Gradient® DCE for Tru64™ UNIX®

Reference Guide

Software Version 4.2

Notices

Gradient DCE for Tru64 UNIX Reference Guide - Software Version 4.2 - Revised November 2001

THIS DOCUMENT AND THE SOFTWARE DESCRIBED HEREIN ARE FURNISHED UNDER A SEPARATE LICENSE AGREEMENT, AND MAY BE USED AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION OF THE COPYRIGHT NOTICE BELOW. TITLE TO AND OWNERSHIP OF THE DOCUMENT AND SOFTWARE REMAIN WITH ENTEGRITY SOLUTIONS CORPORATION AND OR ITS LICENSOREES.

The information contained in this document is subject to change without notice.

ENTEGRITY SOLUTIONS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE SOFTWARE, DOCUMENTATION AND THIS MATERIAL, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Entegrity Solutions shall not be liable for errors contained herein, or for any direct or indirect, incidental, special or consequential damages in connection with the furnishing, performance, or use of this material.

Use, duplication or disclosure by the Government is subject to restrictions as set forth in Entegrity's standard commercial license agreement and is commercial computer software and documentation pursuant to Section 12.212 of the FAR and 227.7202 subparagraph (c) (1) (i) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Entegrity, Entegrity Solutions, and Gradient are registered trademarks of Entegrity Solutions Corporation. NetCrusader is a trademark of Entegrity Solutions Corporation.

Compaq, TruCluster,and AlphaServer are registered trademarks of Compaq Computer Corporation. Tru64 is a trademark of Compaq Computer Corporation. The the names of other Compaq products referenced herein are trademarks or service marks, or registered trademarks or service marks, of Compaq Computer Corporation.

Kerberos is a trademark of Massachusetts Institute of Technology. UNIX is a registered trademark of The Open Group. The Open Group is a trademark of The Open Group. DCE is copyrighted by The Open Group and other parties. Other products mentioned in the document are trademarks or registered trademarks of their respective holders.

Copyright © 1991–2001 The Open Group

Copyright © 2001 Entegrity Solutions Corporation & its subsidiaries.

All Rights Reserved.

Entegrity Solutions Corporation, 2077 Gateway Place, Suite 200, San Jose, CA 95110, USA

Contents

Chapter 4 XDS Object Management Reference Pages 85

Chapter 5 Command Reference Pages 121

Preface

Intended Audience

The audience for this guide includes the following:

- Experienced programmers who want to write client/server applications.
- Experienced programmers who want to port existing applications to DCE.
- Users who want to run distributed applications.

Overview of this Guide

The *Gradient® DCE for Tru64™ UNIX® Reference Guide* provides users of Gradient DCE with reference information necessary to use Gradient DCE on Tru64 UNIX Version 5.0a systems. This guide should be used with the documents listed under Related Documentation .

Gradient DCE for Tru64 UNIX v4.0 is a layered product on the Tru64 UNIX Version 5.0, 5.0*a*, and 5.1operating systems. It is a compatible upgrade of DCE for Tru64 UNIX Version 3.0.. It consists of a full DCE implementation as defined by The Open Group (TOG). This software includes these components:

- Remote Procedure Call (RPC)
- Cell Directory Service (CDS)
- Distributed Time Service (DTS)
- DCE Security
- DCE Distributed File Service (DFS, based on DCE Release 1.2.2)
- Lightweight Directory Access Protocol (LDAP)

NOTE: The products named Gradient DCE for Tru64 UNIX v3.1 (and higher), Digital DCE v3.1, and Compaq DCE v3.1 provide essentially the same features; however, only Gradient DCE for Tru64 UNIX functions on the Tru64 UNIX v5.0a operating system. Although other company names may be referenced within this document (Digital, Compaq, or Gradient Technologies), this DCE product is now produced and supported by Entegrity Solutions® Corporation.

Conventions

The following conventions are used in this guide:

UPPERCASE and lowercase	The operating system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function definitions must be typed exactly as shown.

bold	Boldface type in interactive examples indicates typed user input. In general text reference, bold indicates file names and commands.
italics	Italic type indicates variable values, placeholders, and function argument names.
`special type`	Indicates system output in interactive and code examples.
%	The default user prompt is your system name followed by a right angle bracket (>). In this manual, a percent sign (%) is used to represent this prompt.
#	A number sign (#) represents the superuser prompt.
Ctrl/*x*	This symbol indicates that you hold down the Ctrl key while pressing the key or mouse button that follows the slash.
<Return>	Refers to the key on your terminal or workstation that is labeled with Return or Enter.

Related Documentation

The following documents are available in HTML and Acrobat format on the Entegrity software CD:

- *Gradient DCE for Tru64 UNIX Installation and Configuration Guide* — Describes how to install DCE and configure and manage your DCE cell.
- *Gradient DCE for Tru64 UNIX Product Guide* — Provides supplemental documentation for Gradient DCE value-added features.
- *Gradient DCE for Tru64 UNIX Reference Guide* (this guide) — Provides supplemental reference information for Gradient DCE value-added features.
- *Gradient DCE for Tru64 UNIX Release Notes*— Lists new features, bug fixes, and known problems and restrictions.

The following OSF DCE Release 1.2.2 technical documentation is provided on the Entegrity software CD in PDF format:

- *Introduction to OSF DCE* — Contains a high-level overview of DCE technology including its architecture, components, and potential use.
- *OSF DCE Administration Guide - Introduction* — Describes the issues and conventions concerning DCE as a whole system and provides guidance for planning and configuring a DCE system.
- *OSF DCE Administration Guide - Core Components* — Provides specific instructions on how core components should be installed and configured.
- *OSF DCE Application Development Guide - Introduction and Style Guide* — Serves as a starting point for application developers to learn how to develop DCE applications.
- *OSF DCE Application Development Guide - Core Components* — Provides information on how to develop DCE applications using core DCE components such as RPC and security.

- *OSF DCE Application Development Guide - Directory Services* — Contains information for developers building applications that use DCE Directory Services.
- *OSF DCE Application Development Reference* — Provides reference information for DCE application programming interfaces.
- *OSF DCE Command Reference* — Describes commands available to system administrators.

Contacting Entegrity Solutions

Contact	Address	Phone/Fax/E-mail
DCE Product and Sales Information	Entegrity Solutions Corporation 410 Amherst Street, Suite 150 Nashua, NH 03063 USA	E-mail: DCESales@entegrity.com Web: **www.entegrity.com** Telephone and Fax: United States and Canada Tel: +1 (603) 882-1306 Tel (US Only): 1-800-525-4343 Fax: +1 (603) 882-6092
All Other Product and Sales Information Requests	Entegrity Solutions Corporation 2077 Gateway Place, Suite 200 San Jose, CA 95110 USA	E-mail: info@entegrity.com Web: **www.entegrity.com** Telephone and Fax: Tel: +1 (408) 487-8600 Tel (US Only): 1-866-487-8600 Fax: +1 (408) 487-8610
Technical Support	Entegrity Solutions Corporation Technical Support 2 Mount Royal Ave. Marlborough, MA 01752 USA	United States and Canada: Tel: +1 (508) 229-0239 Tel (US Only): 1 (888) 368-3555 Fax: +1 (508) 229-0338 E-mail: support@entegrity.com **http://support.entegrity.com**

Obtaining Technical Support

If you purchased your NetCrusader™ product directly from Entegrity Solutions Corporation or Gradient Technologies, Inc. you are entitled to 30 days of limited technical support beginning on the day the product is expected to arrive.

You may also purchase a support plan that entitles you to additional services. You *must* register prior to receiving this support. For details, refer to the customer support information package that accompanied your shipment or refer to the Technical Support area of **http://support.entegrity.com**. The web site also contains online forms for easy registration.

If you purchased NetCrusader from a reseller, please contact the reseller for information on obtaining technical support.

Obtaining Additional Technical Information

Contact	Address	Phone/Fax/Email
The Open Group™ Developer of DCE (Distributed Computing Architecture) software and standards.	The Open Group™ 29B Montvale Ave Woburn MA 01801 U. S. A.	Tel: +1 781-376-8200 Fax: +1 781-376-9358 **http://www.opengroup.org**

Obtaining Additional Documentation

All documentation for your NetCrusader product is provided in electronic format on the same CD on which the product ships. See the product CD for information on accessing this documentation.

Documentation for all of Entegrity's products is available at **http://support.entegrity.com**. Enter the Support Web area and click the Documentation link.

We are always trying to improve our documentation. If you notice any inaccuracies or cannot find information, please send email to **docs@entegrity.com**. We welcome any comments or suggestions.

CHAPTER 1

dcesetup Reference Page

This chapter contains the **dcesetup(8dce)** reference page.
Gradient DCE for Tru64 UNIX provides the **dcesetup** utility for use in configuring DCE services. The **dcesetup** utility replaces the **dce_config** program provided with OSF DCE Release 1.1.

Description

dcesetup - Configures and starts up DCE. The command invokes an enhanced configuration utility to configure and start DCE services. The configuration utility displays a menu that allows you to perform the following operations:

- Configure DCE services (**config**)
- Show DCE configuration and active daemons (**show**)
- Terminate all active DCE daemons (**stop**)
- Start all DCE daemons (**start**)
- Terminate and restart all DCE daemons (**restart**)
- Terminate all active DCE daemons and remove all temporary local DCE databases (**clean**)
- Terminate all active DCE daemons and remove all permanent local DCE databases (**clobber**)
- Run the Configuration Verification Program (**cvp**)

Permissions Required

You must have **root** authority to run the **dcesetup** command. See the *Gradient DCE for Tru64 UNIX Installation and Configuration Guide* and the *Gradient DCE for Tru64 UNIX Product Guide* for more information about **dcesetup**.

CHAPTER 2

idl Reference Page

This chapter contains a revised **idl(1rpc)** reference page which includes information about the following value-added functions:

- Support for Digital Fortran applications
- Support for C++ object oriented idl extensions.
- Support for Digital extensions provided in Digital products based on OSF DCE Release 1.0.

Name

idl - Invokes the Interface Definition Language (IDL) compiler

Synopsis

idl *filename* [*argument*] ...

Arguments

-client *file_type*

Determines which client files to generate. If you do not specify this argument, the compiler generates all client files. The file types are as follows:

none

Does not generate client files.

stub

Generates only a client stub file.

aux

Generates only a client auxiliary file. A client auxiliary file is generated only if the interface contains any out-of-line or self-pointing types.

all

Generates client stub and client auxiliary files. This is the default and is the same as not specifying the **-client** argument.

-server *file_type*

Determines which server files to generate. If you do not specify this argument, the compiler generates all server files. The file types are as follows:

none

Does not generate server files.

stub

Generates only a server stub file.

aux

Generates only a server auxiliary file. A server auxiliary file is generated only if the interface contains any out-of-line, self-pointing, or pipe types.

all

Generates server stub and server auxiliary files. This is the default and is the same as not specifying the **-server** argument.

-standard *standard_type*

Allows you to specify portable or extended features of the OSF DCE. This option is useful when you perform builds. The *standard_type* argument specifies what IDL features 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 language features available in OSF DCE Version 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 Digital extensions to its products based on OSF DCE Version 1.0.

extended

Allows all language features supported in the current version of the compiler. This is the default.

dce_v11

Synonymous with the **extended** argument.

The following example command line compiles the IDL interface **test.idl** and enables extended features of the OSF DCE:

```
%% idl test.idl -standard extended
```

-cstub *filename*

Specifies a pathname for the client stub file. When you give a filename, do not give a file extension; the **idl** compiler appends **.c** to the C source file and **.o** to the object file. If you do not use the **-cstub** argument, the **idl** compiler appends **_cstub.c** to the C source file and **_cstub.o** to the object file. If the **-lang cxx** option is used, the source file has the **.cxx** extension.

-sstub *filename*

Specifies a pathname for the server stub file. When you give a filename, do not give a file extension; the **idl** compiler appends **.c** to the C source file and **.o** to the object file. If you do not use the **-sstub** argument, the **idl** compiler appends **_sstub.c** to the C source file and **_sstub.o** to the object file. If the **-lang cxx** option is used, the source file has the **.cxx** extension.

-caux *filename*

Specifies a pathname for the client auxiliary file. When you give a filename, do not give a file extension; the **idl** compiler appends **.c** to the C source file and **.o** to the object file. If you do not use the **-caux** argument, the **idl** compiler appends **_caux.c** to the C source file and **_caux.o** to the object file. If the **-lang cxx** option is used, the source file has the **.cxx** extension.

-saux *filename*

Specifies a pathname for the server auxiliary file. When you give a filename, do not give a file extension; the **idl** compiler appends **.c** to the C source file and **.o** to the object file. If you do not use the **-caux** argument, the **idl** compiler appends **_saux.c** to the C source file and **_saux.o** to the object file. If the **-lang cxx** option is used, the source file has the **.cxx** extension.

-header *header_file*

Allows you to specify a name for the generated header file. By default the compiler takes the basename of the IDL file and appends the **.h** extension to it.

-out *directory*

Places the output files in the directory you specify. By default the compiler places the output files in the current working directory.

-I*directory* Specifies a directory name that contains imported interface definition files. You can specify more than one directory by specifying additional **-I***directory* arguments on the command line. The compiler searches the directories in the order you list them. If a file is present in more than one directory, the compiler takes the first occurrence of the file. The default behavior of the compiler is to first search the current directory, then all directories you specify, then the system IDL directory. The directory you specify is also passed to the C preprocessor and the C compiler.

-no_def_idir

Specifies that the compiler search only the current directory for imported files. When you use this with **-I***directory*, the compiler searches only the directories you list, not the current directory, and not the system IDL directory.

-no_mepv

Causes the compiler to not generate a manager entry point vector (EPV) in the server stub. Use this argument if the manager code and IDL file do not use the same operation names. If you specify this argument you must provide an EPV within the manager code that can be used when the interface is registered with the RPC server runtime.

The name of the type that you construct an EPV with is *if_name*_**v***major-version_minor-version*_**epv_t** where *if_name* is the interface name. It is not necessary to use this argument if the operation names in the manager code and IDL file are the same. In this case, the compiler generates a manager EPV in the server stub using the names of the operations in the IDL file. (For information on registering the server, see the **intro(3rpc)** and **rpc_server_register_if(3rpc)** reference pages. See the *OSF DCE Application Development Guide*.)

-cepv

Generates local routines in the client stub file (*filename*_**cstub.c**) and defines a Client Entry Point Vector (CEPV) of the name *if_name*_**v***major-version_minor-version*_**c_epv** where *if_name* is the interface name. The CEPV contains the addresses of the local routines. The client code must call the routines indirectly by using the addresses in the CEPV; otherwise, the stub routines in the client stub file must have the same names as the operations in the IDL file. (For information on registering the server, see the **intro(3rpc)** and **rpc_server_register_if(3rpc)** reference pages. See the *OSF DCE Application Development Guide*.)

-cpp_cmd *'c_preprocessor_command_line'*

Allows you to specify a C preprocessor other than the default. The compiler invokes the C preprocessor found in that command line. The output of the C preprocessor is an expanded version of the input file(s) containing replacement text for any preprocessor directives (for example, the **#include** preprocessor directive).

-cpp_opt *'command_options'*

Specifies additional options to be passed to the C preprocessor. You can add options to the command line used to invoke the C preprocessor independent of the **-cpp_cmd** argument. The IDL compiler concatenates the **-cpp_cmd, -cpp_opt, -D, -U, -I** arguments and the source filename into a command used to invoke the C preprocessor. The compiler repeats this process for each Attribute Configuration File (ACF) and IDL file.

-no_cpp

Does not invoke the C preprocessor. Note that the C preprocessor must be run on files that contain preprocessor directives (such as **#include**) in the interface definition.

-cc_cmd *'command_line'*

Invokes the C compiler and compiler options you specify in the *'command_line'* argument rather than the default C compiler and compiler options. When used with the **-lang cxx** option, the **-cc_cmd** option specifies the C++ compiler.

-cc_opt *'command_options'*

Specifies additional options to be passed to the C compiler. You can add options to the command line used to invoke the C compiler independent of the **-cc_cmd** argument. The IDL compiler concatenates the **-cc_cmd**, **-cc_opt**, **-I** arguments and the source filename into a command that invokes the C compiler. This procedure is done for each generated stub or auxiliary file. When used with the **-lang cxx** option, the **-cc_opt** option specifies the C++ compiler options.

-D*name*[=*definition*]

Defines a symbol name and an optional value to be passed to the C preprocessor. You can use this method of defining a symbol instead of using **#define** in the source code. You can use more than one **-D***name* argument on the command line. This argument has no effect if you use the **-no_cpp** argument.

-U*name*

Removes (undefines) any initial definition of a symbol name as defined by **-D***name*. You can use this method to remove a symbol name instead of using **#undef** in the source code. You can use more than one **-U***name* argument on the command line. This argument has no effect if you use the **-no_cpp** argument. If you define and undefine a name on the same command line, undefining takes precedence.

-space_opt

Generates code for the marshalling and unmarshalling of data that is optimized for space, rather than speed.

-syntax_only

Checks only the syntax of the IDL file, but does not generate any output files.

-keep *file_types*

Specifies which files to retain. To produce the object modules, the IDL compiler first creates C source modules, then invokes the target C compiler to produce object modules, and finally, deletes the C source modules. If you do not use **-keep**, only the object modules are saved.

The file types are as follows:

none

Does not save the C source or the object modules. Does not invoke the C compiler.

c_source

Saves only the C source modules. Does not invoke the C compiler.

object

Saves only the object modules.

all

Saves both the C source and the object modules.

-bug *n*, **-no_bug** *n*

Retains (**-bug**) or does not retain (**-no_bug**) a specified bug from earlier IDL compiler versions. (This in an NCS compatibility argument and is not supported in OSF DCE Version 1.1).

-stdin

Takes the standard output of a previous utility as the input to the **idl** command. For example:

$$ **cat my_filename.idl | idl -stdin**

-version

Displays the current version of the IDL compiler.

-v

Prints informational messages (verbose mode) on the screen while the compiler is running.

-no_warn

Suppresses compiler warning messages.

-confirm

Displays all the **idl** command arguments you chose, but does not compile the source IDL file. If you use this with the **-v** argument, informational messages about how the compiler behaves if you do not use **-confirm** are displayed but no corresponding actions are performed.

-template_client *filename*

Requests that the IDL compiler generate a C source file containing a template implementation of each routine that must appear in the client application to use the specified IDL interface. If you do not specify an extension for *filename*, the compiler assigns the file extension **.c**.

-template_manager *filename*

Requests that the IDL compiler generate a C source file containing a template implementation of each routine and operation that must appear in the manager module of the server side of an application to use the specified IDL interface. If you do not specify an extension for *filename*, the compiler assigns the file extension **.c**.

-trace *value*

Enables event logging.

You can specify one of the following values for the *value* argument:

all

Log all events.

none

Disable all previously specified trace options.

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.

-lang *{c, cxx, fortran}*

Allows you to select a programming language.

If you are generating stubs and include files for application code written in C++, you must specify cxx as the language of choice when you compile the application's IDL file. When appropriate, you can extend the class hierarchy and derive other classes from this one, to implement some or all interface operations. The C++ compiler gives a warning if any functions in the interface class have not been implemented. Avoid overwriting the *manager class header* file by using the **-no_cxxmgr** argument in conjunction with the **-lang cxx** argument.

If you are generating stubs and include files for application code written in Fortran, you must specify Fortran as the language of choice when you compile the application's IDL file.

If you do not specify **-lang fortran** or **-lang cxx**, the default value is the C programming language or **-lang c**.

-no_cxxmgr

Prevents the compiler from overwriting the *manager classheader* file. Use this argument in conjunction with the **-lang cxx** argument if you implement application-specific C++ code in the manager class header file.

Description

The **idl** command invokes the IDL compiler to convert an interface definition, written in IDL, into output files. The output files include a header file, server stub file, client stub file, and auxiliary files. The compiler constructs the names of the output files by keeping the basename of the interface definition source file but replacing the filename extension with the new extension (or suffix and extension) appropriate to the newly generated type of output file. For example, **math.idl** could produce **math_sstub.c** or **math_sstub.o** for the server stub.

The **idl** command accepts the following input:

- An interface definition filename.
- Arguments to indicate either special actions to be performed by the compiler, or special properties of the input or output files.

The IDL compiler searches through directories for any related ACF. For example, if you compile a file named **source.idl**, the compiler automatically searches for a file named **source.acf**. The compiler also searches for any imported IDL file (and its related ACF). The compiler searches for these files using the following order:

1. The current working directory. The compiler always searches this directory unless you specify the **-no_def_idir** and **-I***directory* arguments together.
2. Any imported directory. The compiler searches each directory you are specifying in the **-I***directory* argument.
3. The system IDL directory. The compiler automatically imports **nbase.idl**, which resides in the system IDL directory. The compiler always searches this directory unless you specify the **-no_def_idir** argument.
4. The directory specified in the source filename. If you explicitly specify a directory in the source IDL pathname, then that directory is searched for the corresponding ACF. For example,

   ```
   $$ idl /path/pathname/my_source.idl
   ```

   causes the IDL compiler to look for **/path/pathname/my_source.acf** if **my_source.acf** is not found in the directories in 1,2 and 3.

   Note that this directory is not searched for any imported IDL file or its corresponding ACF.

