses RPC and the name service.
Mandelbrot Set Program	Uses multiple DCE RPC servers per client; also uses the name service.
DTSS Programs	Are time provider services that work in conjunction with a DTSS service to provi accurate time source.
GSSAPI Program	Shows how a distributed application can make itself secure by using the GSSAP
Stock Quote Program	Uses two DCE RPC servers distributing data to multiple clients and the name se
Tic-Tac-Toe Program	Uses two DCE RPC clients interoperating with a server and the name service
Virtual Whiteboard Program	Uses multiple DCE RPC clients exchanging large amounts of data in real-time w and the name service.
Account Program	Tests inheritance, binding to an object using another interface, binding to an obje unsupported interface, and the reflexive, symmetic, and transitive relation proper bind() API.
Accountc Program	Tests the same properties as the account program , but uses the C interfaces fo APIs.
Card Program	Tests the passing of C++ objects as parameters using the [cxx_delegate] attribut polymorphism property of the base class.
Stack Program	Tests the passing of C++ objects as parameters using the [cxx_delegate] attribut user defined Stack class.
EMS Consumer and Supplier Programs	Shows how basic consumer and supplier programs can be written to work in cor with EMS for simple event transmission.
Serviceability Programs	Shows how to use DCE Messaging and Serviceability APIs.

You should copy the example program files to a private area before you attempt to build them.

To build the example programs, do the following:

1. Use the Windows Explorer or a command such as **xcopy** to copy the files in %dceloc%\dcelocal\examples ***program_directory*** * to your own directory. For example:

```
   C:\> cd \mydir
C:\MYDIR> xcopy /s %DCELOC%\dcelocal\examples\test1
```

2. Build the test program using the provided makefile. For example:

```
   C:\MYDIR>nmake -f makefile.test1
```

Notes:

1. For Visual C++ Version 4.2 or greater, you need to modify the *.**mak** makefiles in the **xidl** directory.
2. The example makefiles assume that the C compiler has been fully configured in a standard manner. For Visual C++, one way to do this is to include the statement

   ```
   call <msdevdir>\vcvars32.bat
   ```

   in your **autoexec.bat**.

Chapter 19. Enhanced Online Information

Extensive online documentation is shipped as part of DCE for Windows NT. Both online OSF DCE books and the DCE for Windows NT product documentation are provided as native Windows Help files. In addition the online information has been enhanced to provide the following:

- OSF documentation set, in native Windows Help format
- Additional product information
- Important online information to help troubleshoot problems
- Improved viewing, navigation, and print capabilities.

For more information, see:

"Online Documents"

"Online Help Files"

Online Documents

The following books are available online:

- *Quick Beginnings*
- *Introduction to OSF DCE*
- *OSF DCE Command Reference*
- *OSF DCE Adminstration Guide - Inroduction*
- *OSF DCE Adminstration Guide - Core Components*
- *OSF DCE Application Development Guide - Introduction and Style Guide*
- *OSF DCE Application Development Guide - Core Components*
- *OSF DCE Application Development Guide - Directory Services*
- *OSF DCE Application Development Guide - Reference*
- *DCE Problem Determination Guide*
- *Guide to DECthreads*
- *DCE Enhancements*
- *Troubleshooting*

Online Help Files

DCE for Windows NT provides the following value-added files in Windows help file format:

- *Using DCEsetup*
- *Using Visual ACL Editor*
- *Using DCE Director*

Chapter 20. Additional Considerations

This section outlines the compatibility and interoperbility issues with other DCE systems. Refer to the README file to review the latest updates.

For more information, see:

Readme File

All last minute information is documented in the online readme file that is located in the following directory where you installed the program.

```
%DCELOC%\dcelocal\bin\readme.txt
```

where %DCELOC% is the drive and directory in which you have your program files.

Compatibility and Interoperability with Other DCE Systems

DCE for Windows NT has been tested with and is compatible with most other vendor DCE products that are based on the OSF DCE R1.1, R1.2.1, or R1.2.2 code bases.

This product provides interoperability and source-level runtime compatibility with DCE systems from other vendors, as long as DCE implementations and applications conform to the OSF DCE Application Environment Specification (AES).

Note: If your system is configured with CDS Version 3.0 and you configure a secondary CDS server specifying a Directory Version of 4.0, there may be compatibility problems. These problems are caused by the acl format differences between the 3.0 and 4.0 versions of the CDS server.

If you plan on testing or debugging with this type of configuration, you need to do the following:

Configure your AIX machine with its CDS server. Before you configure any NT machines with additional CDS servers, for every directory created at configuration time for the primary CDS server, which includes:

```
/.:
/.:/hosts
/.:/hosts/<AIX  machine  name>
/.:/subsys
/.:/subsys/dce
/.:/subsys/dce/dfs
/.:/subsys/dce/sec
/.:/users
```

Type the next four commands::

```
dcecp -c directory modify  -add {CDS_UpgradeTo 4.0} -single
dcecp -c directory synchronize
dcecp -c clearinghouse verify /.:/<primary_cds_ch>
dcecp -c directory synchronize <dir>
```

If the versions do not match, one of the following error messages is displayed:

Replica cannot be added to old Clearinghouse.

Old replica cannot be included in new replica set.

An error occurred trying to create the clearinghouse for the secondary cds server.

Interoperability with Microsoft RPC on Windows NT Systems

Microsoft provides DCE-compatible Remote Procedure Call (RPC) component with the Windows NT operating system. Although functionally compatible, the Microsoft RPC uses different routine names and syntax. With DCE for Windows NT, you can write portable code directly to the OSF DCE RPC standard because DCE for Windows NT maps OSF DCE RPC to Microsoft RPC.

Supported Transport Protocols

DCE for Windows NT, Version 2.2 is supported on Microsoft Windows NT Version 4.0 with Service Pack 3 (or later). Service Pack updates are automatically sent to Microsoft Developer Network (MSDN) Level 2 members. If you are not an MSDN Level 2 member, you can obtain the appropriate Service Pack by accessing the Microsoft FTP server directly from the Internet **(ftp ftp.microsoft.com)** or from a World-Wide web browser **(ftp://ftp.microsoft.com)**.

If you cannot obtain the Service Pack as described above, you can obtain it by calling Microsoft to order a CD or 3.5″ diskette version.

DCE for Windows NT provides RPC communications over the following transport protocols:

DCE String
Protocol Name

ncacn_ip_tcp
NCA Connection over Transmission Control Protocol/Internet Protocol (TCP/IP)

ncadg_ip_udp
Datagram-oriented TCP/IP.NCA Datagram over User Datagram Protocol/Internet Protocol (UDP/IP)

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.