Restrictions

The following filenames are reserved by the IDL compiler. Naming an IDL file with one of these names may result in unexpected behavior

iovector.idl	lbase.idl	nbase.idl	ncastat.idl
ndrold.idl	rpc.idl	rpcbase.idl	rpcpvt.idl
rpcsts.idl	rpctypes.idl	twr.idl	uuid.idl

Cautions

The IDL compiler generates ANSI C code. It also supports C compilers that are not fully ANSI compliant although a warning message may occur during compilation of the stubs by the C compiler. A C compiler that is not fully ANSI compliant may generate the following warning messages:

```
warning: & before array or function: ignored
warning: enumeration type clash, operator =
```

Files

/lib/cpp	C preprocessor
dceshared/**bin/idl**	Compiler
dceshared/**include**	System IDL directory for imported files
dceshared/**include/dce/nbase.idl**	Predefined IDL types
dceshared/**nls/msg/***LANG*/**idl.cat**	Compiler error messages
dceshared/**share/include/***file.ext*	All **.idl** or **.h** files that are part of DCE RPC

Examples

1 Invoke the IDL compiler to compile the interface definition file **test.idl** and keep the generated C source modules. Only server files are generated. The server stub default filename is overridden by creating a file named **test_ss.c** for the server stub module.

```
$$ idl test.idl -keep c_source -client none -sstub test_ss.c
```

2 Invoke the IDL compiler to compile the interface definition file **test.idl**, but do not run the C preprocessor. The manager entry point vector is not defined in the generated server stub module. The IDL compiler searches the parent directory of the current directory for any IDL files that **test.idl** could import. The generated output files are located in the **output** subdirectory under the current directory.

```
$$ idl test.idl -no_cpp -no_mepv -I.. -out ./output
```

Related Information

Books: *OSF DCE Application Development Guide*.

CHAPTER 3

XDS Directory Services Reference Pages

This chapter provides reference pages for the X/Open Directory Services (XDS) API functions. Digital's XDS implementation includes two additional XDS functions that support asynchronous operations. The functions are as follows:

- The **ds_abandon** function abandons the outstanding asynchronous function call.
- The **ds_receive_result** function retrieves completed results of an outstanding asynchronous operation.

ds_intro(3xds)

Name

ds_intro - This reference page introduces the X/OPEN Directory Services (XDS) functions.

Syntax

```
#include <xom.h>
#include <xds.h>
```

Description

This reference page lists the XDS interface functions supported in the Digital X.500 product. XDS provides a C language binding

Function	Description
ds_abandon	Abandons an outstanding asynchronous operation.
ds_add_entry	Adds a leaf entry to the Directory Information Tree (DIT).
ds_bind	Opens a session with a directory user agent.
ds_compare	Compares a purported attribute value with the attribute value stored in the directory for a particular entry.
ds_initialize	Initializes the interface.
ds_list	Enumerates the immediate subordinates of a particular directory entry.

ds_modify_entry	Performs an atomic modification of a directory entry.
ds_modify_rdn	Changes the Relative Distinguished Name (RDN) of a leaf entry.
ds_read	Queries information on a directory entry by name
ds_receive_result	Retrieves the result of an asynchronously executed operation
ds_remove_entry	Removes a leaf entry from the DIT
ds_search	Finds entries of interest in a portion of the DIT
ds_shutdown	Shuts down the interface
ds_unbind	Unbinds from a directory session
ds_version	Negotiates features of the interface and service
dsX_trace_object	Displays an explanation of the content of an object

DCE Notes

The DEC X.500 Directory Service supports asynchronous operations, which the Distributed Computing Environment (DCE) XDS interface does not. Thus, the Abandon and Receive Result functions are included in the Digital product.

The differences between the X.500 Directory Service and the Cell Directory Service (CDS) are as follows:

- All functions operate on the X.500 name space.
- CDS does not support the Modify RDN or Search functions. The Service-Error *unwilling-to-perform* is returned if either function is attempted on CDS.
- CDS does not support the X.500 schema. Therefore, CDS does not have:
- The concept of an object class
- Mandatory attributes for a given object
- A set of attributes expressly permitted for a given object
- A predefined definition of single and multivalued attributes

  The absence of the schema means that the usual errors, which are returned by X.500 for breach of the rules, are not returned by CDS.
- The CDS naming Directory Information Tree (DIT) is modeled on a typical file system architecture, in which directories are used to store objects and can contain subdirectories. Leaf objects in the CDS DIT are similar to X.500 naming objects. However, subtree objects are called directories as in a file system directory. All new objects must be added to an existing directory. CDS directory objects cannot be added, removed, modified, or compared using the XDS programming interface.

- In CDS, the naming attribute of an object is not stored in the object. Consequently, in CDS the Read operation never returns this attribute, and the Compare operation applied to this attribute returns with the Attribute-Error *constraint-violation*.

See the notes in the relevant reference page for function-specific differences.

ds_abandon(3xds)

Name

ds_abandon - This function abandons an outstanding asynchronous operation.

Syntax

Status = ds_abandon*(Session, Invoke-ID)*

Argument	Data Type	Access
Session	OM_private_object	read
Invoke-ID	Integer	write
DS_status	Status	

C Binding

DS_status ds_abandon*(session, invoke_id)*

OM_private_object	session
OM_sint	invoke_id

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session in which the operation was submitted to the directory.

Invoke-ID

Identifies the operation that is to be abandoned. You can only abandon interrogatory operations (Compare, List, Read, and Search).

The value of Invoke-ID must be that which was returned by the function call that initiated the asynchronous directory operation that is now to be abandoned.

Description

This function abandons the outstanding asynchronous function call. The asynchronous function is no longer outstanding after the Abandon function returns, and the results of the asynchronous function will never be returned by the Receive-Result function.

DCE Notes

The DCE XDS interface does not support asynchronous operations.

Return Value

DS_SUCCESS	The operation completedsuccessfully.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Abandon-Failed
- Communications-Error
- Library-Error, with Problem attribute values of *bad-session,*or *miscellaneous*
- The result of the asynchronous operation will not be returned even if an Abandon-Failed error is returned.

Example

The following code extract shows an example call to the Abandon function.

```
OM_private_object  bound_session;
OM_sint            invoke_id;
{
    DS_status          status;
    status = ds_abandon(bound_session, invoke_id);
    if (status == DS_SUCCESS)
    {
      printf("ABANDON was successful\n");
    }
    else
    {
      printf("ABANDON failed\n");
    }
}
```

The abandon function abandons the results of the asynchronous operation identified by the Invoke-ID argument.

ds_add_entry(3xds)

Name

ds_add_entry - Adds an entry to the Directory Information Tree (DIT).

Syntax

Status = ds_add_entry*(Session, Context, Name, Entry, Invoke-ID)*

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read
Entry	OM_object	read
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_add_entry*(session, context, name, entry, invoke_id_return)*
session

OM_private_object	session
OM_private_object	context
OM_object	name
OM_object	entry
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The Context parameters to be used for this operation. The Size-Limit and Dont-Dereference-Aliases Context parameters do not apply to this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object containing the distinguished name of the entry to be added. The immediate superior of the new entry is determined by removing the last RDN component that belongs to the new entry. The immediate superior should exist in the same DSA; otherwise, the function may fail with an Update-Error, *affecting-multiple-DSAs*. It does not fail if an agreement exists between the DSAs that allows the entry to be added. Any aliases in the name will not be dereferenced.

Entry

The attribute information which, together with the RDN, constitutes the entry to be created. The information must be contained in an Attribute List OM object, or an OM object that is a subclass of Attribute-List. The object parameter should not contain the value of the RDN of the entry being created.

Invoke-ID

The Invoke-ID of an asynchronous directory operation. This is passed by reference.

Description

This function adds an entry to the Directory. The entry can be either an object entry or an alias entry. The Directory checks that the resulting entry conforms to the Directory schema.

DCE Notes

Ideally, the user does not know whether X.500 or CDS is actually handling the DCE naming operations. There are, however, some situations where naming results will differ depending on which service is handling the operation. (The **intro** reference page for XDS functions describes the general differences between operations on X.500 and CDS.)

Note the following issues for the Add Entry function:

- All CDS operations are synchronous. If a CDS operation is attempted and the Context parameter Asynchronous has been set true, a Library-Error, *not-supported*, is returned.
- When a CDS name is passed to XDS and DCE is not installed, a Library-Error, *not-supported*, is returned. This error is also returned when an X.500 name is passed to XDS and X.500 is not installed.
- Only leaf objects (that is, objects that are not CDS directory objects) can be added to CDS through the XDS interface.

- The DS_A_OBJECT_CLASS attribute of an object is single valued in CDS and multivalued in X.500. If the Entry argument contains a DS_A_OBJECT_CLASS attribute with a value of DS_O_ALIAS, a CDS alias (soft link) will be created. If the attribute value is DS_O_GROUP_OF_NAMES, a CDS Group object will be created. Any other value for DS_A_OBJECT_CLASS, or the absence of this attribute, will result in the creation of an ordinary CDS object.
- Only the DS_A_COMMON_NAME and DS_A_MEMBER attributes are valid for the DS_O_GROUP_OF_NAMES object in CDS.
- CDS supports only the following X.500 attribute syntaxes:

  OM_S_TELETEX_STRING
  OM_S_OBJECT_IDENTIFIER_STRING
  OM_S_OCTET_STRING
  OM_S_PRINTABLE_STRING
  OM_S_NUMERIC_STRING
  OM_S_BOOLEAN
  OM_S_INTEGER
  OM_S_UTC_TIME_STRING
  OM_S_ENCODING_STRING

If attributes of any other syntax are supplied to an Add Entry operation that references CDS, then it returns the Attribute-Error *constraint-violation*. Because CDS does not implement the X.500 schema rules, some CDS objects may not contain mandatory attributes like object class and so on.

Return Value

DS_SUCCESS	The entry was added, if the operation was invoked synchronously. The operation was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants are returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Attribute-Error, *constraint-violation*
- Communications-Error
- Library-Error, with Problem attribute values of *bad-argument*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*, *not-supported* or *too-many*-operations
- Name-Error, *no-such-object*

- Referral
- Security-Error
- Service-Error
- Update-Error

Examples

The following code extracts show an example call to the Add Entry function. The Add Entry function is used to create a new directory entry containing two attributes: common name and organization unit.

There are two examples. The first example shows how to perform an asynchronous Add Entry operation. The second example shows how to perform a synchronous Add Entry operation.

The Bound_Session argument contains the identity of a session returned from an earlier call to the Bind function. This object identifies the session through which the request should be issued. The Name argument is assumed to have been previously defined. Examples of how to define a Name argument, including an example of a CDS Name argument, are shown in the Read function.

Example 1

```
OM_private_object bound_session, context;
OM_workspace      workspace;
OM_return_code    om_status = OM_SUCCESS;
OM_descriptor     ATLST_entry[4],/* contents of entry */
  ATTR_sn_Black[4],
  ATTR_oc_OrgPerson[5],
          Context[3];/* For the context */
/* Define the first X.500 Object Class attribute */
OMX_CLASS_DESC(     ATTR_oc_OrgPerson[0], DS_C_ATTRIBUTE);
OMX_ATTR_TYPE_DESC( ATTR_oc_OrgPerson[1], DS_ATTRIBUTE_TYPE,
                                          DS_A_OBJECT_CLASS);
OMX_STRING_DESC(   ATTR_oc_OrgPerson[2],  OM_S_OBJECT_IDENTIFIER_STRING,
                                          DS_ATTRIBUTE_VALUES,
                                          DS_O_PERSON.elements,
                                          DS_O_PERSON.length);
OMX_STRING_DESC(   ATTR_oc_OrgPerson[3],  OM_S_OBJECT_IDENTIFIER_STRING,
                                          DS_ATTRIBUTE_VALUES,
                                          DS_O_ORG_PERSON.elements,
                                          DS_O_ORG_PERSON.length);
OMX_OM_NULL_DESC(   ATTR_oc_OrgPerson[4]);
/* Define the X.500 Surname attribute */
OMX_CLASS_DESC(     ATTR_sn_Black[0], DS_C_ATTRIBUTE);
OMX_ATTR_TYPE_DESC( ATTR_sn_Black[1], DS_ATTRIBUTE_TYPE,
                                      DS_A_SURNAME);
OMX_ZSTRING_DESC(   ATTR_sn_Black[2], OM_S_PRINTABLE_STRING,
                                      DS_ATTRIBUTE_VALUES,
                                      "Black");
OMX_OM_NULL_DESC(   ATTR_sn_Black[3]);
/* Define the Attribute List */
OMX_CLASS_DESC(     ATLST_entry[0],   DS_C_ATTRIBUTE_LIST);
```

```
OMX_OBJECT_DESC(    ATLST_entry[1],   DS_ATTRIBUTES, ATTR_sn_Black);
OMX_OBJECT_DESC(    ATLST_entry[2],   DS_ATTRIBUTES, ATTR_oc_OrgPerson);
OMX_OM_NULL_DESC(   ATLST_entry[3]);
/* now create the context object and set the Asynchronous flag to */
/* true to indicate that the operation should be asynchronous.    */
om_status = om_create(DS_C_CONTEXT, OM_TRUE, workspace, &context);
OMX_CLASS_DESC(  Context[0],   DS_C_CONTEXT);
OMX_BOOLEAN_DESC(Context[1],   DS_ASYNCHRONOUS,OM_TRUE);
OMX_OM_NULL_DESC(Context[2]);
/* Now place the contents of the public object cpub_context into */
/* the private object context                                    */
om_status = om_put(context, OM_REPLACE_ALL, Context, 0, 0, 0);
{
     DS_status         status;
     OM_sint           invoke_id;
     OM_uint           completion_flag;
     DS_status         operation_status;
     OM_return_code    om_status;
     OM_private_object entry, add_entry_result;
     /* create the OM private object: entry */
     om_status = om_create(DS_C_ATTRIBUTE_LIST, OM_FALSE, workspace,
                           &entry);
     /* Copy the attribute list from the cpub_attr_list public */
     /* object into the entry private object                   */
     om_status = om_put(entry, OM_REPLACE_ALL, ATLST_entry, 0, 0, 0);
     /* Call the Add Entry function using entry as a parameter */
     status = ds_add_entry(bound_session, context, name, entry,
                           &invoke_id);


     if (status == DS_SUCCESS)
     {
        printf("ADD ENTRY request was successful\n");
     }
     else
     {
        printf("ADD ENTRY request failed\n");
     }


     /* now wait for the response... */
     completion_flag = DS_OUTSTANDING_OPERATIONS;
     /* loop around calls to receive_result() until we get one back */
     while ( (status == DS_SUCCESS)
              && ( completion_flag == DS_OUTSTANDING_OPERATIONS) )
     {
        status = ds_receive_result(bound_session, &completion_flag,
                                   &operation_status,
                                   &add_entry_result, &invoke_id);
        if (status == DS_SUCCESS)
        {
            switch (completion_flag)
            {
                case DS_COMPLETED_OPERATION:
                /* we have a completed operation */
                if(operation_status == DS_SUCCESS)
                {
                    printf("ADD ENTRY was successful\n");
                }
```

```
                    break;
                    case DS_OUTSTANDING_OPERATIONS:
                    printf("There are outstanding operations\n");
                    break;
                    case DS_NO_OUTSTANDING_OPERATION:
                    printf("There are NO outstanding operations\n");

            break;
          }
        }
      }
}
```

The above example shows:

- How to define a private object containing context parameters.
- How to define a public object (cpub_attr_list) containing the attributes to be added to the new directory entry.
- How to use the OM Create function to create a private object (entry) and how to use the OM Put function to copy the entry's attributes from the public object (cpub_attr_list) into the newly created private object (entry).
- How to use the Receive Result function to obtain the result of the Add Entry function.

The OM Create and the OM Put functions are assumed to succeed.

Example 2

```
OM_private_object bound_session, context, name;
{
    DS_status         status;
    OM_private_object entry;
    status = ds_add_entry(bound_session, DS_DEFAULT_CONTEXT, name,
                          entry, NULL);
    if (status == DS_SUCCESS)
    {
        printf("ADD ENTRY was successful\n");
    }
    else
    {
        printf("ADD ENTRY failed\n");
    }
}
```

The above example shows how to perform a synchronous Add Entry operation. Note that the Invoke_id argument is not needed and therefore set to NULL. The example assumes that all other arguments have been defined as shown in Example 1.

ds_bind(3xds)

Name

ds_bind - Opens a session with the directory service.

Syntax

Status = ds_bind*(Session, Workspace, Bound-Session)*

Argument	Data Type	Access
Session	OM_object	read
Workspace	OM_workspace	read
Bound-Session	OM_private_object	write
Status	DS_status	

C Binding

DS_status ds_bind*(session, workspace, bound_session_return)*

OM_object	session
OM_workspace	workspace
OM_private_object	bound_session_return

Arguments

Session

A Session OM object specifying the address of the DSA to bind to, and other information. You can submit either an OM public object or an OM private object as this argument. You can also use the constant Default-Session as the value of this argument, causing a new session to be created with default values for all its OM attributes. The Bind operation uses information from the DUA defaults file when the constant Default-Session is used.

Workspace

Specifies the workspace (obtained from a call to the Initialize function) which is to be associated with the session. All function results from directory operations using this session will be returned as private objects in this workspace. If the Session argument is a private object, it must be a private object in this workspace.

Bound-Session

A Session OM private object identifying a directory session. This session may be used as an argument to other functions, for example the Read function. If the value of Session was Default-Session or a public object, then Bound-Session is a new private object. Otherwise, when the Session is a private object, then Bound-Session is that private object. The function supplies default values for any of the OM attributes that were not present in the session instance supplied as an argument. It also sets the value of the File-Descriptor OM Attribute. The initial value of this attribute is No-Valid-File-Descriptor. On an OpenVMS system, a file descriptor is not returned and the value of this attribute does not change. Note also that if the application binds only to a CDS directory, the value of the File-Descriptor OM attribute does not change.

Description

This function opens a session with the directory service and returns a session object for use in subsequent function calls. This function must be called before any other directory functions.

DCE Notes

Ideally, the user does not know whether X.500 or CDS is actually handling the DCE naming operations. There are, however, some situations where naming results will differ depending on which service is handling the operation. (The **intro** reference page for XDS functions describes the general differences between operations on X.500 and CDS.)

Note that to use CDS when X.500 is not active, the Bind function must be called with the value of the *session* parameter to set to DS_DEFAULT_SESSION. In this case, the Bind function will return DS_SUCCESS, but the returned Bound Session object may be used only for directory operations on the CDS namespace. If an operation is attempted against X.500 with this Bound Session, the directory routine will return the Library-Error, *not-supported*.

If your application was built and runs on a system where CDS is installed but X.500 is not installed, the Bind function will only attempt to bind to the CDS directory. If your application was built and runs on a system where X.500 is installed but CDS is not installed, the Bind function will only attempt to bind to the X.500 directory, and will return an error if it fails. If both CDS and X.500 are installed on the system and your application was built and runs against the XDS shareable library files (or on a DEC OSF/1 system was linked against the archive libraries libdxd.a and libdxdcds.a), then the Bind function will attempt to bind to both directories.

Note that in normal operation, no error message is returned if the Bind function fails to connect to an X.500 directory, but an error will be returned when your application attempts an X.500 operation. If you require error messages to be returned when the Bind function fails, your application must call the Version function and negotiate the Digital extension feature DSX-RET-X500-BIND-ERR-FTR.

Return Value

DS_SUCCESS	The operation completed successfully.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- System-Error
- Library-Error, with Problem attribute values of *bad-session*, *miscellaneous*, *not-supported* or *too-many-sessions*
- Security-Error
- Service-Error
- Communications-Error

Example

The following code extract shows an example call to the Bind function. It establishes a session with the directory service.

```
OM_private_object  bound_session;
OM_workspace       workspace;
{
    DS_status         status;


    status = ds_bind(DS_DEFAULT_SESSION, workspace, &bound_session);
    if (status == DS_SUCCESS)
    {
      printf("BIND was successful\n");
    }
    else
    {
      printf("BIND failed\n");
    }


}
```

The Bind function associates a workspace, obtained from a call to the Initialize function, with the directory service session returned in the Bound_Session argument. The function uses the default session constant DS_DEFAULT_SESSION as the Session argument.

ds_compare(3xds)

Name

ds_compare - Compares an attribute value with the attribute value stored in the Directory for a particular entry.

Syntax

Status = ds_compare*(Session, Context, Name, AVA, Result, Invoke-ID)*

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read
AVA	OM_object	read
Result	OM_private_object	write
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_compare*(session, context, name, ava, result_return,* invoke_id_return)

OM_private_object	session
OM_private_object	context
OM_object	name
OM_object	ava
OM_private_object	result_return
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The Context parameters to be used for this operation. The Size-Limit Context parameter does not apply to this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object containing the name of the target entry. Any aliases in the name will be dereferenced unless prohibited by the Context parameter Dont-Dereference-Aliases.

AVA

An AVA OM object containing the attribute-value-assertion that specifies the attribute type and value to be compared with that in the entry.

Result

A Compare-Result OM private object containing flags indicating whether the values matched and whether the comparison was made against the original entry. It also contains the Distinguished Name of the target object if an alias was dereferenced.

Invoke-ID

The Invoke-ID of an asynchronous directory operation. This is only valid if the Asynchronous OM attribute in the Context parameter is set to True.

Description

This function checks that the value supplied in the given AVA is the same as the value or values of the same attribute type in the named entry. The operation fails and an error is returned if the target object is not found or if the target entry does not have the required attribute type.

If this function is called asynchronously, then the result can be abandoned by calling the Abandon function.

DCE Notes

Ideally, the user does not know whether X.500 or CDS is actually handling the DCE naming operations. There are, however, some situations where naming results will differ depending on which service is handling the operation. (The **intro** reference page for XDS functions describes the general differences betweenoperations on X.500 and CDS.) Note the following issues for the Compare function:

- All CDS operations are synchronous. If a CDS operation is attempted and the Context parameter Asynchronous has been set true, a Library-Error, *not-supported*, is returned.
- When a CDS name is passed to XDS and DCE is not installed, a Library-Error, *not-supported*, is returned. This error is also returned when an X.500 name is passed to XDS and X.500 is not installed.
- In CDS, the naming attribute of an object is not stored in the attribute list of an object. Thus in CDS a Compare operation of the purported naming attribute value with the naming attribute value of the directory object always fails to match.
- CDS supports only the following X.500 attribute syntaxes:

OM_S_TELETEX_STRING
OM_S_OBJECT_IDENTIFIER_STRING
OM_S_OCTET_STRING
OM_S_PRINTABLE_STRING
OM_S_NUMERIC_STRING
OM_S_BOOLEAN
OM_S_INTEGER
OM_S_UTC_TIME_STRING
OM_S_ENCODING_STRING

If attributes of any other syntax are supplied to a Compare operation that references CDS, then it returns the Attribute-Error *constraint-violation*.

- In CDS, the name parameter supplied to the Compare function must ultimately resolve to the name of a leaf (that is, a CDS Object) entry; otherwise, the Name-Error *no-such-object* is returned. The function never interprets the name parameter as the name of a CDS Directory entry.

Return Value

DS_SUCCESS	The comparison was completed, if the operation was invoked synchronously. The operation was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with Problem attribute values of *bad-argument*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*, *not-supported* or *too-many*-operations
- Attribute-Error, *constraint-violation*
- Name-Error, *no-such-object*
- Referral
- Security-Error
- Service-Error
- Communications-Error

Examples

The following code extracts show an example call to the Compare function. The Compare function is used to compare the common name attribute with the name attribute contained within the directory entry identified by the Name argument.

There are two examples. The first example shows how to perform an asynchronous Compare operation. The second example shows how to perform a synchronous Compare operation.

The Bound_Session argument contains the identity of a session returned from an earlier call to the Bind function. This object identifies the session through which the request should be issued. The Name argument is assumed to have been previously defined. Examples of how to define a Name argument, including an example of a CDS Name argument, are shown in the Read function.

Example 1

```
OM_private_object  ava;
OM_workspace       workspace;
OM_descriptor      cpub_ava[4];
DS_status          status;
OM_sint            invoke_id;
OM_uint            completion_flag;
DS_status          operation_status;
OM_return_code     om_status;
OM_private_object  name, compare_result;
OM_return_code     om_status = OM_SUCCESS;
OMX_CLASS_DESC(     cpub_ava[0], DS_C_AVA);
OMX_ATTR_TYPE_DESC( cpub_ava[1], DS_ATTRIBUTE_TYPE,
                                 DS_A_COMMON_NAME);
OMX_ZSTRING_DESC(   cpub_ava[2], OM_S_PRINTABLE_STRING,
                                 DS_ATTRIBUTE_VALUES,
                                 "Albert Einstein");
OMX_OM_NULL_DESC(   cpub_ava[3]);
/* create the OM private object: ava */
om_status = om_create(DS_C_AVA, OM_FALSE, workspace, &ava);
/* Copy the attribute list from the cpub_ava public object */
/* into the ava private object                              */

om_status = om_put(ava, OM_REPLACE_ALL, cpub_ava, 0,0,0);
/* call the ds_compare function using ava as a parameter */
status = ds_compare(bound_session, context, name,
                    ava, &compare_result, &invoke_id);

if (status == DS_SUCCESS)
{
   printf("COMPARE request was successful\n");
}
else
{
   printf("COMPARE request failed\n");
}
/* now wait for the response... */
completion_flag = DS_OUTSTANDING_OPERATIONS;
/* loop around calls to receive_result() until we get one back */
while ( (status == DS_SUCCESS)
         && ( completion_flag == DS_OUTSTANDING_OPERATIONS) )
{
   status = ds_receive_result(bound_session, &completion_flag,
                              &operation_status, &compare_result,
```

```
                                &invoke_id);
        if (status == DS_SUCCESS)
        {
            switch (completion_flag)
            {
                case DS_COMPLETED_OPERATION:
                /* we have a completed operation     */
                /* now see what we have got back ... */
                if(operation_status == DS_SUCCESS)
                {
                    printf("COMPARE result received\n");
                    /* use OM to examine compare_result object */
                    ...
                }
                else
                {
                    printf("COMPARE request failed\n");
                }
                break;


                case DS_OUTSTANDING_OPERATIONS:
                ...
                break;
                case DS_NO_OUTSTANDING_OPERATION:
                ...
                break;
            }
        }
    }
```

The above example shows:

- How to define an attribute value assertion and use that in the Compare function.
- How to define an AVA OM public object (cpub_ava) containing the attribute value assertion.
- How to use the OM Create function to create an AVA OM private object (ava) and how to use the OM Put function to copy the attribute value assertion from the public object (cpub_ava) into the newly created private object (ava).
- How to use the Receive Result function to obtain the result of the Compare function.

  The OM Create and the OM Put functions are assumed to succeed.

Example 2

```
OM_private_object bound_session, name, context;
{
     DS_status           status;
     OM_private_object   ava;
     status = ds_compare(bound_session, DS_DEFAULT_CONTEXT,
                         name, ava, &compare_result, NULL);
```

```
    if (status == DS_SUCCESS)
    {
       printf("COMPARE request was successful\n");
       /* examine compare result object to see if */
       /* comparison was TRUE or FALSE */
    }
    else
    {
       printf("COMPARE request failed\n");
    }
}
```

The above example shows how to perform a synchronous Compare operation. Note that the Invoke-ID argument is not needed and therefore set to NULL. The example assumes that all other arguments have been defined as shown in Example 1.

ds_initialize(3xds)

Name

ds_initialize - Initializes the interface.

Syntax

Workspace = ds_initialize*(void)*

Argument	Data Type
Workspace	OM_workspace

C Binding

OM_workspace ds_initialize*(void)*

Description

This function performs any necessary initialization of the X.500 API including the creation of a workspace. You must call this function before you call any other X.500 API functions. It may be called multiple times, in which case each call returns a workspace that is distinct from other workspaces created by the Initialize function but not yet deleted by the Shutdown function.

Return Value

Workspace

Upon successful completion this function returns a pointer to a workspace in which OM objects can be created and manipulated. Only objects created in this workspace can be used as arguments to the other directory interface functions. This function returns NULL if it fails.

Errors

This function does not return any errors.

Example

The following code extract shows an example of a call to the Initialize function. The Initialize function is used to initialize the X.500 API and create a workspace which can then be used by other functions.

```
OM_workspace workspace;
{
    if ((workspace = ds_initialize()) != NULL)
    {
      printf("INITIALIZE was successful\n");
    }
    else
    {
      printf("INITIALIZE failed\n");
    }
}
```

The Initialize function establishes the workspace that you can then use to communicate with the directory, for the remainder of the session.

ds_list(3xds)

Name

ds_list - Lists all the immediate subordinate entries of a directory entry.

Syntax

Status = ds_list*(Session, Context, Name, Result, Invoke-ID)*

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read
Result	OM_private_object	write
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_list*(session, context, name, result_return, invoke_id_return)*

OM_private_object	session
OM_private_object	context
OM_object	name
OM_private_object	result_return
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The directory context to be used for this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object specifying the name of the object entry whose immediate subordinates are to be listed. Any aliases in the name will be dereferenced unless prohibited by the Context parameter Dont-Dereference-Aliases.

Result

A List-Result OM private object, passed by reference, containing some information about the target object's immediate subordinates. It also contains the distinguished name of the target object if an alias was dereferenced to find it. Aliases in the subordinate names are identified, but not dereferenced. Additionally, there may be a partial outcome qualifier which indicates that the result is incomplete. It also explains the reason why, for example, the time limit expired, and contains information that may be helpful when attempting to complete the operation.

Invoke-ID

The Invoke-ID of an asynchronous directory operation.

Description

This function is used to obtain a list of all the immediate subordinates of a named entry. It is possible that the list will be incomplete in some circumstances.

If this function is called asynchronously, then the result can be abandoned by calling the Abandon function.

DCE Notes

Ideally, the user does not know whether X.500 or CDS is actually handling the DCE naming operations. There are, however, some situations where naming results will differ depending on which service is handling the operation. (The **intro** reference page for XDS functions describes the general differences between operations on X.500 and CDS.)

Note the following issues for the List function:

- All CDS operations are synchronous. If a CDS operation is attempted and the Context parameter Asynchronous has been set true, a Library-Error, *not-supported*, is returned.
- When a CDS name is passed to XDS and DCE is not installed, a Library-Error, *not-supported*, is returned. This error is also returned when an X.500 name is passed to XDS and X.500 is not installed.

Return Value

DS_SUCCESS	The target object was located regardless of whether it has any subordinates, if the operation was invoked synchronously. The operation was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants are returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with Problem attribute values of *bad-argument*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*, *not-supported* or *too-many-* operations
- Name-Error
- Referral
- Security-Error
- Service-Error
- Communications-Error

Examples

The following code extracts show an example call to the List function. The List function is used to list the subordinates of the directory entry identified in the Name argument.

There are two examples. The first example shows how to perform an asynchronous List operation. The second example shows how to perform a synchronous List operation.

The Bound_Session argument contains the identity of a session, established using the Bind function, through which the request should be issued. The Name argument is assumed to have been previously defined. Examples of how to define a Name argument, including an example of a CDS Name argument, are shown in the Read function.

Example 1

```
OM_private_object bound_session, context, name;
OM_workspace      workspace;
{
    DS_status         status;
    OM_private_object list_result;
    OM_sint           invoke_id;
    OM_uint           completion_flag;
    DS_status         operation_status;
    OM_return_code    om_status;
    OM_public_object  spub_result;
    OM_value_position desc_count;
    /* call ds_list to list the subordinates of the entry */
    /* identified in name                                 */
    status = ds_list(bound_session, context, name, &list_result,
                     &invoke_id);
    completion_flag = DS_OUTSTANDING_OPERATIONS;
    /* loop around calls to receive_result() until we get one back */
    while ((status == DS_SUCCESS) &&
           (completion_flag == DS_OUTSTANDING_OPERATIONS))
    {
        status = ds_receive_result(bound_session, &completion_flag,
                                   &operation_status, &list_result,
                                   &invoke_id);


        if (status == DS_SUCCESS)
        {
            switch (completion_flag)
            {
            case DS_COMPLETED_OPERATION:
            /* we have a completed operation     */
            /* now see what we have got back ... */
            if (operation_status == DS_SUCCESS)
            {


                om_status = om_get(list_result, OM_NO_EXCLUSIONS,
                                   0, 0, 0, OM_ALL_VALUES,
                                   &spub_result, &desc_count);
                if (om_status == OM_SUCCESS)
                {
                   /* if desc_count is not zero, the results are now */
                   /* available in the public object spub_result     */
                }
                else
                {
                   /* error getting results */
                }
            }
            else
```

```
            {...}
            break;
            case DS_COMPLETED_OPERATION:
            ...
            break;
            case DS_COMPLETED_OPERATION:
            ...
            break;
        }
    }

}
```

This example shows:

- A call to the List function.
- How to use the Receive Result function to obtain the result of the List function.
- How to use the OM Get function to copy the attributes of the List-Result OM private object into the equivalent List-Result OM public object (Spub_Result) for examination.

The OM Get function is assumed to succeed.

Example 2

```
OM_private_object bound_session, context, name;
{
    DS_status         status;
    OM_private_object list_result;
    OM_public_object  spub_result;
    OM_value_position desc_count;
    status = ds_list(bound_session, DS_DEFAULT_CONTEXT, name,
                     &list_result, NULL);
    if (status == DS_SUCCESS)
    {
        /* LIST was successful */
        /* now see what we have got back ... */
        om_status = om_get(list_result, OM_NO_EXCLUSIONS,
                           0, 0, 0, OM_ALL_VALUES,
                           &spub_result, &desc_count);
        if (om_status == OM_SUCCESS)
        {
            /* if desc_count!=0, results now available as a public */
            /* object                                              */
        }
        else
        {
            /* error getting results */
        }
    }
    else
    {...}
}
```

This example shows how to perform a synchronous List operation. Note that the Invoke-ID argument is not needed and therefore set to NULL. The example assumes that all other arguments have been defined as shown in Example 1.

ds_modify_entry(3xds)

Name

ds_modify_entry - Performs an modification on an entry.

Syntax

Status = ds_modify_entry*(Session, Context, Name, Changes, Invoke-ID)*

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read
Changes	OM_object	read
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_modify_entry*(session, context, name, changes, invoke_id_return)*

OM_private_object	session
OM_private_object	context
OM_object	name
OM_object	changes
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The Context parameters to be used for this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object containing the name of the target entry. Any aliases in the name will be dereferenced if the DSA attribute Dereference Alias on Modify is set and the Dont Deference Aliases service control is not set.

Changes

An Entry-Modification-List OM object specifying a sequence of modifications to the named entry.

Invoke-ID

The Invoke-ID of an asynchronous directory operation.

Description

This function is used to make a series of one or more of the following changes to a single entry:

- Add a new attribute (add-attribute)
- Remove an attribute (remove-attribute)
- Add attribute values (add-values)
- Remove attribute values (remove-values)

You can replace values by a combination of adding values and removing values in a single operation. You can only change the RDN of an entry by using the Modify-RDN function.

The result of the operation is as if each modification is made in the order specified in the Changes argument. If any of the individual modifications fail, then an Attribute-Error is reported and the entry is left as it was before the whole operation. The operation is atomic, either all the changes are made or none are. The Directory Service checks that the resulting entry conforms to the schema.

DCE Notes

Ideally, the user does not know whether X.500 or CDS is actually handling the DCE naming operations. There are, however, some situations where naming results will differ depending on which service is handling the operation. (The **intro** reference page for XDS functions describes the general differences between operations on X.500 and CDS.)

Note the following issues for the Modify Entry function:

- All CDS operations are synchronous. If a CDS operation is attempted and the Context parameter Asynchronous has been set true, a Library-Error, *not-supported*, is returned.
- When a CDS name is passed to XDS and DCE is not installed, a Library-Error, *not-supported*, is returned. This error is also returned when an X.500 name is passed to XDS and X.500 is not installed.

- Naming schema rules do not apply in CDS. At the XDS API, all CDS attributes are treated as multivalued. Adding an attribute that already exists on the CDS entry causes an additional value to be added to that attribute's set of values. Thus the following Attribute-Errors are never returned by CDS:
  - *no-such-attribute-or-value*
  - *attribute-or-value-already-exists*

  Naming operations that would normally return these errors succeed in CDS. In particular, the addition of an attribute that already exists does not return with an error. Instead, the values of the attribute to be added are combined with the values of the existing attribute.
- CDS supports only the following X.500 attribute syntaxes:
  - OM_S_TELETEX_STRING
  - OM_S_OBJECT_IDENTIFIER_STRING
  - OM_S_OCTET_STRING
  - OM_S_PRINTABLE_STRING
  - OM_S_NUMERIC_STRING
  - OM_S_BOOLEAN
  - OM_S_INTEGER
  - OM_S_UTC_TIME_STRING
  - OM_S_ENCODING_STRING

  If attributes of any other syntax are supplied to a Modify Entry operation that references CDS, then it returns the Attribute-Error *constraint-violation*.
- In CDS, the name parameter supplied to the Modify Entry function must ultimately resolve to the name of a leaf (that is, a CDS Object) entry; otherwise the Name-Error *no-such-object* is returned. The function never interprets the name parameter as the name of a CDS Directory entry.

Return Value

DS_SUCCESS	All the modifications were made to the entry, if the operation was invoked synchronously. The operation was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with Problem attribute values of *bad-argument*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*, *not-supported*,or *too-many*- operations
- Attribute-Error, *constraint-violation*
- Name-Error, *no-such-object*
- Referral
- Security-Error
- Service-Error
- Update-Error
- Communications-Error

An Attribute-Error is returned if you attempt any of the following:

- To use Add-Attribute to add an existing attribute
- To add a value to a nonexistent attribute type
- To use Remove-Attribute to remove a nonexistent attribute or nonexistent attribute value

An attempt to remove an attribute or attribute value which is part of the object's RDN or to modify the object class attribute results in an Update-Error.

Examples

The following code extracts show an example call to the Modify Entry function. Note that the standard schema does not contain an object class with the attributes used in the example. The Modify Entry function is used to modify the directory entry, identified in the Name argument, as follows:

- Add a new X.500 attribute Title with the value "Sales & Marketing Director"
- Add the value "Abacus Trading Corporation" to the X.500 attribute Organization Name
- Remove the X.500 attribute Organizational Unit Name
- Remove the value "US" from the Country Name X.500 attribute

There are two examples. The first example shows how to perform an asynchronous Modify Entry operation. The second example shows how to perform a synchronous Modify Entry operation.

The Bound_Session argument contains the identity of a session, established using the Bind function, through which the request should be issued. Two arguments are assumed to have been previously defined. These are the Name argument and the Context argument. Examples of how to define a Name argument, including an example of a CDS Name argument, are shown in the Read function. An example of how to define a Context argument is shown in the Add Entry function.

Example 1

```
OM_private_object bound_session, context, name;
/* define some public objects to contain the changes to be made to */
/* the directory entry                                              */
/* declare the descriptor lists (public objects) */
OM_descriptor cpub_mod_list[6];
OM_descriptor cpub_mod1[5];
OM_descriptor cpub_mod2[6];
OM_descriptor cpub_mod3[4];
OM_descriptor cpub_mod4[5];
/* define the first descriptor list */
OMX_CLASS_DESC(     cpub_mod1[0], DS_C_ENTRY_MOD);
OMX_ENUM_DESC(      cpub_mod1[1], DS_MOD_TYPE,
                                  DS_ADD_ATTRIBUTE);
OMX_ATTR_TYPE_DESC( cpub_mod1[2], DS_ATTRIBUTE_TYPE,
                                  DS_A_TITLE);
OMX_ZSTRING_DESC(   cpub_mod1[3], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "Sales & Marketing Director");
OMX_OM_NULL_DESC(   cpub_mod1[4]);
/* define the second descriptor list */
OMX_CLASS_DESC(     cpub_mod2[0], DS_C_ENTRY_MOD);
OMX_ENUM_DESC(      cpub_mod2[1], DS_MOD_TYPE,
                                  DS_ADD_VALUES);
OMX_ATTR_TYPE_DESC( cpub_mod2[2], DS_ATTRIBUTE_TYPE,
                                  DS_A_ORG_NAME);
OMX_ZSTRING_DESC(   cpub_mod2[3], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "Abacus Trading Corporation");
OMX_ZSTRING_DESC(   cpub_mod2[4], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "Abacus");
OMX_OM_NULL_DESC(   cpub_mod2[5]);
/* define the third descriptor list */
OMX_CLASS_DESC(     cpub_mod3[0], DS_C_ENTRY_MOD);
OMX_ENUM_DESC(      cpub_mod3[1], DS_MOD_TYPE,
                                  DS_REMOVE_ATTRIBUTE);
OMX_ATTR_TYPE_DESC( cpub_mod3[2], DS_ATTRIBUTE_TYPE,
                                  DS_A_ORG_UNIT_NAME);
OMX_OM_NULL_DESC(   cpub_mod3[3]);
/* define the fourth descriptor list */
OMX_CLASS_DESC(     cpub_mod4[0], DS_C_ENTRY_MOD);
OMX_ENUM_DESC(      cpub_mod4[1], DS_MOD_TYPE,
                                  DS_REMOVE_VALUES);
OMX_ATTR_TYPE_DESC( cpub_mod4[2], DS_ATTRIBUTE_TYPE,
                                  DS_A_COUNTRY_NAME);
OMX_ZSTRING_DESC(   cpub_mod4[3], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "US");
OMX_OM_NULL_DESC(   cpub_mod4[4]);


/* define the fifth descriptor list */
OMX_CLASS_DESC(     cpub_mod_list[0],       DS_C_ENTRY_MOD_LIST);
OMX_OBJECT_DESC(    cpub_mod_list[1],       DS_CHANGES, cpub_mod1);
OMX_OBJECT_DESC(    cpub_mod_list[2],       DS_CHANGES, cpub_mod2);
OMX_OBJECT_DESC(    cpub_mod_list[3],       DS_CHANGES, cpub_mod3);
```

```
OMX_OBJECT_DESC(    cpub_mod_list[4],       DS_CHANGES, cpub_mod4);
OMX_OM_NULL_DESC(   cpub_mod_list[5]);
{
    DS_status         status;
    OM_sint           invoke_id;
    OM_uint           completion_flag;
    DS_status         operation_status;
    OM_return_code    om_status;
    OM_private_object changes, modify_entry_result;
    /* create an OM Private object called changes*/
    om_status = om_create(DS_C_ENTRY_MOD_LIST, OM_FALSE, workspace,
                          &changes);
    /* now put the contents of the public object, cpub_mod_list, */
    /* in to the changes private object                          */
    om_status = om_put(changes, OM_REPLACE_ALL, cpub_mod_list,
                       0, 0, 0);
    /* Call the Modify Entry function using the changes object as */
    /* a parameter                                                 */
    status = ds_modify_entry(bound_session, context, name, changes,
                             &invoke_id);


    if (status == DS_SUCCESS)
    {
        printf("MODIFY ENTRY was successful\n");
    }
    else
    {
        printf("MODIFY ENTRY failed\n");
    }
    /* now wait for the response... */
    completion_flag = DS_OUTSTANDING_OPERATIONS;
    /* loop around calls to receive_result() until we get one back */
    while ((status == DS_SUCCESS) &&
           (completion_flag == DS_OUTSTANDING_OPERATIONS))
    {
        status = ds_receive_result(bound_session, &completion_flag,
                                   &operation_status,
                                   &modify_entry_result,
                                   &invoke_id);
        if (status == DS_SUCCESS)
        {
            switch (completion_flag)
            {
                case DS_COMPLETED_OPERATION:
                /* we have a completed operation */
                /* check operation_status */
                break;
                case DS_OUTSTANDING_OPERATIONS:
                ...
                break;
                case DS_NO_OUTSTANDING_OPERATION:
                ...
                break;
            }
        }
    }
}
```

The above example shows the following:

- How to define an Entry-Modification-List OM public object (cpub_mod_list) containing the modifications to be made.
- How to use the OM Create function to create an Entry-Modification-List OM private object (changes) and how to use the OM Put function to copy the modifications from the public object (cpub_mod_list) into the newly created private object (changes).

  Both the OM Create and the OM Put functions are assumed to succeed.
- How to obtain the result of the Modify Entry function using the Receive Result function.

Example 2

```
OM_private_object bound_session, context, name;
{
    DS_status         status;
    OM_private_object changes;
    status = ds_modify_entry(bound_session, DS_DEFAULT_CONTEXT,
                             name, changes, NULL);
    if (status == DS_SUCCESS)
    {
        printf("MODIFY_ENTRY was successful\n");
    }
    else
    {
        printf("MODIFY_ENTRY failed\n");
    }
}
```

The above example shows how to perform a synchronous Modify Entry operation. Note that the Invoke-ID argument is not needed so NULL is used.

This example assumes that the Changes argument has been defined as shown in Example 1.

ds_modify_rdn(3xds)

Name

ds_modify_rdn - Changes the Relative Distinguished Name (RDN) of an entry.

Syntax

Status = ds_modify_rdn*(Session, Context, Name, New-RDN, Delete-Old-RDN,* Invoke-ID)

Argument	Data Type	Access
Session	OM_private_object	read

Argument	Data Type	Access
Context	OM_private_object	read
Name	OM_object	read
New-RDN	OM_object	read
Delete-Old-RDN	OM_boolean	read
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_modify_rdn*(session, context, name, new_rdn, delete_old_rdn,* invoke_id_return)

OM_private_object	session
OM_private_object	context
OM_object	name
OM_object	new_rdn
OM_boolean	delete_old_rdn
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The directory context to be used for this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object containing the current name of the target entry. Any aliases in the name will be dereferenced if the DSA attribute Dereference Alias on Modify is set and the Dont Deference Aliases service control is not set.

New-RDN

A Relative-Name OM object specifying the new RDN. If an attribute value in the new RDN does not already exist in the entry, either as part of the old RDN or as a non-distinguished value, then the new value is added.

Delete-Old-RDN

When this takes the value *false* the old values will remain, but not as part of the RDN. When this takes the value *true*, all attribute values in the old RDN that are not also in the new RDN are deleted. If the operation removes the last value of an attribute, the attribute is deleted. This argument must be true when the value of a single-valued attribute is changed.

Invoke-ID

The Invoke-ID of an asynchronous directory operation.

Description

This function is used to change the RDN of a leaf entry. This can be either an object entry or an alias entry.

DCE Notes

CDS does not support the Modify RDN function; it returns with the Service-Error *unwilling-to-perform*.

Return Value

DS_SUCCESS	The RDN of the entry was changed, if the operation was invoked synchronously. The operation was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with Problem attribute values of *bad-argument*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*,or *too-many-operations*
- Attribute-Error
- Name-Error
- Referral
- Security-Error
- Service-Error
- Update-Error
- Communications-Error

The Update-Error *affects-multiple-DSAs* that is referred to in the argument descriptions need not be returned if there is local agreement between the DSAs to allow the entry to be modified.

Example

The following code extract shows an example call to the Modify RDN function:

```
OM_private_object  bound_session, context, name, new_rdn;
OM_sint            invoke_id;
OM_boolean         delete_old_rdn;
{
    DS_status          status;
    status = ds_modify_rdn(bound_session, DS_DEFAULT_CONTEXT, name,
                           new_rdn, delete_old_rdn, NULL);
    if (status == DS_SUCCESS)
    {
      printf("MODIFY RDN was successful\n");
    }
    else
    {
      printf("MODIFY RDN failed\n");
    }
    return status;}
```

ds_read(3xds)

Name

ds_read - Queries information in a particular entry.

Syntax

Status = ds_read*(Session, Context, Name, Selection, Result, Invoke-ID)*

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read
Selection	OM_object	read
Result	OM_object	write
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_read *(session, context, name, selection, result_return,* invoke_id_return)

OM_private_object session

OM_private_object	context
OM_object	name
OM_object	selection
OM_private_object	result_return
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The directory context to be used for this operation. The Size-Limit Context parameter does not apply to this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object containing the name of the target entry. Any aliases in the name will be dereferenced unless prohibited by the Context parameter Dont-Dereference-Aliases.

Selection

An Entry-Information-Selection OM object or a constant specifying what information from the named entry is requested. Information about no attributes, all attributes, or just a named set can be chosen. Attribute types are always returned, but the attribute values need not be. The following constants can be used:

- Select-No-Attributes, to verify the existence of an entry
- Select_All-Types, to return just the types of all attributes
- Select-All-Types-And-Values, to return the types and values of all attributes

Result

A Read-Result OM object, passed by reference, containing the distinguished name of the target object and a flag indicating whether the result came from the original entry or a copy. It also contains any requested attribute types and values. Attribute information is only returned if access rights are sufficient. No object is returned if the call does not complete successfully.

Invoke-ID

The Invoke-ID of an asynchronous directory operation.

Description

This function is used to extract information from an explicitly named entry. It can also be used to verify a distinguished name.

If this function is called asynchronously, then the result can be abandoned by calling the Abandon function.

DCE Notes

Ideally, the user does not know whether X.500 or CDS is actually handling the DCE naming operations. There are, however, some situations where naming results will differ depending on which service is handling the operation. (The **intro** reference page for XDS functions describes the general differences between operations on X.500 and CDS.)

Note the following issues for the Read function:

- All CDS operations are synchronous. If a CDS operation is attempted and the Context parameter Asynchronous has been set true, a Library-Error, *not-supported*, is returned.
- When a CDS name is passed to XDS and DCE is not installed, a Library-Error, *not-supported*, is returned. This error is also returned when an X.500 name is passed to XDS, and X.500 is not installed.
- Because CDS does not implement the X.500 schema rules, some CDS objects may not contain mandatory attributes such as object class and so on. In CDS, a read of an alias object fails if the DS_A_ALIASED_OBJECT_NAME does not exist. Instead, CDS returns with the Name-Error *no-such-object*.
- In CDS, the naming attribute of an object is not stored in the attribute list for the object. Thus in CDS, the Read function does not return this attribute in the attribute list for an object.

Return Value

DS_SUCCESS	The read was completed, if the operation was invoked synchronously. The operation was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with Problem attribute values of *bad-argument*, *bad-attribute*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*, *not-supported* or *too-many-operations*
- Attribute-Error
- Name-Error, *no-such-object*
- Referral
- Security-Error
- Service-Error
- Communications-Error
- An Attribute-Error, *no-such-attribute*, is reported if an explicit list of attributes is specified by the selection argument but none of them are present in the entry. This error is not reported if any of the selected attributes are present.
- A Security-Error, *insufficient-access-rights*, is reported where access rights prohibit the reading of all requested attribute values.

Examples

The following code extracts show an example call to the Read function. The Read function is used to read all the types and values from all attributes of the directory entry identified in the Name argument.

There are three examples. The first example shows how to perform an asynchronous Read operation. The second example shows how to perform a synchronous Read operation. The third example shows how to perform a synchronous Read operation with a CDS name.

The Bound_Session argument contains the identity of a session returned from an earlier call to the Bind function. This object identifies the session through which the request should be issued. The Context argument is assumed to have been previously defined. An example of how to define a Context argument is shown in the Add Entry function.

Example 1

```
{
     OM_workspace      workspace;
     OM_descriptor     cpub_dn[6];
     OM_descriptor     cpub_rdn1[3];
     OM_descriptor     cpub_rdn2[3];
     OM_descriptor     cpub_rdn3[3];
     OM_descriptor     cpub_rdn4[3];
     OM_descriptor     cpub_ava1[4];
     OM_descriptor     cpub_ava2[4];
     OM_descriptor     cpub_ava3[4];
     OM_descriptor     cpub_ava4[4];
     OM_value_position desc_count;
     DS_status         status;
     OM_sint           invoke_id;
```

```
OM_uint           completion_flag;
DS_status         operation_status;
OM_return_code    om_status;
OM_private_object name, read_result;
OM_public_object  spub_result;
OMX_CLASS_DESC(     cpub_ava1[0], DS_C_AVA);
OMX_ATTR_TYPE_DESC( cpub_ava1[1], DS_ATTRIBUTE_TYPE,
                                  DS_A_COMMON_NAME);
OMX_ZSTRING_DESC(   cpub_ava1[2], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "Albert Einstein");
OMX_OM_NULL_DESC(   cpub_ava1[3]);
OMX_CLASS_DESC(     cpub_ava2[0], DS_C_AVA);
OMX_ATTR_TYPE_DESC( cpub_ava2[1], DS_ATTRIBUTE_TYPE,
                                  DS_A_ORG_UNIT_NAME);
OMX_ZSTRING_DESC(   cpub_ava2[2], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "Research");
OMX_OM_NULL_DESC(   cpub_ava2[3]);
OMX_CLASS_DESC(     cpub_ava3[0], DS_C_AVA);
OMX_ATTR_TYPE_DESC( cpub_ava3[1], DS_ATTRIBUTE_TYPE,
                                  DS_A_ORG_NAME);
OMX_ZSTRING_DESC(   cpub_ava3[2], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "Digital Equipment Corporation");
OMX_OM_NULL_DESC(   cpub_ava3[3]);
OMX_CLASS_DESC(     cpub_ava4[0], DS_C_AVA);
OMX_ATTR_TYPE_DESC( cpub_ava4[1], DS_ATTRIBUTE_TYPE,
                                  DS_A_COUNTRY_NAME);
OMX_ZSTRING_DESC(   cpub_ava4[2], OM_S_PRINTABLE_STRING,
                                  DS_ATTRIBUTE_VALUES,
                                  "US");
OMX_OM_NULL_DESC(   cpub_ava4[3]);
OMX_CLASS_DESC(     cpub_rdn1[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn1[1],       DS_AVAS, cpub_ava1);
OMX_OM_NULL_DESC(   cpub_rdn1[2]);
OMX_CLASS_DESC(     cpub_rdn2[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn2[1],       DS_AVAS, cpub_ava2);
OMX_OM_NULL_DESC(   cpub_rdn2[2]);
OMX_CLASS_DESC(     cpub_rdn3[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn3[1],       DS_AVAS, cpub_ava3);
OMX_OM_NULL_DESC(   cpub_rdn3[2]);
OMX_CLASS_DESC(     cpub_rdn4[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn4[1],       DS_AVAS, cpub_ava4);
OMX_OM_NULL_DESC(   cpub_rdn4[2]);
OMX_CLASS_DESC(     cpub_dn[0],         DS_C_DS_DN);
OMX_OBJECT_DESC(    cpub_dn[1],         DS_RDNS, cpub_rdn4);
OMX_OBJECT_DESC(    cpub_dn[2],         DS_RDNS, cpub_rdn3);
OMX_OBJECT_DESC(    cpub_dn[3],         DS_RDNS, cpub_rdn2);
OMX_OBJECT_DESC(    cpub_dn[4],         DS_RDNS, cpub_rdn1);
OMX_OM_NULL_DESC(   cpub_dn[5]);
/* create the OM private object: name */
om_status = om_create(DS_C_DS_DN, OM_FALSE, workspace, &name);
/* Copy the attribute list from the cpub_dn public object into   */
/* the name private object                                       */
```

```
om_status = om_put(name, OM_REPLACE_ALL, cpub_dn, 0,0,0);
/* call the ds_read function using Name as a parameter and */
/* select only the information specified by rdn_type_list  */
status = ds_read(bound_session, context, name,
                DS_SELECT_ALL_TYPES_AND_VALUES, &read_result,
                &invoke_id);


if (status == DS_SUCCESS)
{
   printf("READ request was successful\n");
}
else
{
   printf("READ request failed\n");
}
/* now wait for the response... */
completion_flag = DS_OUTSTANDING_OPERATIONS;
/* loop around calls to receive_result() until we get one back */
while ( (status == DS_SUCCESS)
         && ( completion_flag == DS_OUTSTANDING_OPERATIONS) )
{
   status = ds_receive_result(bound_session, &completion_flag,
                              &operation_status, &read_result,
                              &invoke_id);
   if (status == DS_SUCCESS)
   {
       switch (completion_flag)
       {
           case DS_COMPLETED_OPERATION:
           /* we have a completed operation     */
           /* now see what we have got back ... */
           if (operation_status == DS_SUCCESS)
           {
              om_status = om_get(read_result, OM_NO_EXCLUSIONS,
                                 0, 0, 0, OM_ALL_VALUES,
                                 &spub_result, &desc_count);
              if (om_status == OM_SUCCESS)
              {
                 /* check desc_count != 0 */
                 /* results now available in public object */
                 /* spub_result                            */
              }
              else
              {
                 /* error getting results */
                 /* search_result not deleted */
              }
           }
           else
           {...}
           break;


           case DS_OUTSTANDING_OPERATIONS:
           ...
           break;
           case DS_NO_OUTSTANDING_OPERATION:
           ...
```

```
                break;
            }
        }
    }
}
```

The above example shows:

- How to define a private object containing a distinguished name.
- How to define a DS-DN OM public object (cpub_dn) containing the entry's distinguished name: */C=US/O=Digital Equipment* Corporation/ OU=Research/CN=Albert Einstein
- How to use the OM Create function to create a DS-DN OM private object (name) and how to use the OM Put function to copy the distinguished name from the public object (cpub_dn) into the newly created private object (name).
- How to use the Receive Result function to obtain the result of the Read function.
- How to use the OM Get function to copy the attributes of the Read-Result OM private object into the Read-Result OM public object (Spub_Result) for examination.

  The OM Create, OM Put and the OM Get functions are assumed to succeed.

Example 2

```
OM_private_object bound_session, name, context;
{
     DS_status           status;
     OM_private_object   name;
     status = ds_read(bound_session, DS_DEFAULT_CONTEXT,
                      name, selection, &info, NULL);


     if (status == DS_SUCCESS)
     {
        printf("READ was successful\n");
     }
     else
     {
        printf("READ failed\n");
     }
}
```

The above example shows how to perform a synchronous Read operation. Note that the Invoke-ID argument is not needed and therefore set to NULL. The example assumes that all other arguments have been defined as shown in Example 1.

Example 3

```
{
     OM_workspace      workspace;
     OM_descriptor     cpub_dn[7];
     OM_descriptor     cpub_rdn0[3];
     OM_descriptor     cpub_rdn1[3];
     OM_descriptor     cpub_rdn2[3];
     OM_descriptor     cpub_rdn3[3];
     OM_descriptor     cpub_rdn4[3];
     OM_descriptor     cpub_ava0[4];
     OM_descriptor     cpub_ava1[4];
     OM_descriptor     cpub_ava2[4];
     OM_descriptor     cpub_ava3[4];
     OM_descriptor     cpub_ava4[4];
     OM_value_position desc_count;
     DS_status         status;
     OM_sint           invoke_id;
     OM_uint           completion_flag;
     DS_status         operation_status;
     OM_return_code    om_status;
     OM_private_object name, read_result;
     OM_public_object  spub_result;
     OMX_CLASS_DESC(     cpub_ava0[0], DS_C_AVA);
     OMX_ATTR_TYPE_DESC( cpub_ava0[1], DS_ATTRIBUTE_TYPE,
                                       DSX_TYPELESS_RDN);
     OMX_ZSTRING_DESC(   cpub_ava0[2], OM_S_PRINTABLE_STRING,
                                       DS_ATTRIBUTE_VALUES,
                                       "CDS");
     OMX_OM_NULL_DESC(   cpub_ava0[3]);
     OMX_CLASS_DESC(     cpub_ava1[0], DS_C_AVA);
     OMX_ATTR_TYPE_DESC( cpub_ava1[1], DS_ATTRIBUTE_TYPE,
                                       DSX_TYPELESS_RDN);
     OMX_ZSTRING_DESC(   cpub_ava1[2], OM_S_PRINTABLE_STRING,
                                       DS_ATTRIBUTE_VALUES,
                                       "Projects");
     OMX_OM_NULL_DESC(   cpub_ava1[3]);
     OMX_CLASS_DESC(     cpub_ava2[0], DS_C_AVA);
     OMX_ATTR_TYPE_DESC( cpub_ava2[1], DS_ATTRIBUTE_TYPE,
                                       DS_A_ORG_UNIT_NAME);
     OMX_ZSTRING_DESC(   cpub_ava2[2], OM_S_PRINTABLE_STRING,
                                       DS_ATTRIBUTE_VALUES,
                                       "Research");
     OMX_OM_NULL_DESC(   cpub_ava2[3]);
     OMX_CLASS_DESC(     cpub_ava3[0], DS_C_AVA);
     OMX_ATTR_TYPE_DESC( cpub_ava3[1], DS_ATTRIBUTE_TYPE,
                                       DS_A_ORG_NAME);
     OMX_ZSTRING_DESC(   cpub_ava3[2], OM_S_PRINTABLE_STRING,
                                       DS_ATTRIBUTE_VALUES,
                                       "Digital Equipment Corporation");
     OMX_OM_NULL_DESC(   cpub_ava3[3]);
     OMX_CLASS_DESC(     cpub_ava4[0], DS_C_AVA);
     OMX_ATTR_TYPE_DESC( cpub_ava4[1], DS_ATTRIBUTE_TYPE,
                                       DS_A_COUNTRY_NAME);
     OMX_ZSTRING_DESC(   cpub_ava4[2], OM_S_PRINTABLE_STRING,
                                       DS_ATTRIBUTE_VALUES,
                                       "US");
     OMX_OM_NULL_DESC(   cpub_ava4[3]);
```

```
OMX_CLASS_DESC(     cpub_rdn0[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn0[1],       DS_AVAS, cpub_ava0);
OMX_OM_NULL_DESC(   cpub_rdn0[2]);
OMX_CLASS_DESC(     cpub_rdn1[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn1[1],       DS_AVAS, cpub_ava1);
OMX_OM_NULL_DESC(   cpub_rdn1[2]);
OMX_CLASS_DESC(     cpub_rdn2[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn2[1],       DS_AVAS, cpub_ava2);
OMX_OM_NULL_DESC(   cpub_rdn2[2]);
OMX_CLASS_DESC(     cpub_rdn3[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn3[1],       DS_AVAS, cpub_ava3);
OMX_OM_NULL_DESC(   cpub_rdn3[2]);
OMX_CLASS_DESC(     cpub_rdn4[0],       DS_C_DS_RDN);
OMX_OBJECT_DESC(    cpub_rdn4[1],       DS_AVAS, cpub_ava4);
OMX_OM_NULL_DESC(   cpub_rdn4[2]);
OMX_CLASS_DESC(     cpub_dn[0],         DS_C_DS_DN);
OMX_OBJECT_DESC(    cpub_dn[1],         DS_RDNS, cpub_rdn4);
OMX_OBJECT_DESC(    cpub_dn[2],         DS_RDNS, cpub_rdn3);
OMX_OBJECT_DESC(    cpub_dn[3],         DS_RDNS, cpub_rdn2);
OMX_OBJECT_DESC(    cpub_dn[4],         DS_RDNS, cpub_rdn1);
OMX_OBJECT_DESC(    cpub_dn[5],         DS_RDNS, cpub_rdn0);
OMX_OM_NULL_DESC(   cpub_dn[6]);
/* create the OM private object: name */
om_status = om_create(DS_C_DS_DN, OM_FALSE, workspace, &name);
/* Copy the attribute list from the cpub_dn public object into   */
/* the name private object                                       */


om_status = om_put(name, OM_REPLACE_ALL, cpub_dn, 0,0,0);
/* call the ds_read function using Name as a parameter and */
/* specify that all attribute types and values be read.    */
/* Note that invoke_id parameter is may be set NULL in the */
/* case of synchronous operation.                          */
status = ds_read(bound_session, DS_DEFAULT_CONTEXT, name,
                 DS_SELECT_ALL_TYPES_AND_VALUES, &read_result,
                 NULL);


if (status == DS_SUCCESS)
{
   printf("READ request was successful\n");
   om_status = om_get(read_result, OM_NO_EXCLUSIONS,
                      0, 0, 0, OM_ALL_VALUES,
                      &spub_result, &desc_count);
   if (om_status == OM_SUCCESS)
   {
       /* check desc_count != 0 */
       /* results now available in public object */
       /* spub_result                            */
   }
   else
   {
       /* error getting results */
       /* search_result not deleted */
   }
}
else
{
```

```
        printf("READ request failed\n");
    }
}
```

The above example shows the synchronous reading of all attribute types and values from the CDS entry */.../C=US/O=Digital Equipment* Corporation/ OU=Research/Projects/CDS

Note the use of the special attribute type DSX_TYPELESS_RDN in the rightmost RDNs of the name. The presence of one or more occurrences of this attribute type indicates to the XDS API that a name is a CDS distinguished name.

Note that the CDS global naming root /... need not be explicitly supplied as the first RDN in a CDS distinguished name. When the XDS API encounters a CDS distinguished name, it will internally prepend the CDS global naming root, unless one of the CDS local naming roots such as /.: or /: has been explicitly supplied.

A CDS local naming root, if desired, must be explicitly supplied as the first RDN of a distinguished name. It is specified with an attribute type of DSX_TYPELESS_RDN and an attribute value of .: or : as appropriate.

Note that the Invoke-ID argument is not needed for synchronous operation and is therefore set to NULL. The Bound-Session argument is assumed to have been set up as in Example 1.

ds_receive_result(3xds)

Name

ds_receive_result - This function retrieves the result of an asynchronously executed operation.

Syntax

Status = ds_receive_result*(Session, Completion-Flag, Operation-Status, Result,* Invoke-ID)

Argument	Data Type	Access
Session	OM_private_object	read
Completion-Flag	Unsigned Integer	write
Operation-Status	DS_status	write
Result	OM_private_object	write
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_receive_result*(session, completion_flag, operation_status, result,* invoke_id)

OM_private_object	session
OM_uint	completion_flag_return
DS_status	operation_status_return
OM_private_object	result_return
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session in which the operation was performed.

Completion-Flag

One of the following values to indicate the status of outstanding asynchronous operations:

- Completed-Operation. At least one outstanding asynchronous operation is completed and its result is available.
- Outstanding-Operations. There are outstanding asynchronous operations but none is completed.
- No-Outstanding-Operation. There are no outstanding asynchronous operations.

The result of the Completion-Flag parameter is valid if Status has the value Success.

Upon successful return with Completion-Flag having the value *completed-operation,* Status and Invoke-ID parameter values for the completed operation are returned.

Operation-Status

Takes an error value if an error occurred during the execution of the asynchronous directory operation. If no error occurred, then it takes the value *success*. The possible error values are listed for each individual operation in the corresponding function description. This result is only valid if the status has the value *success* and Completion-Flag has the value *completed-operation*.

Result

The result of the completed asynchronous operation. Its value is the constant Null-Result if the operation was one that does not return a result (Add-Entry, Modify-Entry, Modify-RDN, or Remove-Entry). Otherwise it

is an OM object of the appropriate OM class for the result of the asynchronous operation. You can check the class of the Result by using the OM functions. This result is only valid if the following conditions are true:

- Status has the value *success*
- Completion-Flag has the value *completed-operation*
- Operation-Status has the value *success*

Invoke-ID

The Invoke-ID of the operation whose result is being returned.

This result is valid if the Status has the value *success* and Completion-Flag has the value *completed-operation*.

Description

This function is used to retrieve the completed results of an outstanding asynchronous operation.

The function results include two status indications. One, called Status, indicates that the function call itself was successful and is always returned. The other, called Operation-Status, is used to return the status of the completed asynchronous operation and is only returned if there is one. See *DEC X.500 Directory Service Programming* for information about calling functions asynchronously.

DCE Notes

The DCE XDS interface does not support asynchronous operations.

Return Value

DS_SUCCESS	The operation completed successfully.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants are returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error object:

- Library-Error, with Problem attribute values of *bad-session*,or *miscellaneous*

Any errors related to the completed asynchronous operation are reported in Operation-Status as described above.

Example

The following code extract shows an example call to the Receive Result function.

The Receive Result function is used to obtain the result of an outstanding asynchronous operation.

```
{
    /* Call the Modify Entry function asynchronously using the   */
    /* changes object as a parameter. The Asynchronous attribute */
    /* on the OM Context object has value True                   */
    status = ds_modify_entry(session,context,name,changes,&invoke_id);
    if (status == DS_SUCCESS)
    {...}
    else
    {...}
    /* now wait for the response... */
    completion_flag = DS_OUTSTANDING_OPERATIONS;
    /* loop around calls to receive_result() until we get one back */
    while ((status == DS_SUCCESS) &&
           (completion_flag == DS_OUTSTANDING_OPERATIONS))
    {
        status = ds_receive_result(bound_session, &completion_flag,
                                   &operation_status,
                                   &modify_entry_result,
                                   &invoke_id);
        if (status == DS_SUCCESS)
        {
            switch (completion_flag)
            {
                case DS_COMPLETED_OPERATION:
                /* operation is complete */
                break;
                case DS_OUTSTANDING_OPERATIONS:
                ...
                break;
                case DS_NO_OUTSTANDING_OPERATION:
                ...
                break;
            }
        }
    }
}
```

The Receive Result function uses, as input, the Invoke-ID argument output from the asynchronous function.

ds_remove_entry(3xds)

Name

ds_remove_entry - Removes an entry from the Directory Information Tree (DIT).

Syntax

Status = ds_remove_entry*(Session, Context, Name, Invoke-ID*)

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read
Invoke-ID	Integer	write
Status	DS_status	

C Binding

DS_status ds_remove_entry*(session, context, name, invoke_id_return)*

OM_private_object	session
OM_private_object	context
OM_object	name
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The directory context to be used for this operation. The Size-Limit and Dont-Dereference-Aliases Context parameters do not apply to this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object containing the name of the target entry. Any aliases in the name will not be dereferenced.

Invoke-ID

The Invoke-ID of an asynchronous directory operation.

Description

This function is used to remove an entry from the Directory. This may be an object entry or an alias entry. The entry must not have any subordinate entries.

DCE Notes

Ideally, the user does not know whether X.500 or CDS is actually handling the DCE naming operations. There are, however, some situations where naming results will differ depending on which service is handling the operation. (The **intro** reference page for XDS functions describes the general differences between operations on X.500 and CDS.)

Note the following issues for the Remove Entry function:

- All CDS operations are synchronous. If a CDS operation is attempted and the Context parameter Asynchronous has been set true, a Library-Error, *not-supported*, is returned.
- When a CDS name is passed to XDS and DCE is not installed, a Library-Error, *not-supported*, is returned. This error is also returned when an X.500 name is passed to XDS, and X.500 is not installed.
- In CDS, the name parameter supplied to the Remove Entry function must ultimately resolve to the name of a leaf (that is, a CDS Object) entry; otherwise, the Name-Error *no-such-object* is returned. The function never interprets the name parameter as the name of a CDS Directory entry.

Return Value

DS_SUCCESS	The entry was removed, if the operation was invoked synchronously. The operation,was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with Problem attribute values of *bad-argument*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*, *not-supported* or *too-many-* operations
- Name-Error, *no-such-object*
- Referral
- Security-Error
- Service-Error
- Update-Error
- Communications-Error

Examples

The following code extracts show an example call to the Remove Entry function. The Remove Entry function is used to remove an existing directory entry.

There are two examples. The first example shows how to perform an asynchronous Remove Entry operation. The second example shows how to perform a synchronous Remove Entry operation.

The Bound_Session argument contains the identity of a session returned from an earlier call to the Bind function. This object identifies the session through which the request should be issued. The Name argument and the Context argument are assumed to have been previously defined. Examples of how to define a Name argument, including an example of a CDS Name argument, are shown in the Read function. An example of how to define a Context argument is shown in the Add Entry function.

Example 1

```
OM_private_object bound_session, context, name;
{
    DS_status         status;
    OM_sint           invoke_id;
    OM_uint           completion_flag;
    DS_status         operation_status;
    OM_private_object remove_entry_result;
    /* Call the Remove Entry function */
    status = ds_remove_entry(bound_session, context, name,
                             &invoke_id);


    if (status == DS_SUCCESS)
    {
        printf("REMOVE ENTRY request was successful\n");
    }
    else
    {
        printf("REMOVE ENTRY request failed\n");
    }
    /* now wait for the response... */
    completion_flag = DS_OUTSTANDING_OPERATIONS;
    /* loop around calls to receive_result() until we get one back */
    while ((status == DS_SUCCESS) &&
           (completion_flag == DS_OUTSTANDING_OPERATIONS))
    {
        status = ds_receive_result(bound_session, &completion_flag,
                                   &operation_status,
                                   &remove_entry_result,
                                   &invoke_id);
        if (status == DS_SUCCESS)
        {
            switch (completion_flag)
            {
                case DS_COMPLETED_OPERATION:
                /* we have a completed operation */
                /* check operation_status */
```

```
                break;
                case DS_OUTSTANDING_OPERATIONS:
                ...
                break;
                case DS_NO_OUTSTANDING_OPERATION:
                ...
                break;
            }
        }
    }
}
```

The above example removes the directory entry, identified in the Name argument, from the directory. Since the operation is executed asynchronously, an invoke identifier is returned in the Invoke-ID argument. This uniquely identifies this specific operation and is therefore used in the subsequent Receive Result function to obtain the result of the operation.

Example 2

```
OM_private_object bound_session, context, name;
{
    DS_status         status;
    OM_private_object changes;
    status = ds_remove_entry(bound_session, DS_DEFAULT_CONTEXT,
                             name, changes, NULL);
    if (status == DS_SUCCESS)
    {
        printf("REMOVE_ENTRY was successful\n");
    }
    else
    {
        printf("REMOVE_ENTRY failed\n");
    }
    return status;
}
```

ds_search(3xds)

Name

ds_search - Finds entries of interest in a portion of the Directory.

Syntax

Status = ds_search*(Session, Context, Name, Subset, Filter, Search_Aliases,* Selection, Result, Invoke-ID)

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read

Argument	Data Type	Access
Subset	Integer	read
Filter	OM_object	read
Search_Aliases	OM_boolean	read
Selection	OM_object	read
Result	OM_private_object	write
Status	DS_status	write
Invoke-ID	Integer	

C Binding

DS_status ds_search*(session, context, name, subset, filter, search_aliases, selection, result_return, invoke_id_return*)

OM_private_object	session
OM_private_object	context
OM_object	name
OM_sint	subset
OM_object	filter
OM_boolean	search_aliases
OM_object	selection
OM_private_object	result_return
OM_sint	invoke_id_return

Arguments

Session

The Session OM private object that was returned by the Bind function, identifying the directory session to be used.

Context

The Context parameters to be used for this operation. This argument must be a Context OM private object or the constant Default-Context.

Name

A Name OM object containing the name of the target entry, which forms the base of the search. Any aliases in the name will be dereferenced unless prohibited by the Dont-Dereference-Aliases Context parameter.

Subset

The search limit that specifies a portion of the Directory tom be searched. Its value must be one of:

- *base-object*, meaning search just the target entry
- *one-level*, meaning search just the immediate subordinates of the target entry
- *whole-subtree*, meaning search the target entry and all its subordinates

Filter

The filter is used to prevent unwanted entries being returned in the results of the search. Information is only returned on entries that satisfy the filter. The constant No-Filter can be used as the value of this argument if you want to search all entries. This corresponds to a filter with a value of *and* for the attribute Filter-Type, and no values of the attributes Filters or Filter-Items.

Search-Aliases

Any aliases in the subordinate entries being searched are dereferenced if the value of this argument is *true*. They are not dereferenced if its value is *false*. Note that Digital's X.500 DSA does not honor this control; it assumes the value is *false* and does not dereference aliases.

Selection

An Entry-Information-Selection OM object or a constant specifying what information from the named entry is requested. Information about no attributes, all attributes, or just a named set can be chosen. Attribute types are always returned, but the attribute values need not be. The following constants can be used:

- Select-No-Attributes, to verify the existence of an entry
- Select_All-Types, to return just the types of all attributes
- Select-All-Types-And-Values, to return the types and values of all attributes

Result

A Search-Result OM private object, passed by reference, containing the requested information from each object in the search space that satisfied the filter. The distinguished name of the target object is present if an alias was dereferenced. Additionally there may be a partial outcome qualifier that indicates the result is incomplete. It also explains why it is not complete and how it could be completed.

Invoke-ID

The Invoke-ID of an asynchronous operation.

Description

This function is used to search a portion of the directory and return selected information from the entries of interest. It is possible that the information will be incomplete in some circumstances.

DCE Notes

CDS does not support the Search function. It returns with the Service-Error *unwilling-to-perform*.

Return Value

DS_SUCCESS	The target object was located, if the operation was invoked synchronously. The operation was initiated, if it was invoked asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with Problem attribute values of *bad-argument*, *bad-context*, *bad-name*, *bad-session*, *miscellaneous*, *missing-type*,or *too-many-operations*
- Attribute-Error
- Name-Error
- Referral
- Security-Error
- Service-Error
- Communications-Error

An unfiltered search of just the base object succeeds even if none of the requested attributes is found while Read fails with the same selected attributes.

A Security-Error, *insufficient-access-rights*, is only reported where access rights prohibit the reading of all requested attribute values.

Examples

The following code extract shows an example call to the Search function. The Search function is used to search the directory for a specific entry and then extract the values of the Surname and the Title attributes from that entry.

There are two examples. The first example shows how to perform an asynchronous Search operation. The second example shows how to perform a synchronous Search operation.

The Bound_Session argument contains the identity of a session returned from an earlier call to the Bind function. This object identifies the session through which the request should be issued. The Name argument and the Context

argument are assumed to have been previously defined. Examples of how to define a Name argument are shown in the Read function. An example of how to define a Context argument is shown in the Add Entry function.

Example 1

```
{
    OM_private_object bound_session, context, name;
    OM_workspace      workspace;
    OM_descriptor     cpub_eis[5];
    OM_value_position desc_count;
    DS_status         status;
    OM_private_object search_result;
    OM_sint           invoke_id;
    OM_uint           completion_flag;
    DS_status         operation_status;
    OM_return_code    om_status;
    OM_public_object  spub_result;
    OM_value_position desc_count;
    OM_private_object selection;
    /* create a descriptor list for surname and title of class */
    /* entry information selection                             */
    OMX_CLASS_DESC(       cpub_eis[0], DS_C_ENTRY_INFO_SELECTION);
    OMX_ATTR_TYPE_DESC(   cpub_eis[1], DS_ATTRIBUTES_SELECTED,
                                       DS_A_SURNAME);
    OMX_ATTR_TYPE_DESC(   cpub_eis[2], DS_ATTRIBUTES_SELECTED,
                                       DS_A_TITLE);
    OMX_ENUM_DESC(        cpub_eis[3], DS_INFO_TYPE,
                                       DS_TYPES_ONLY);
    OMX_OM_NULL_DESC(     cpub_eis[4]);
    /* Create an OM private object called selection */
    om_status = om_create(DS_C_ENTRY_INFO_SELECTION,OM_FALSE,
                          workspace, &selection);
    /* Object created, now put in the attributes from cpub_eis */
    om_status = om_put(selection, OM_REPLACE_ALL, cpub_eis ,0,0,0);
    /* now start the search using selection as a parameter*/
    status = ds_search(bound_session, context, name, DS_ONE_LEVEL,
                       DS_NO_FILTER, OM_FALSE, selection,
                       &search_result, &invoke_id);
    completion_flag = DS_OUTSTANDING_OPERATIONS;
    /* loop around calls to receive_result() until we get one back */
    while ((status == DS_SUCCESS) &&
           (completion_flag == DS_OUTSTANDING_OPERATIONS))
    {
        status = ds_receive_result(bound_session, &completion_flag,
                                   &operation_status, &search_result,
                                   &invoke_id);


        if (status == DS_SUCCESS)
        {
            switch (completion_flag)
            {
            case DS_COMPLETED_OPERATION:
            /* we have a completed operation     */
            /* now see what we have got back ... */
```

```
            if (operation_status == DS_SUCCESS)
            {

                om_status = om_get(search_result, OM_NO_EXCLUSIONS,
                                   0, 0, 0, OM_ALL_VALUES,
                                   &spub_result, &desc_count);
                if (om_status == OM_SUCCESS)
                {
                   /* results now available as a public object */
                   /* check desc_count != 0 */
                   /* delete the search result... */
                   om_status = om_delete(search_result);
                }
                else
                {
                   /* error getting results */
                   /* search_result not deleted */
                }
            }
            else
            {...}
            break;
            case DS_COMPLETED_OPERATION:
            ...
            break;
            case DS_COMPLETED_OPERATION:
            ...
            break;
        }
    }
}
```

The above example shows the following:

- How to define an Entry-Information-Selection OM public object (cpub_eis) containing details of the information that is to be returned from the search.
- How to use the OM Create function to create a private object (selection) and how to use the OM Put function to copy the details of the required information from the Entry-Information-Selection OM public object (cpub_eis) into the newly-created Entry-Information-Selection OM private object (selection).
- How to obtain the result of the Search function using the Receive Result function.
- How to use the OM Get function to copy the attributes of the Search-Result OM private object into the Search-Result OM public object (Spub_Result) for examination.

The OM Create, OM Put, OM Get and OM Delete functions are assumed to succeed.

Example 2

```
{
    OM_private_object bound_session, context, name;
    OM_value_position desc_count;
    DS_status         status;
    OM_private_object search_result;
    OM_private_object selection;
    OM_public_object  spub_result;
    /* start the search using selection as a parameter */
    status = ds_search(bound_session, DS_DEFAULT_CONTEXT, name,
                       DS_ONE_LEVEL, DS_NO_FILTER, OM_FALSE,
                       selection, &search_result, NULL);
    if (status == DS_SUCCESS)
    {
        /* now see what we have got back ... */
        om_status = om_get(search_result, OM_NO_EXCLUSIONS,
                           0, 0, 0, OM_ALL_VALUES,
                           &spub_result, &desc_count);
        if (om_status == OM_SUCCESS)
        {
            /* results now available as a public object */
            /* check desc_count != 0 */
            /* delete the search result... */
            om_status = om_delete(search_result);
        }
        else
        {
            /* error getting results */
            /* search_result not deleted */
        }
    }
    else
    {...}
}
```

The above example shows how to perform a synchronous Search operation. Note that the Invoke-ID argument is not needed and NULL is used. This example assumes that the Selection argument has been defined as shown in Example 1.

ds_shutdown(3xds)

Name

ds_shutdown - Shuts down the interface and closes the workspace.

Syntax

Status = ds_shutdown*(Workspace)*

Argument	Data Type	Access
Workspace	OM_workspace	read

Argument	Data Type	Access
Status	DS_status	

C Binding

DS_status ds_shutdown*(workspace)*

OM_workspace	workspace

Arguments

Workspace

Specifies the workspace (obtained from a call to the Initialize function) that is to be deleted.

Description

This function shuts down the interface previously established by Initialize and enables the service to release resources.

After this function has been called, no OM objects or other data values associated with the workspace are valid, with the exception of client-generated public objects. You should call the Unbind function for all sessions in this workspace. You must not subsequently call any X.500 API functions that operate on OM objects in this workspace.

In order to ensure that resources are freed, applications should release all private objects by calling the OM Delete function for all top-level OM private objects before calling this function. This is not necessary for subobjects. Applications should also release all service-generated public objects by calling the OM Delete function. You can do this either before or after the calling of this function.

Return Value

DS_SUCCESS	The shutdown was completed, if the operation was invoked synchronously. The operation was initiated, if it was invoked. asynchronously.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

Errors

This function does not return any error objects.

Example

The following code extract shows an example call to the Shutdown function:

```
OM_workspace workspace;
{
  DS_status    status;
  /* Finally, close down the workspace */
  ds_status = ds_shutdown(workspace);
}
```

The Shutdown function closes down the workspace identified in the Workspace argument. The workspace identity is obtained from the Initialize function.

ds_unbind(3xds)

Name

ds_unbind - This function closes a directory session.

Syntax

Status = ds_unbind*(Session)*

Argument	Data Type	Access
Session	OM_private_object	read
Status	DS_status	

C Binding

DS_status ds_unbind*(session)*

OM_private_object session

Arguments

Session

The directory session that is to be unbound. This argument must be the Session OM private object that was returned by the Bind function, identifying the directory session. If the function succeeds, the value of the File-Descriptor OM attribute is *No-Valid-File-Descriptor.* The other OM attributes are unchanged.

Description

This function terminates the given directory session and makes the argument unavailable for use with all other interface functions except Bind.

The results of any outstanding asynchronous operations that were initiated using the given Session can no longer be received, and it is not possible to find out if they succeeded. It is therefore recommended that you obtain the results of all outstanding asynchronous operations by calling the Receive-Result function before calling Unbind.

It is possible to use the unbound session again as an argument to Bind, perhaps after modification by the Object Management functions.

Return Value

DS_SUCCESS	The operation completed successfully.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error object:

- Library-Error, with Problem attribute values of *bad-session*,or *miscellaneous*.

Example

The following code extract shows an example call to the Unbind function.

```
{
  OM_private_object bound_session;
  DS_status         status;


  status = ds_unbind(bound_session);
  if (status == DS_SUCCESS)
  {
      printf("UNBIND was successful\n");
  }
  else
  {
      printf("UNBIND failed\n");
  }
}
```

The Unbind function closes down a session established by the Bind function. The Bound_Session argument identifies the session to be closed.

ds_version(3xds)

Name

ds_version - Negotiates the features of the interface and service.

Syntax

Status = ds_version*(Feature-List, Workspace)*

Argument	Data Type	Access
Feature-List	DS_Feature	write/read
Workspace	OM_workspace	read
Status	DS_status	

C Binding

DS_status ds_version*(feature_list, workspace)*

DS_feature	feature_list[]
OM_workspace	workspace

Arguments

Feature-List

An ordered sequence of features, each represented by an object identifier. The sequence is terminated by an object identifier having no components (that is, a length of zero, and any value of the data pointer in the C representation).

Workspace

Specifies the workspace (obtained from a call to the Initialize function) for which the features are to be negotiated. The features will be in effect for operations that use the workspace or directory sessions associated with the workspace.

Description

This function negotiates features of the interface that are represented by object identifiers. Features are negotiated after a workspace has been initialized.

Negotiable features include the Basic-Directory-Contents Package, the Strong-Authentication Package, and the MHS Directory User Package.

The Gradient DCE for Tru64 UNIX implementation of this function does not support the features listed above, but supports one extension, DSX-RET-X500-BIND-ERR-FTR. This feature guarantees that the Bind function will always return an error if it fails to connect to an X.500 directory. This feature is useful if the system where your application runs is capable of simultaneous connections to both CDS and X.500 directories in the same XDS session. In other circumstances, this feature is not needed.

Return Value

DS_SUCCESS	The features were successfully negotiated.
DS_NO_WORKSPACE	A workspace has not been set up by a call to the Initialize function.

If neither of these constants is returned, then the function returns a pointer to an error object of one of the classes listed below.

Errors

This function can return pointers to the following error objects:

- Library-Error, with the Problem attribute values of *miscellaneous, bad_workspace*.
- System-Error

Example

The following code extract shows an example call to the Version function.

```
{
    OM_workspace       workspace;
    DS_feature         feature_list[];
    DS_status          status;
    status = ds_version(feature_list, workspace);
    if (status == DS_SUCCESS)
    {
      printf("VERSION was successful\n");
    }
    else
    {
      printf("VERSION failed\n");
    }
}
```

dsX_trace_object(3xds)

Name

dsX_trace_object - Displays an explanation of the content of an object on the current output device.

Syntax

(void) dsX_trace_object*(Object)*

Argument	Data Type	Access
Object	OM_object	read

C Binding

dsX_trace_object*(object)*

OM_object object

Arguments

OM_object

The object whose content you want to inspect.

Description

This function displays on the current output device information about the content of an OM object, as follows:

- A full expansion of a public object
- The type of a private object
- Details of the content of an error object
- For a name object or AVA encoded in ASN.1, both the ASCII and hexadecimal representations of the ASN.1 encoding

The routine also checks for null pointers.

Errors

This function does not return any errors.

Example

The following code extract shows an example call to the Trace Object function:

```
{
  OM_workspace workspace;
  OM_return_code status;
  OM_object session = NULL;


  status = om_create(DS_C_SESSION,OM_TRUE,workspace,&session);
  if (status == OM_SUCCESS)
    {
```

```
        dsX_trace_object(session);
    }
}
```

CHAPTER 4

XDS Object Management Reference Pages

This chapter provides reference pages for the X/Open Object Management (XOM) API functions. The Gradient DCE for Tru64 UNIX XOM implementation supports two additional ways to create private objects. The functions are as follows:

- The **om_encode** function uses the encoding rules you specify to create a private object independent of the original private object.
- The **om_decode** function decodes the ASN.1 of the original object to create the new one. You must specify the class of the existing object and the rules used to encode it.

om_intro(3xom)

Name

om_intro - This reference page introduces the OM API functions.

Syntax

#include <xom.h>

Description

This reference page defines the functions of the C interface in the Digital X.500 product.

Function	Description
om_copy	Copies a private object
om_copy_value	Copies a string between private objects
om_create	Creates a private object
om_decode	Creates a new private object that decodes an existing ASN.1 private object.
om_delete	Deletes a private or service-generated object
om_encode	Creates a new private object that encodes an existing private object.

Function	Description
om_get	Gets copies of attribute values from a private object
om_instance	Tests an object's class
om_put	Puts attribute values into a private object
om_read	Reads a segment of a string in a private object
om_remove	Removes attribute values from a private object
om_write	Writes a segment of a string into a private object

As indicated in the table, the service interface comprises a number of functions whose purpose and range of capabilities are summarized as follows:

Function	Description
om_copy	This function creates an independent copy of an existing private object and all its subobjects. The copy is placed in the original's workspace, or in another specified by the XOM application.
om_copy_value	This function replaces an existing attribute value or inserts a new value in one private object with a copy of an existing attribute value found in another. Both values must be strings.
om_create	This function creates a new private object that is an instance of a particular class. The object can be initialized with the attribute values specified as initial in the class definition. The service does not permit the API user to explicitly create instances of all classes, but rather only those indicated by a package's definition as having this property.
om_decode	This function creates a new private object by decoding the ASN.1 of the original object.
om_delete	This function deletes a service-generated public object, or makes a private object inaccessible.
om_encode	This function creates a new private object, the encoding, which exactly and independently encodes an existing private object, the original.
om_get	This function creates a new public object that is an exact but independent copy of an existing private object. The client can request certain exclusions, each of which reduces the copy to a part of the original. The client can also request that values be converted from one syntax to another before they are returned. The copy can exclude: attributes of types other than those specified, values at positions other than those specified within an attribute, the values of multivalued attributes, copies of (not handles for) subobjects, or all attribute values (revealing only an attribute's presence).
om_instance	This function determines whether an object is an instance of a particular class. The client can determine an object's class simply by inspection. This function is useful because it reveals that an object is an instance of a particular class, even if the class is an instance of a subclass of that class.

Function	Description
om_put	This function places or replaces in one private object copies of the attribute values of another public object or private object. The source values can be inserted before any existing destination values, before the value at a specified position in the destination attribute, or after any existing destination values. Alternatively, the source values can be substituted for any existing destination values or for the values at specified positions in the destination attribute.
om_read	This function reads a segment of a value of an attribute of a private object. The value must be a string. The value can first be converted from one syntax to another. The function enables the client to read an arbitrarily long value without requiring that the service place a copy of the entire value in memory.
om_remove	This function removes and discards particular values of an attribute of a private object. The attribute itself is removed if no values remain.
om_write	This function writes a segment of a value of an attribute to a private object. The value must be a string. The segment can first be converted from onesyntax to another. The written segment becomes the value's last segment because any elements beyond it are discarded. The function enables the client to write an arbitrarily long value without having to place a copy of the entire value in memory.

In the C interface, the functions are realized by macros. The function prototype in the synopsis of a function's specification shows the client's view of the function.

The intent of the interface definition is that each function be atomic; that is, either it carries out its assigned task in full and reports success, or it fails to carry out even a part of the task and reports an exeception. However, the service does not guarantee that a task is always carried out in full.

om_copy(3xom)

Name

om_copy - Creates a copy of an existing private object.

Syntax

OM_return_code om_copy*(original, workspace, copy)*

Argument	Data Type	Access
original	OM_private object	read
workspace	OM_workspace	read
copy	OM_private_object	write
return_code	OM_return_code	

C Binding

OM_return_code om_copy*(original, workspace, copy)*

OM_private_object original,
OM_workspace workspace,
OM_private_object #copy

Arguments

Original

The original private object.

Workspace

The workspace in which the Service creates the copy. The workspace that the Client specifies in this argument must be one that is associated with a package containing the class of the original object.

Copy

The copy of the original object. The Service returns this argument if the Return Code of the function is OM_SUCCESS.

Description

This function creates a new private object, the copy, which is an exact but independent copy of an existing private object, the original. The function also copies the original's subobjects, if it has any.

The Client can specify a workspace in which the Service should place the copy. If the Client does not do so, the Service places the copy in the original's workspace.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_CLASS	There is an undefined class identifier
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_WORKSPACE	You have specified a nonexistent workspace
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code

OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code
OM_TOO_MANY_VALUES	An implementation limit prevents the addition to an object of another attribute value

Examples

The following example shows how to copy an object of the MH class Local NDR ndr. The copy is `ndr_copy`.

```
OM_return_code      result;
 OM_private_object   ndr,
                     ndr_copy;
  OM_workspace        workspace;
          result =   copy (ndr,
                           /* object to be copied */
                     workspace,
                           /* workspace in which to create copy */
                     &ndr_copy);
                           /* the copy */
```

om_copy_value(3xom)

Name

om_copy_value - Copies a value (string) from a private object and places it in another private object.

Syntax

OM_return_code om_copy_value*(source, source_type, source_value_position, destination, destination_type, destination_value_position)*

Argument	Data Type	Access
source	OM_private_object	read
source_type	OM_type	read
source_value_position	OM_value_position	read
destination	OM_private_object	read
destination_type	OM_type	read
destination_value_position	OM_value_position	read
return_code	OM_return_code	

C Binding

OM_return_code om_copy_value*(source, source_type, source_value_position,destination, destination_type, destination_value_position)*

OM_private_object source,
OM_type source_type,
OM_value_position source_value_position,
OM_private_object destination,
OM_type destination_type,
OM_value_position destination_value_position

Arguments

Source

The object from which you want to copy the value.

Source Type

The type of the attribute value from which you want tocopy the value.

Source Value Position

The position within the attribute of the value to be copied.

Destination

The object to which you want to copy the value.

Destination Type

The type of the attribute to which you want to copy thevalue.

Destination Value Position

The position within the destination attribute at which you want to place the copied value. If the value of this argument exceeds the number of values in the Destination attribute, then it is taken to be equal to that number.

Description

This function either replaces, or fills in for the first time, an attribute value in the destination object with a copy of an attribute value from the source object. The source value should be a string. The copy has the same syntax as the source value.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_DECLINED	The function does not apply to the object to which it is addressed
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention

OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_TYPE	There is an undefined type identifier
OM_NOT_PRESENT	An expected attribute value is missing
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument or as the receptacle for a function result
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code
OM_WRONG_VALUE_LENGTH	There is an attribute with a value that violates the value length constraints in force
OM_WRONG_VALUE_SYNTAX	There is an attribute value with an illegal syntax
OM_WRONG_VALUE_TYPE	There is an attribute value of an illegal type

Examples

The following example shows the copying of a string value between two objects of the MH class Delivery Envelope (`envelope1` and `envelope2`).

```
OM_private_object    envelope1,
                     envelope2;
OM_return_code       result;
     result = om_copy_value (envelope1,
                             /* source object */
                              MH_T_CONTENT_IDENTIFIER,
                             /* source attribute */
                              0,
                             /* position of value in source attribute */

                            envelope2,
                             /* destination object */
                              MH_T_CONTENT_IDENTIFIER,
                             /* destination attribute */
                            0);
                             /* position in destination attribute */
```

This call to Copy Value reads the string value at position 0 in the Content Identifier attribute of `envelope1`. This value is then copied to position 0 in the Content Identifier attribute of `envelope2`.

om_create(3xom)

Name

om_create - Creates a new private object that is an instance of a particular class.

Syntax

OM_return_code om_create*(class, initialize, workspace, object)*

Argument	Data Type	Access
class	OM_object_identifier	read
initialize	OM_boolean	read
workspace	OM_workspace	read
object	OM_private_object	write
return_code	OM_return_code	

C Binding

OM_return_code om_create*(class, initialize, workspace, object)*

OM_object_identifier class,
OM_boolean initialize,
OM_workspace workspace,
OM_private_object #object

Arguments

Class

The class of the object you are creating. It must be a concrete class.

Initialize

If you set this argument to OM_TRUE, the object that you create has some of its attributes initialized. These are the attributes for which initial values are specified in the class definition table. You can find these class definition tables in the documentation for the X.400 and X.500 APIs. If you set this argument to OM_FALSE, the object you create has only its Class attribute initialized.

Workspace

The workspace in which the Service should create the object. The class you specify for the object must be in a package that you already associated with this workspace.

Object

This is the created object. The Service returns this argument if the Return Code of the function is OM_SUCCESS.

Description

This function creates a private object in the workspace that you specify.

You can add new values and replace or remove existing values, any time after the object has been created. In this way, you can create any possible instance of the object's class.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_DECLINED	The function does not apply to the object to which it is addressed
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_CLASS	There is an undefined class identifier
OM_NO_SUCH_WORKSPACE	You have specified a nonexistent workspace
OM_PERMANENT_ERROR	A class is abstract, not concrete OM_NOT_CONCRETE The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code

Examples

The first example shows how to create a private object of Message Handling (MH) class Local Per-recipient NDR, with the Initialize argument set to OM_FALSE; the function does not initialize any of the object's attributes.

The second example shows the creation of a private object of X.500 class Entry-Info-Selection, with the Initialize argument set to OM_TRUE; the function initializes two of the object's attributes.

```
OM_private_object    ndr;
OM_workspace         workspace;
OM_return_code       result;
```

```
     result = om_create (MH_C_LOCAL_PER_RECIP_NDR,
                             /* class of object */
                         OM_FALSE,
                             /* do not initialize attributes */
                         workspace,
                             /* workspace in which object created */
                         &ndr);
                             /* created object */

OM_private_object    select_info;
OM_workspace         workspace;
OM_return_code       result;
     result = om_create (DS_C_ENTRY_INFO_SELECTION,
                             /* class of object */
                         OM_TRUE,
                             /* initialize attributes */
                         workspace,
                             /* workspace in which object created */
                         &select_info);
                             /* created object */
```

om_decode(3xom)

Name

om_decode - Creates a new private object that decodes an existing ASN.1 private object.

Syntax

OM_return_code om_decode*(encoding, original)*

Argument	Data Type	Access
encoding	OM_private_object	read
original	OM_private_object	write
return_code	OM_return_code	

C Binding

OM_return_code om_decode*(encoding, original)*

OM_private_object encoding,
OM_private_object #original

Arguments

Encoding

The encoded object that you want to decode. It must be an instance of the Encoding class.

Original

An object that is the decoded version of the encoding. The Service creates this object in the workspace in which the encoding is located. The Service returns this argument if the Return Code of the function is OM_SUCCESS.

Description

This function creates a new private object by decoding the ASN.1 of the original object.

In the Encoding argument, you specify the class of the existing object and the rules used to encode it. In the current version of the OM API, you must specify ASN.1 BER.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_ENCODING_INVALID	The value of the Object Encoding is invalid
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_CLASS	There is an undefined class identifier
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_RULES	There is an undefined rules identifier
OM_NOT_AN_ENCODING	There is an object that is not an instance of the Encoding class
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code
OM_TOO_MANY_VALUES	An implementation limit prevents the addition to an object of another attribute value
OM_WRONG_VALUE_LENGTH	There is an attribute with a value that violates the value length constraints in force

OM_WRONG_VALUE_MAKEUP	There is an attribute with a value that violates a constraint of its syntax
OM_WRONG_VALUE_NUMBER	There is an attribute with a value that violates the value number constraints in force
OM_WRONG_VALUE_SYNTAX	There is an attribute value with an illegal syntax
OM_WRONG_VALUE_TYPE	There is an attribute value with an illegal type

Examples

The following example shows the decoding of the object encoded in the code example from OSI-Abstract-Data Manipulation. The encoded object is `encoding`, and the decoded object is `decoded_object` .

```
OM_return_code      result;
OM_private_object   encoding,
                    decoded_object;
     result = om_decode (encoding,
                              /* object to be decoded */
                         &decoded_object);
                              /* decoded object */
```

om_delete(3xom)

Name

om_delete - Deletes a service-generated public object or makes a private object inaccessible.

Syntax

OM_return_code om_delete*(subject)*

Argument	Data Type	Access
subject	OM_object	read
return_code	OM_return_code	

C Binding

OM_return_code om_delete*(subject)*

OM_object subject

Arguments

Subject

The object that you want the Service to delete. It must be a service-generated public object or a private object. If the object that you specify is a client-generated public object, the function returns an error status.

Description

This function deletes a service-generated public object, or makes a private object inaccessible.

When you apply this function to a service-generated public object, the function deletes the object and releases the resources associated with it. The resources include the space occupied by descriptors and attribute values. The function also deletes all public subobjects of the subject. This function does not delete private subobjects.

When you apply this function to a private object, the function makes the object inaccessible by making its Handle invalid. The function also makes invalid the Handles of any private subobjects of the subject. Note that the effect of using an object's Handle once it has been made invalid is undefined.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_SYNTAX	There is an undefined syntax identifier
OM_NO_SUCH_TYPE	There is an undefined type identifier
OM_NOT_THE_SERVICES	There is an object that is Client generated when it should be Service generated or private
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code

Examples

The following example shows how to delete a service-generated public object of the X.500 class Entry-Info-Selection (`info_select_copy`). The object is a public copy of a private object of class Entry-Info-Selection `info_select` .

```
OM_return_code      result;
OM_private_object   info_select;
OM_object           info_select_copy;
OM_value_position   total_number;
     result = om_get (info_select,
                             /* object to be copied */
                      OM_NO_EXCLUSIONS,
                             /* no exclusions */
                      NULL,
                             /* ignored because no exclusions specified */
                      OM_FALSE,0,0,
                             /* no translation into local char set */
                      &info_select_copy,
                             /* the copy */
                      &total_number);
                             /* number of attributes copied */
/* Examine info_select_copy public object using C programming language
constructs */
/* Delete object when finished with it */
     result = om_delete (info_select_copy);
                             /* the object to be deleted */
```

om_encode(3xom)

Name

om_encode - Creates a new private object that encodes an existing private object.

Syntax

OM_return_code om_encode*(original, rules, encoding)*

Argument	Data Type	Access
original	OM_private_object	read
rules	OM_object_identifier	read
encoding	OM_private_object	write
return_code	OM_return_code	

C Binding

OM_return_code om_encode*(original, rules, encoding)*

OM_private_object original,

OM_object_identifier rules,
OM_private_object #encoding

Arguments

Original

The object you want to encode.

Rules

The set of rules that the Service must follow to produce an encoding. In this version of the OM API, you can only specify ASN.1 BER.

Encoding

An object that is the encoded version of the original. The Service creates this object in the workspace in which the original is located. The Service returns this argument if the Return Code of the function is OM_SUCCESS. The returned object is an instance of the Encoding class.

Description

This function creates a new private object, the encoding, which exactly and independently encodes an existing private object, the original. When you apply this function to a private object, the function uses the encoding rules you specify to create a new private object. The new encoded private object is independent of the original private object.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_DECLINED	The function does not apply to the object to which it is addressed
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_RULES	There is an undefined rules identifier
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system

OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code

Examples

The following example shows the encoding of an object of the MH class Report ((`encodable_object`). The object is encoded according to the rules `OM_BER`, and the encoded object is `encoding`.

```
OM_return_code      result;
OM_private_object   encodable_object,
                    encoding;
     result = om_encode (encodable_object,
                             /* object to be encoded */
                         OM_BER,
                             /* encoding rules */
                         &encoding);
                             /* encoded object */
```

om_get(3xom)

Name

om_get - Creates a new public object that is a copy of the whole or part of a private object.

Syntax

OM_return_code om_get*(original, exclusions, included_types, local_strings, initial_value, limiting_value, copy, total_number)*

Argument	Data Type	Access
original	OM_private_object	read
exclusions	OM_exclusions	read
included_types	OM_type_list	read
local_strings	OM_boolean	read
initial_value	OM_value_position	read
limiting_value	OM_value_position	read
copy	OM_public_object	write
total_number	OM_value_position	write
return_code	OM_return_code	

C Binding

OM_return_code om_get*(original, exclusions, included_types, local_strings, initial_value, limiting_value, copy, total_number)*

OM_private_object original,
OM_exclusions exclusions,
OM_type_list included_types,
OM_boolean local_strings,
OM_value_position initial_value,
OM_value_position limiting_value,
OM_public_object #copy,
OM_value_position #total_number

Arguments

Original

The private object, all or part of which you want to copy.

Exclusions

A list of zero or more values, each of which reduces the copy to a portion of the original. The exclusions apply to the attributes of the original object, but not to the attributes of its subobjects. This argument has one or more of the following values:

- OM_EXCLUDE_ALL_BUT_THESE_TYPES

  The copy includes descriptors of attributes of specified types only.

- OM_EXCLUDE_MULTIPLES

  The copy includes a single descriptor for each attribute having two or more values, instead of one descriptor for each value. Each such descriptor contains no attribute value, and the No-Value bit of the syntax component is set. If the attribute has values of two or more syntaxes, the descriptor identifies one of those syntaxes. Which syntax it identifies is unspecified.

- OM_EXCLUDE_ALL_BUT_THESE_VALUES

  The copy includes descriptors encompassing only values at specified positions within an attribute.

- OM_EXCLUDE_VALUES

  The copy includes a single descriptor for each attribute value, but the descriptor does not contain the value, and the No-Value bit of the syntax component is set.

- OM_EXCLUDE_SUBOBJECTS

The copy includes a descriptor for each value which has a syntax of object. Each descriptor contains an object Handle for the original private subobject, instead of a public copy of the original private subobject. The Handle makes the private subobject accessible for use in OM function calls.

- OM_EXCLUDE_DESCRIPTORS

  The function does not return any descriptors, nor does it return a value in the Copy argument. The value of the Total Number argument gives the number of descriptors which would have otherwise been returned.

- OM_NO_EXCLUSIONS

  The copy contains descriptors and values for all attributes in the original.

  If you specify multiple exclusions, the Service applies the exclusions in the order in which they occur in the above list. If a portion of the object disappears after the Service applies an exclusion, the Service applies no further exclusions to that portion.

Included Types

The types of attributes that are to be included in the copy, provided they appear in the original. This argument must be present if you select the OM_EXCLUDE_ALL_BUT_THESE_TYPES exclusion, but must otherwise be set to null.

Local Strings

If you set this argument to OM_TRUE, you indicate to the Service that it must translate all String(*) values included in the Copy into the local character set representation. This translation may cause the loss of some information.

Initial Value

The position within each attribute of the first value to be included in the copy. This argument must be present if you select the OM_EXCLUDE_ALL_BUT_THESE_VALUES exclusion, but must otherwise be set to null. If the value of Initial Value is OM_ALL_VALUES, or if it exceeds the number of values present in an attribute, the Service takes Initial Value to be equal to the number of values present in the attribute.

Limiting Value

The position in each attribute that is one element beyond the position of the last value included in the copy. This argument must be present if you select the OM_EXCLUDE_ALL_BUT_THESE_VALUES exclusion, but must otherwise be set to null. If the value of Limiting Value is less than that of Initial Value, the Service does not put any values in the copy. If the value of Limiting Value is OM_ALL_VALUES, or if it exceeds the number of values present in an attribute, then the Service takes Limiting Value to be equal to the number of values present in the attribute.

Copy

An exact but independent copy of the original. The Service returns this argument if both the following conditions are true:

- The Return Code of the function is OM_SUCCESS
- You do not specify the OM_EXCLUDE_DESCRIPTORS exclusion

You do not have to allocate any space to the copy. If you alter any portion of this space, you may affect the behaviour of the Service.

Total Number

The number of attribute descriptors in the copy. This does not include descriptors in any subobjects of the copy. If you specify the OM_EXCLUDE_DESCRIPTORS exclusion, then there is no copy. Therefore, the value in Total Number is the number of descriptors that the Service would return if you did not specify OM_EXCLUDE_DESCRIPTORS (applying any other exclusions that you specified). Total Number excludes the special descriptor that signals the end of a public object, OM_NULL_DESCRIPTOR.

Description

This function creates a new public object, the copy, which is an exact but independent copy of an existing private object, the original.

When using this function, you can request certain exclusions, each of which reduces the copy to a portion of the original.

When this function is used with the X.400 API, one exclusion is requested implicitly. For each attribute value in the original that is a string exceeding 1024 bytes in length, the copy includes a descriptor that omits the elements, but specifies the length of the string. In this case, the following applies:

- The syntax of the descriptor has its Long-String bit set.
- The value of the descriptor is a string whose elements component is set to OM_ELEMENTS_UNSPECIFIED, but whose length component *does* specify the correct length.

  Note that if you are using the OM API with the X.400 API, you can read long strings using *om_read*.

  When used with the X.500 API, there is no limit to the length of string returned by *om_get*, and so this exclusion is not requested implicitly.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network

OM_NO_SUCH_EXCLUSION	There is an undefined exclusion identifier
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_TYPE	There is an undefined type identifier
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code
OM_WRONG_VALUE_SYNTAX	There is an attribute value with an illegal syntax
OM_WRONG_VALUE_TYPE	There is an attribute value with an illegal type

Examples

The following example shows *om_get* being used to make a public copy of an object of the X.400 API MH class Local Per-recipient NDR (*ndr*) containing the attribute MH_T_TEMPORARY.

```
OM_private_object ndr;
OM_public_object  ndr_copy;
OM_value_position total_number;
OM_type
 temporary[] = {MH_T_TEMPORARY, OM_NO_MORE_TYPES};

     result = om_get (ndr,
                              /* object to be copied */
                      OM_EXCLUDE_ALL_BUT_THESE_TYPES,
                              /* include attributes of specified types */
                      temporary,
                              /* type to be included */
                      OM_FALSE,0,0,
                              /* no translation into local char set */
                      ndr_copy,
                              /* the copy */
                      &total_number);
                              /* number of attributes copied */
```

The example shows the use of the exclusion *exclude-all-but-these-types*. The public copy (*ndr_copy*) includes a descriptor representing the attribute MH_T_TEMPORARY (*temporary*), and its single value.

The following example shows how to use the Get function to work down a nested structure, getting one level at a time. The structure is a Result object returned from a call to the X.500 API DS Read function. Note that this code example uses the dsX_trace_object function, which is not available in Version 1.0 of the MAILbus 400 API.

```
/* declare an OM-type-list structure and variables to hold pointers to the
   entry, DS_object and RDNS subobjects:
   */
OM_integer        desc_count;
OM_object         read_result;
OM_type           included_types[2];
OM_public_object  spub_entry;
OM_public_object  spub_DS_object;
OM_public_object  spub_RDNS;
/* and set up the OM attributes you want to get first: */
          included_types[0] = DS_ENTRY;
          included_types[1] = OM_NO_MORE_TYPES;


/* now get only a pointer to the first subobject, the entry */
  om_status = om_get(read_result,
     OM_EXCLUDE_ALL_BUT_THESE_TYPES+OM_EXCLUDE_SUBOBJECTS,
                    included_types, OM_FALSE, 0, OM_ALL_VALUES,
                    &spub_entry, &desc_count);


/* the object spub_entry now contains only the
   OM-descriptor for an entry-information object */
dsX_trace_object(spub_entry);
* Now use OM_get() again to extract the DN of the object */
         included_types[0] = DS_OBJECT_NAME;
         om_status = om_get(spub_entry->value.object.object,
                     OM_EXCLUDE_ALL_BUT_THESE_TYPES+OM_EXCLUDE_SUBOBJECTS,
                    included_types, OM_FALSE, 0, OM_ALL_VALUES,
                    &spub_DS_object, &desc_count);
dsX_trace_object(spub_DS_object);


/* Next, use OM_get() again to extract the RDNS */
         included_types[0] = DS_RDNS;
         om_status = om_get(spub_DS_object->value.object.object,
                     OM_EXCLUDE_ALL_BUT_THESE_TYPES+OM_EXCLUDE_SUBOBJECTS,
                    included_types, OM_FALSE, 0, OM_ALL_VALUES,
                    &spub_RDNS, &desc_count);


dsX_trace_object(spub_RDNS);
/* Now loop around each RDN, extract a pointer to the AVAS
   and then extract the attribute type and value
   */
   ...
/* When finished, remember to delete all the objects you have used */
```

om_instance(3xom)

Name

om_instance - Determines whether an object is an instance of a particular class or of one of its subclasses.

Syntax

OM_return_code om_instance*(subject, class, instance)*

Argument	Data Type	Access
subject	OM_object	read
class	OM_object_identifier	read
instance	OM_boolean	write
return_code	OM_return_code	

C Binding

OM_return_code om_instance*(subject, class, instance)*

OM_object subject,
OM_object_identifier class,
OM_boolean #instance

Arguments

Subject

The object whose class you want to verify.

Class

The class against which you want to verify the subject.

Instance

The Service sets this argument to OM_TRUE if the subject is an instance of the class you specified, and OM_FALSE if the subject is not. The Service returns this argument if the Return Code of the function is OM_SUCCESS.

Description

This function enables you to determine whether an object is an instance of a specified class or of any of the subclasses of that class.

Note that it is possible to determine an object's class by using programming constructs to inspect the object, if it is public, or by using *om_get*, if it is private. The advantage of the Instance function is that it indicates whether the object is an instance of the specified class, even when it is also a subclass of the specified class.

Return Values

OM_SUCCESS	The function has completed its task successfully

OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_CLASS	There is an undefined class identifier
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_SYNTAX	There is an undefined syntax identifier.
OM_NOT_THE_SERVICES	An object is Client generated rather than Service generated or private.
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code

Examples

The following example shows how to check a private object, `an_object`, to see if it is an instance of the X.400 Message Handling class Submitted Message RD. `inst_true` is the return argument. The function returns OM_TRUE in the parameter if the object is an instance of the class Submitted Message RD, or its subclass Message RD.

```
OM_private_object   sub_mess_rd;
OM_return_code      result;
OM_boolean          inst;
     result = om_instance (an_object,
                           /* object to be checked */
                           MH_C_SUBMITTED_MESSAGE_RD,
                           /* class to be checked against */
                           &inst);
                           /* result */
     if (inst == OM_TRUE)
              /* the object is an instance of specified class */
     else
              /* the object is not an instance of specified class */
```

om_put(3xom)

Name

om_put - Places copies of the attribute values of a private or public object into a private object.

Syntax

OM_return_code om_put*(destination, modification, source, included_types, initial_value, limiting_value)*

Argument	Data Type	Access
destination	OM_private_object	read
modification	OM_modification	read
source	OM_object	read
included_types	OM_type_list	read
initial_value	OM_value_position	read
limiting_value	OM_value_position	read
return_code	OM_return_code	

C Binding

OM_return_code *om_put (destination, modification, source, included_types, initial_value, limiting_value)*

OM_private_object destination,
OM_modification modification,
OM_object source,
OM_type_list included_types,
OM_value_position initial_value,
OM_value_position limiting_value

Arguments

Destination

The object into which you want to put attribute values. This function does not affect the class of the destination.

Modification

A list of modifications to the attributes selected for copying. The modifications you request determine how the function modifies the destination object with the attributes, that is, where it puts them.

The Modification argument can have one of the following values:

- OM_INSERT_AT_BEGINNING

  The Service inserts the source values before all existing destination values. This does not affect the existing destination values.

- OM_INSERT_AT_CERTAIN_POINT

  The Service inserts the source values before the value at a specified position in the destination attribute. This does not affect the existing destination values.

- OM_INSERT_AT_END

  The Service inserts the source values after all existing destination values. This does not affect the existing destination values.

- OM_REPLACE_ALL

  The Service replaces any destination values with the source values, and discards the original destination values.

- • OM_REPLACE_CERTAIN_VALUES

  The Service replaces the values at specified positions in the destination attribute with values from the source. The Service discards the original destination attribute values at those positions.

Source

The object from which you want to copy attribute values. This function ignores the class of the source.

Included Types

The types of attributes that should be copied to the destination, if they appear in the source. If you do not specify a value for this argument, the Service copies all attributes from the source to the destination.

Initial Value

You need to pass a value for this argument when you select the OM_INSERT_AT_CERTAIN_POINT modification or the OM_REPLACE_CERTAIN_VALUES modification. The following table shows what the Initial Value argument represents in each of these cases.

Modification

Meaning of the Initial Value Argument

insert-at-certain-points

The position within each destination attribute at which the Service should insert source values

replace-certain-values

The first value that the Service should replace

If you give this argument a value that is greater than the number of values present in a destination attribute, or if you use the value OM_ALL_VALUES, the Service takes Initial Value to be equal to the number of values present in the destination attribute.

Limiting Value

You need to pass a value for this argument when you select the OM_REPLACE_CERTAIN_VALUES modification. The argument specifies the position within each destination attribute that is one beyond that of the last value to be replaced. The value of Limiting Value must be greater than that of Initial Value.

If you give this argument a value that is greater than the number of values present in a destination attribute, or if you use the value OM_ALL_VALUES, the Service takes Limiting Value to be equal to the number of values present in the destination attribute.

Description

This function places in one private object, the destination, copies of the attribute values of another object, the source. The source can be a public or private object.

You must specify that the Service does one of the following:

- Replace all the values in the destination with values from the source.
- Replace specified values in the destination with values from the source.
- Insert values from the source in a particular position in the destination.

Only use *om_put* to copy attributes from the source that occur in the definition of the class to which the destination belongs.

The Service first converts all string values that are in the local representation into the nonlocal representation for that syntax.

There is no attribute of type C in the source object. The destination object therefore contains no attributes of this type, even though type C is specified in the Included Types argument.

The destination object contains an attribute, D, which is not affected by *om_put*.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_DECLINED	The function does not apply to the object to which it is addressed
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_CLASS	There is an undefined class identifier
OM_NO_SUCH_MODIFICATION	There is an undefined modification identifier
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_SYNTAX	There is an undefined syntax identifier
OM_NO_SUCH_TYPE	There is an undefined type identifier
OM_NOT_CONCRETE	There is an abstract class where there should be a concrete class
OM_NOT_PRESENT	An expected attribute value is missing

OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code
OM_TOO_MANY_VALUES	An implementation limit prevents the addition to an object of another attribute value
OM_VALUES_NOT_ADJACENT	The descriptors for the values of a particular attribute are not adjacent
OM_WRONG_VALUE_LENGTH	There is an attribute with a value that violates the value length constraints in force
OM_WRONG_VALUE_MAKEUP	There is an attribute with a value that violates a constraint of its syntax
OM_WRONG_VALUE_NUMBER	There is an attribute with a value that violates the value number constraints in force
OM_WRONG_VALUE_POSITION	The position specified for an attribute value is invalid
OM_WRONG_VALUE_SYNTAX	There is an attribute value with an illegal syntax
OM_WRONG_VALUE_TYPE	There is an attribute value with an illegal type

Examples

The following example shows how to put the values from a public object into the private object.

```
result = om_put (modification,
                              /* destination object */
                       OM_REPLACE_ALL,
                              /* type of modification */
                       pub_mod,
                              /* source of values to be put */
                       0,0,0);
                              /* include all attributes, all positions */
```

om_read(3xom)

Name

om_read - Reads a segment of a string from a private object.

Syntax

OM_return_code om_read*(subject, type, value_position, local_string, string_offset, elements)*

Argument	Data Type	Access
subject	OM_private_object	read
type	OM_type	read
value_position	OM_value_position	read
local_string	OM_boolean	read
string_offset	OM_string_length	read-write
elements	OM_string	write
return_code	OM_return_code	

C Binding

OM_return_code om_read*(subject, type, value_position, local_string, string_offset, elements)*

OM_private_object subject,
OM_type type,
OM_value_position value_position,
OM_boolean local_string,
OM_string_length #string_offset,
OM_string #elements

Arguments

Subject

The private object from which you want to read the segment.

Type

The type of the attribute containing the value that youwant to read.

Value Position

The position in a multivalued attribute of the value that you want to read.

Local String

If you set this argument to OM_TRUE, the Service translates the attribute segment into the local character set. This translation may result in the loss of some information.

String Offset

If provided by the Client, this argument denotes the position within the attribute value of the first element that you want to read. If you give this argument a value that exceeds the number of elements present in the attribute value, the Service takes the argument to be equal to the number of elements present in the attribute value.

If returned by the Service, this argument denotes the position of the next segment within the attribute value, as an offset in octets. If the segment just read was the last in the string, then this argument is set to zero. The result is present only if the Return Code result is OM_SUCCESS.

The value indicating the next position can be specified in a subsequent call as the position to start from, enabling sequential reading of the segments in a string value.

Elements

A space into which the Service returns the segment of the attribute value that you want to read. This argument is a string with two components, Elements and Length.

The following shows the initial values that you should give to these components:

String Component	Initial Value
Elements	Pointer to a buffer
Length	The number of octets required to contain the segment that the function returns

You must make sure that the buffer is big enough to hold the number of octets.

The Service modifies the Elements argument. Each element that the function returns becomes an element in the string. The string's length becomes the number of octets actually required to hold the segment read (which may be smaller than the length initially specified.)

If the value of Local Strings is OM_TRUE, the final length of the string may not be the same as the initial length of the string. This depends on the characteristics of the translation into the local character set.

Description

The function enables you to read a long string without requiring the Service to place a copy of the entire string in memory.

Return Values

OM_SUCCESS	The function has completed its task successfully

OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_TYPE	There is an undefined type identifier
OM_NOT_PRESENT	An expected attribute value is missing
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code
OM_WRONG_VALUE_SYNTAX	There is an attribute value with an illegal syntax

Examples

The following example shows the reading of a string value from an object of the IM class IA5 Text Body Part (`body_part`). The Service will return 0 when there is no more text left to read.

```
OM_private_object   body_part;
OM_return code      result;
OM_string           message;
OM_string_length    offset;
char                read_buffer[1024];
message.length = 1024
message.elements = read_buffer;
offset = 0;
     result = om_read (body_part,
                             /* object containing value to be read */
                       IM_TEXT,
                             /* attribute from which value is to be read */
                       0,
                             /* position of value to read from */
                       OM_FALSE,
                             /* no translation into local char set */
                       &offset,
                             /* string offset of segment to be read */
                       &message);
                             /* the string read from the value */
```

The type of the attribute read is `IM_TEXT`. The first element of the first value in this attribute is read into `message`.

om_remove(3xom)

Name

om_remove - Removes and discards specified values of an attribute of a private object.

Syntax

OM_return_code om_remove*(subject, type, initial_value, limiting_value)*

Argument	Data Type	Access
subject	OM_private_object	read
type	OM_type	read
initial_value	OM_value_position	read
limiting_value	OM_value_position	read
return_code	OM_return_code	

C Binding

OM_return_code om_remove*(subject, type, initial_value, limiting_value)*
OM_private_object subject,

OM_type type,
OM_value_position initial_value,
OM_value_position limiting_value

Arguments

Subject

The private object from which you want to remove attribute values. The function does not affect the class of the subject.

Type

The type of the attribute from which you want to remove values. The type must not be OM_CLASS.

Initial Value

The position within the attribute of the first value to be removed.

If the value of Initial Value is OM_ALL_VALUES, or if it exceeds the number of values present in the attribute, the Service takes this argument to be equal to the number of values present in the attribute.

Limiting Value

The position within the attribute one beyond that of the last value to be removed. If this argument is less than theInitial Value argument, no values are removed.

If the value of Limiting Value is OM_ALL_VALUES, or if the value exceeds the number of values present in an attribute, the Service takes this argument to be equal to thenumber of values present in the attribute.

Description

This function removes and discards particular values of an attribute of a private object, the subject. If no values remain in an attribute after removal of the values you specify, the Service removes the attribute. If one of the values you specify is a subobject, the Service removes that value, and then applies *om_delete* to make the subobject inaccessible.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_DECLINED	The function does not apply to the object to which it is addressed
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_TYPE	There is an undefined type identifier
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code

Examples

The first example shows a single value being removed from a private object of MH class Message (`message`). `MH_T_LATEST_DELIVERY_TIME` is the attribute type whose value is removed. This attribute has one value. The function therefore removes the value and deletes the attribute.

The second example shows a range of values being removed from a private object of the X.500 class Entry-Information-Selection (`select_info`). `DS_ATTRIBUTES_SELECTED` is the attribute type whose values are removed. If this attribute has five values (positions 0 to 4), then this function removes the second, third and fourth values.

```
OM_private_object    message;
OM_return_code       result;
     result = om_remove (message,
                              /* object from which to remove value */
                         MH_T_LATEST_DELIVERY_TIME,
                              /* attribute to remove */
                         0,1));
                              /* position of single value to remove */

OM_private_object    select_info;
OM_return_code       result;
     result = om_remove (select_info,
                              /* object from which to remove value */
                         DS_ATTRIBUTES_SELECTED,
                              /* attribute to remove */
                         1,4));
                              /* range of values to remove */
```

om_write(3xom)

Name

om_write - Writes a segment of a string to an attribute in a private object.

Syntax

OM_return_code om_write*(subject, type, value_position, syntax, string_offset, elements)*

Argument	Data Type	Access
subject	OM_private_object	read s
type	OM_type	read
value_position	OM_value_position	read
syntax	OM_syntax	read
string_offset	OM_string_length	read-write
elements	OM_string	read
return_code	OM_return_code	

C Binding

OM_return_code om_write*(subject, type, value_position, syntax, string_offset, elements)*

OM_private_object subject,
OM_type type,
OM_value_position value_position,
OM_syntax syntax,
OM_string_length #string_offset,
OM_string elements

Arguments

Subject

The object into which you want to write the string segment.

Type

The type of the attribute to which you want to write the string segment.

Value Position

In a multi-valued attribute, the position of the value in which you want to place the string. This argument must have a positive value, and it must not exceed the number of values present in the attribute. If it equals the number of values present, the Service inserts the segment at the end of the attribute as a new value.

Syntax

If you are writing a new value to an attribute, identify in this argument the syntax that you want the new value to have. It must be a permissible syntax for the attribute to which you are writing. To check that the syntax is permissible, consult the definition of the class of which the subject is an instance.

If you are overwriting or amending a value that is already present in the subject, the Service preserves the syntax of that value, so you can supply a null value.

String Offset

If supplied by the Client, this argument denotes the position, *p*, within the attribute value at which you want the first segment written. The position is specified as an offset in octets relative to the start of the string value.

If this argument has a value greater than the number of elements in the attribute value, the Service takes the argument to be equal to that number.

If returned by the Service, this argument denotes the position of the end of the last segment written. The position is specified as an offset in octets relative to the start of the string value.

The value returned by the Service as the position of the end of the last segment written can be specified as the position from which to start writing the next segment. This enables you to write segments sequentially.

Elements

The string segment that you want to write to the attribute, *n* elements in number. Copies of these elements occupy the positions, within the value, in the interval between *p* and *(p) + n. The function discards any elements already in* or beyond these positions.

The elements are bits, octets, or characters, depending on the nature of the string.

Description

This function writes a segment of an attribute value in a private object, the subject. The segment that the Service writes becomes the last segment in the attribute value. The function discards any segments in the attribute value whose offsets are equal to or greater than the offset specified in String Offset.

If the segment that the Service writes is in the local representation, the Service converts it to the nonlocal representation. This can result in loss of information and may result in a different number of elements than that specified.

Return Values

OM_SUCCESS	The function has completed its task successfully
OM_FUNCTION_DECLINED	The function does not apply to the object to which it is addressed
OM_FUNCTION_INTERRUPTED	The function was aborted by external intervention
OM_MEMORY_INSUFFICIENT	There is not enough memory to complete the function
OM_NETWORK_ERROR	The Service cannot use the underlying network
OM_NO_SUCH_OBJECT	You have specified a nonexistent object, or an invalid Handle for an object
OM_NO_SUCH_SYNTAX	There is an undefined syntax identifier
OM_NO_SUCH_TYPE	There is an undefined type identifier
OM_NOT_PRESENT	An expected attribute value is missing
OM_NOT_PRIVATE	There is a public object where there should be a private object
OM_PERMANENT_ERROR	The Service encountered a permanent problem for which there is no defined error code
OM_POINTER_INVALID	An invalid pointer was supplied as a function argument
OM_SYSTEM_ERROR	The Service cannot use the operating system
OM_TEMPORARY_ERROR	The Service encountered a temporary problem for which there is no defined error code
OM_WRONG_VALUE_LENGTH	There is an attribute with a value that violates the value length constraints in force

OM_WRONG_VALUE_MAKEUP	There is an attribute with a value that violates a constraint of its syntax
OM_WRONG_VALUE_POSITION	The usage of value position(s) identified in the argument(s) of a function is invalid
OM_WRONG_VALUE_SYNTAX	There is an attribute value with an illegal syntax

Examples

The following example shows a string segment being written to an object of the Interpersonal Messaging (IM) class IA5 Text Body Part (`body_part`).

```
OM_return_code result;
OM_value_position position;
input_string = OM_STRING ("Text Body");
position = 0;
     result = om_write (body_part,
                              /* object containing string segment */
                        IM_TEXT,
                              /* attribute containing string segment */
                        0,
                              /* position of value to write to */
                        OM_S_IA5_STRING,
                              /* syntax of value */
                        &position,
                              /* starting position for current/next string */

                        input_string);
                              /* string segment to be written */
```

In this example the type of attribute being written to is `IM_TEXT`; the position of the value being written to is 0; the syntax of the value is `OM_S_IA5_STRING`.

CHAPTER 5

Command Reference Pages

This chapter provides revised reference pages necessary for using Gradient DCE for Tru64 UNIX.

- cdscache.8dce
- clearinghouse.8dce
- directory.8dce
- rand.8sec
- ldap_addcell.8dce
- gdad.8cds
- dtsd.8dts

cdscache(8dce)

Name

cdscache -A **dcecp** object for managing a local CDS cache

Synopsis

cdscache create *server_name* **-binding** *server_binding*
cdscache delete *server_name*
cdscache discard [-entry *entry_name***]** [**-replica** *replica_name*]
cdscache dump
cdscache help [*operation* | **-verbose**]
cdscache operations
cdscache show *server_name* {**-server** | **-clearinghouse**}

Arguments

operation

The name of the **cdscache** operation that displays help information.

server_name

The single simple name of the cached server machine. A “simple name’ is not a cell-relative name (such as **/.:/hosts/pelican**). Some examples of simple names are **pelican** and **hosts/pelican**.

option

A specification to clarify the target of the operation (**-entry, -replica**).

Description

The **cdscache** object represents the Cell Directory Service (CDS) cache on the local node. The CDS cache contains information about servers and clearinghouses known to the local machine, and also contains user data about CDS entries that have been read. The **create** and **delete** operations apply only to the server information. The **show** and **dump** operations can display additional information. The **discard** operation can selectively eliminate cache entries on any client (given the appropriate permissions).

Errors

A representative list of errors that might be returned is not shown here. Refer to the *DCE Problem Determination Guide* for complete descriptions of all error messages.

Operations

cdscache create

Creates knowledge of a server in the local client's cache. The syntax is:

cdscache create *server_name* **-binding** *server_binding*

Options

-binding *server_binding*

The required **-binding** option lets you specify the binding information for a CDS server. This option takes a *server_binding* argument, which is the protocol sequence and network address of the server node. The string format is as follows:

protocol-sequence:network-address

The **dcecp** format is as follows:

{*protocol-sequence network-address***}**

A *protocol-sequence* is a character string identifying the network protocols used to establish a relationship between a client and server. Protocol sequences have a specific format that depends on the network address that is supplied in the binding; for example **ncacn_ip_tcp** (for connection-based protocol) or **ncadg_ip_udp** (for datagram protocol) The *network-address* is a string representing the network address of the server node.

The **create** operation creates knowledge of a server in the local client's cache. The *server_name* argument is the simple name of a cached server. (An example of a simple name would be **pelican**, as opposed to a cell-relative name like **/.:/hosts/pelican**.) This command is typically used to provide configuration information manually to a client that cannot configure itself

automatically. Providing configuration information manually may be necessary, for instance, to provide the client with addressing information about a server across a WAN. Once the client knows about one server, it can find other servers through referrals. This operation returns an empty string on success.

Privileges Required

You must have **w** (**write**) permission to the client system, **/.:/hosts/***hostname***/cds-clerk**.

Examples

The following command creates knowledge of the server **pelican** in the localclient's cache:

```
dcecp> cdscache create pelican -binding ncacn_ip_tcp:16.20.15.25
dcecp>
```

cdscache delete

Removes knowledge of a server that you had specifically created from the local client's cache. The syntax is:

cdscache delete *server_name*

The **delete** operation removes knowledge of a server that was specifically created from the local client's cache. The required *server_name* argument is the simple name of a cached server. (An example of a simple name would be **pelican**,as opposed to a cell-relative name like **/.:/hosts/pelican**.) You can delete only servers that you have specifically created with the **cdscache create** command. This operation returns an empty string on success.

Privileges Required

You must have **w** (**write**) permission to the client system, **/.:/hosts/***hostname***/cds-clerk**.

Examples

The following command removes knowledge of the server **gumby** from the client cache:

```
dcecp> cdscache delete gumby
dcecp>
```

cdscache discard

Discards the contents of the client cache. The syntax is:

cdscache discard [-entry *entry_name***]** [**-replica** *replica_name*]

Options

-entry *entry_name*

Specifies the name of an entry in the clerk cache.

-replica *replica_name*

Specifies the clearinghouse name of a replica pointer.

Description

The **cdscache discard** operation eliminates data structures, frees memory, and shrinks the size of the client cache on the host specified by *host_name*. If *host_name* is not specified, the operation defaults to the cache that resides on the local host. Only a single host name can be specified. If both **-entry** and **-replica** options are specified, the command removes the specified replica pointer from a cached directory entry.

If a clearinghouse becomes unreachable, attempting to contact it to look up a replica pointer can result in a very time-consuming failure. If you use the option **-replica**, all replicas referencing the clearinghouse are removed from all cache directory entries. In this way, all references to an unreachable clearinghouse can be removed. If successful, this operation returns a null string.

The operation, cdscache discard with the options **-entry** or **-replica**, does the following:

- Deletes only the cached structures specified by the -entry and -replica options.
- Does not stop and restart CDS.
- Does not stop and restart DCE.

Privileges Required

If it is a local operation you must have superuser (root) privileges on the client system. No DCE permissions are required.

If it is a remote operation you must have permissions on the **srvrconf** directory of the selected host:

- **I** permission on **/.:/hosts/<HOSTS>/config/srvrconf**
- **x** permission on the created server object (that is, **x** permission in the initial object ACL of **/.:/<HOSTS>/config/srvrconf)**
- **d** permission on the created server object (that is, **d** permission in the initial object ACL of **/.:/<HOSTS>/config/srvrconf)**

Examples

The following command discards the entry, job_300 at the clearinghouse named gloria_ch from the client cache on the local host:

```
dcecp> cdscache discard -entry /.:/ job_300 -replica /.:/gloria_ch
dcecp>
```

cdscache dump

Displays the entire contents of the client cache. The syntax is:

cdscache dump

The **cdscache dump** operation displays the contents of the client cache on the screen. Use this command when solving CDS problems.

Privileges Required

You must have superuser (root) privileges on the client system. No DCE permissions are required.

Examples

The following command displays the contents of the client cache on the screen (the output is not shown in the example):

```
dcecp> > cdscache dump
dcecp> >
```

cdscache help

Returns help information about the **cdscache** object and its operations. The syntax is:

cdscache help [*operation* | **-verbose**]

Options

-verbose

Displays information about the **cdscache** object.

Description

Used without an argument or option, the **cdscache help** command returns brief information about each **cdscache** operation. The optional *operation* argument is the name of an operation about which you want detailed information. Alternatively, you can use the **-verbose** option for more detailed information about the **cdscache** object itself.

Privileges Required

No special privileges are needed to use the **cdscache help** command.

Examples

```
dcecp> cdscache help
create     Adds information about named server in local cds cache.
delete     Removes information about named server in local cds cache.
discard    Discards all cdsadv (CDS advertiser) cache information on the specified host.
dump       Dumps all information from local cds cache.
show       Shows information stored in cds cache.
help       Prints a summary of command-line options.
operations Returns a list of valid operations for this command.
dcecp>
```

cdscache operations

Returns a list of the operations supported by the **cdscache** object. The syntax is as follows:

cdscache operations

The list of available operations is in alphabetical order except for **help** and **operations**, which are listed last.

Privileges Required

No special privileges are needed to use the **cdscache operations** command.

Examples

```
dcecp> cdscache operations
create delete discard dump show help operations
dcecp>
```

cdscache show

Returns information about clearinghouses or servers stored in the cache. The syntax is as follows:

cdscache show *server_name* {**-server** | **-clearinghouse**}

Options

-clearinghouse

This option displays all the names and values of the attributes in the specified cached clearinghouse. The following are valid attributes:

- **Creation Time**

  Specifies the time at which this clearinghouse was added to the cache.

- **Miscellaneous Operations**

  Specifies the number of operations other than read and write (that is, skulks, new epochs, and so on) performed by this client on the cached clearinghouse.

- **Read Operations**

  Specifies the number of lookup operations of any sort performed by the client on the cached clearinghouse.

- **Towers**

  Specifies the protocol sequence and network address of the server that maintains the cached clearinghouse.

- **Write Operations**

  Specifies the number of write operations performed by this client on the cached clearinghouse.

-server

This option displays address information of a server in the local client's cache. The following attributes are valid:

- **Name**

The directory cell name

- **Towers**

  The protocol sequence and network address of the server node

The **show** operation displays information about clearinghouses or servers stored in the cache. The required *server_name* argument is the simple name of a server or a CDS name of a clearinghouse for which you want to display information. You must use one of the **-clearinghouse** or **-server** options to select the information you want to display.

Privileges Required

You must have **r** (**read**) permission to the CDS client.

Examples

The following command displays all attributes of the cached clearinghouse **/.:/claire_ch**:

```
dcecp> cdscache show /.:/claire_ch -clearinghouse
{CH_Name /.../blue.cell.osf.org/claire_ch}
{Created 1994-10-07-11:41:23.131}
{Others 458}
{Reads 150221}
{Tower {ncacn_ip_tcp 130.105.4.158}}
{Tower {ncadg_ip_udp 130.105.4.158}}
{Writes 162}
dcecp>
```

The following command displays all attributes of the cached server **mysystem**:

```
dcecp> cdscache show mysystem -server
{CH_Name /.../terrapin_cell.osf.org/mysystem_ch}
{Tower {ncacn_ip_tcp 130.105.5.16}}
{Tower {ncadg_ip_udp 130.105.5.16}}
```

Related Information

Commands: **clearinghouse(8dce)**, **dcecp(8dce)**, **directory(8dce)**, **link(8dce)**, **object(8dce)**. **define_cached-server(8cds)**. **dump-clerk_cache(8cds)**. **show_cached-clearinghouse(8cds)**. **show_cached-server(8cds)**.

clearinghouse(8dce)

Name

clearinghouse -A **dcecp** object that manages a clearinghouse in CDS

Synopsis

clearinghouse catalog [*cell_name***]** [-*simplename*]

clearinghouse create *clearinghouse_name_list*

clearinghouse delete *clearinghouse_name_list*

clearinghouse disable *clearinghouse_name_list*

clearinghouse initiate *clearinghouse_name_list* **-checkpoint**

clearinghouse help [*operation* | **-verbose**]

clearinghouse modify *object_name_list* {[**-add** *attribute_list* [**-single**]] [**-remove** *attribute_list* [**-types**]] **-change** *attribute_list*}

clearinghouse operations

clearinghouse repair *clearinghouse_name_list* **-timestamps**

clearinghouse show *clearinghouse_name_list* **[-counters | -attributes | -all] [-schema]**

clearinghouse verify *clearinghouse_name_list*

Arguments

cell_name

This optional argument is the name of a single cell. The name must be a fully qualified cell name as shown in the following:

/.../their_cell.goodco.com

clearinghouse_name_list

A list of one or more names of clearinghouses on which to operate.

operation

The name of one specific **clearinghouse** operation (subcommand) about which you want to see help information.

Description

The **clearinghouse** object represents Cell Directory Service (CDS) clearinghouses. Clearinghouses are databases located on CDS server machines that store data (directories, objects, and links) in CDS. On the server machines are files that contain the actual clearinghouse data. Clearinghouses are also represented in the CDS namespace by an entry that contains information about the clearinghouse.

You must run the **create** command on the host where you want to create the new clearinghouse and the **delete**, **disable**, **initiate**, **repair**, and **verify** commands on the host where the clearinghouse to be operated on resides.

If the **_s(cds)** convenience variable is set, it is treated as the name of a clearinghouse to contact for this operation. This is the only clearinghouse that will be contacted in an attempt to complete the operation. These commands do not set the value of this variable after completion.

Attributes

The following are the CDS-defined attributes that may be present in CDS clearinghouse objects:

CDS_AllUpTo

Indicates the date and time the clearinghouse object has been updated to reflect the **CDS_CHDirectories** attribute.

CDS_CHDirectories

Specifies the full name and Universal Unique Identifier (UUID) of every directory that has a replica in this clearinghouse.

CDS_CHLastAddress

Specifies the current reported network address of the clearinghouse.

CDS_CHName

Specifies the full name of the clearinghouse.

CDS_CHState

Specifies the state of the clearinghouse. The state **on** indicates the clearinghouse is running and available.

CDS_CTS

Specifies the creation timestamp (CTS) of the clearinghouse.

CDS_DirectoryVersion

Specifies the current version of the directory in the clearinghouse in which the directory was created.

CDS_NSCellname

Specifies the name of the cell in which the clearinghouse resides.

CDS_ObjectUUID

Specifies the UUID of the clearinghouse. This read-only attribute is set by the system when the clearinghouse object is created.

CDS_ReplicaVersion

Specifies the current version of the replica in which the directory was created.

CDS_UpgradeTo

A single-valued attribute used to control the upgrading of a directory from one version of CDS to another. By modifying this attribute, the process of upgrading a directory to a newer version of CDS may be initiated.

CDS_UTS

Specifies the timestamp of the most recent update to an attribute of the clearinghouse.

Counters

The clearinghouse counters are as follows:

badchs

Specifies the number of times the **clearinghouse entry missing** event was generated.

baddata

Specifies the number of times that the **data corruption** event was generated.

badskulks

Specifies the number of times that a skulk of a directory, initiated from this clearinghouse, failed to complete—usually because one of the replicas in the replica set was unreachable.

badupgrades

Specifies the number of times that upgrades failed.

disables

Specifies the number of times that the clearinghouse was disabled since it was last started.

enables

Specifies the number of times that the clearinghouse was enabled since it was last started.

partials

Specifies the number of requests directed to this clearinghouse that resulted in the return of a partial answer instead of satisfying the client's request.

reads

Specifies the number of read operations directed to this clearinghouse.

rootlosts

Specifies the number of times the **root lost** event was generated.

writes

Specifies the number of write operations directed to this clearinghouse.

See the *OSF DCE Administration Guide* for more information about attributes and counters.

Operations

clearinghouse catalog

Returns a list of the names of all clearinghouses in a cell. The syntax is as follows:

clearinghouse catalog [*cell_name***] [-simplename]**

The **clearinghouse catalog** command returns a list of the names of all clearinghouses in a cell. If you do not specify the optional argument, the cell name defaults to **/.:**.

Privileges Required

No special privileges are needed to use the **clearinghouse catalog** command.

Examples

```
dcecp> clearinghouse catalog
/.../dcecp.cell.osf.org/frick_ch
dcecp> clearinghouse catalog -simplename
frick_ch
```

clearinghouse create

Creates a new clearinghouse on the local machine. The syntax is as follows:

clearinghouse create *clearinghouse_name_list*

The **clearinghouse create** operation creates a new clearinghouse on the local machine. The *clearinghouse_name* argument is a list of one or more names of the clearinghouses you want to create. Clearinghouses should only be named in the root directory—that is, **/.:**. This operation also stores a read-only replica of the root directory in the new clearinghouse. The process that creates the new clearinghouse initiates a skulk of the root directory, so all replicas of the root should be reachable when you enter the **clearinghouse create** command. To ensure this, perform an immediate skulk of **/.:** prior to invoking the command, using the **directory synchronize /.:** command. The operation returns an empty string on success.

Privileges Required

You need **w** (**write**) permission to the server on which you intend to create the clearinghouse and **A** (**Admin**) permission to the cell root directory. The server principal needs **r** (**read**), **w** (**write**), and **A** (**Admin**) permission to the cell root directory.

Examples

The following command creates a clearinghouse named **/.:/Boston_CH** on the local server system:

```
dcecp > clearinghouse create /.:/Boston_CH
```

clearinghouse delete

Deletes the specified clearinghouse from the local machine. The syntax is as follows:

clearinghouse delete *clearinghouse_name_list*

The **clearinghouse delete** operation deletes the specified clearinghouse from the local server system. The *clearinghouse_name_list* argument is a list of one or more names of the clearinghouses you want to delete. Clearinghouses that contain master replicas of directories are not deleted (and also return errors). This command also automatically deletes all read-only replicas from the clearinghouse; however, you should delete all read-only replicas by hand (see **directory delete -replica**) before invoking this command since invoking many skulks will cause the command to execute more slowly. The command returns an empty string on success.

CDS does not permit you to delete a disabled (cleared) clearinghouse. Before you can delete a disabled (cleared) clearinghouse, you must recreate it using the **clearinghouse create** command.

Privileges Required

You must have **w** (**write**) and **d** (**delete**) permission to the clearinghouse and **A** (**Admin**) permission to all directories that store replicas in the clearinghouse. The server principal must have **d** (**delete**) permission to the associated clearinghouse object entry and **A** (**Admin**) permission to all directories that store replicas in the clearinghouse.

Examples

The following command deletes a clearinghouse named **/.:/Orion_CH** from the local server system:

```
dcecp > clearinghouse delete /.:/Orion_CH
```

clearinghouse disable

Removes knowledge of the specified clearinghouse from the local server's memory. The syntax is as follows:

clearinghouse disable *clearinghouse_name_list*

The **clearinghouse disable** operation removes knowledge of the specified clearinghouse from the local server's memory. The *clearinghouse_name_list* argument is a list of names of one or more clearinghouses you want to disable. Use this command when relocating a clearinghouse. This command removes the name of the prefix of the clearinghouse files from the **/opt/dcelocal/var/directory/cds/cds_files** file and notifies the local CDS server that the clearinghouse is disabled. The clearinghouse entry is not removed from the namespace, nor are the datafiles associated with the clearinghouse removed. The operation returns an empty string on success.

Privileges Required

You must have **w** (**write**) permission to the CDS server on which the clearinghouse resides.

Examples

The following command disables the clearinghouse **/.:/Paris2_CH** so that it can be moved to another server:

```
dcecp > clearinghouse disable /.:/Paris2_CH
```

clearinghouse help

Returns help information about the **clearinghouse** object and its operations. The syntax is as follows:

clearinghouse help [operation | **-verbose**]

Options

-verbose

Displays information about the **clearinghouse** object.

Used without an argument or option, the **clearinghouse help** command returns brief information about each **clearinghouse** operation.

The optional operation argument is the name of an operation about which you want detailed information.

Alternatively, you can use the **-verbose** option for more detailed information about the **clearinghouse** object itself.

Privileges Required

No special privileges are needed to use the **clearinghouse help** command.

Examples

```
dcecp>clearinghouse help
catalog             Returns the names of all clearinghouses in a cell.
create              Creates the named clearinghouse.
delete              Deletes the named clearinghouse.
disable             Disables the named clearinghouse.
initiate            Initiates an action on the named CDS clearinghouse.
modify              Add, remove or change an attribute in the named
                      clearinghouse.
repair              Repairs an aspect of the named CDS clearinghouse.
show                Returns the attributes of a clearinghouse.
verify              Verifies the consistency of the clearinghouse.
help                Prints a summary of command-line options.
operations          Returns a list of the valid operations for this command.
```

clearinghouse initiate

Initiates a defined action on the specified clearinghouse on the local machine. The syntax is

clearinghouse initiate *clearinghouse_name_list* **-checkpoint**

Options

-checkpoint

Forces the clearinghouse to checkpoint to disk.

The **clearinghouse initiate** operation initiates a defined action on the specified clearinghouse. The required *clearinghouse_name_list* argument is a list of one or more names of clearinghouses you want to initiate actions on. Currently, only a checkpoint action is available. This operation returns an empty string on success.

Privileges Required

You need write permission on the clearinghouse server and administer permission on the cell root directory. The server principal needs read, write, and administer permission on the cell root directory.

Examples

The following command initiates a checkpoint operation on the clearinghouse named **/.:/oddball_ch** on the local system.

```
dcecp > clearinghouse initiate /.:/oddball_ch -checkpoint
```

clearinghouse modify

Adds or removes attributes or changes attribute values for clearinghouse entries in the Cell Directory Service. This task is usually done through a client application.

The syntax is as follows:

clearinghouse modify *object_name_list* {[**-add** *attribute_list* [**-single**]] [**-remove** *attribute_list* [**-types**]] **-change** *attribute_list*}

Options

-add

Adds one or more new attributes to an clearinghouse entry.

-single

Optionally used with the **-add** option to specify that the attribute is single-valued.

-remove

Removes one or more attribute values from an attribute type of a clearinghouse entry. For instance, removing a value from an attri-bute with three values leaves the attribute with two values.

```
{{attribute_name attribute_value}... \\
{attribute_name attribute_value}}
```

To remove an attribute type as well as its values, use the **-types** option with the **-remove** option. For example:

**clearinghouse modify /.:/foo **
**-remove {RPC_CLASS RPC_CLASS_VERSION} **
-types

If an attribute is not present, an error is returned. Fixed CDS attribute types, such as the CDS creation Timestamp (CDS_CTS), cannot be removed.

-types

Can be used with the **-remove** option to remove the attribute type as well as its values. This option is invalid without the **-remove** option.

-change

Changes one attribute value to another for a clearinghouse entry. The value of this option is an attribute list. The existing value of each attribute named in the list is replaced by the new value given for each attribute attribute in the list. For multivalued attributes, all existing values are replaced by all the values listed for the attribute in the attribute list. If an attribute or value is not present, an error is returned.

The **clearinghouse modify** operation adds or removes attributes or changes attribute values for clearinghouse entries in the Cell Directory Service. This task is usually done through a client application. The required *clearinghouse_name* argument is the full CDS name of the clearinghouse entry to be modified.

Privilege Required

You must have **w** (**write**) permission to the clearinghouse entry.

Examples

To change the value of the sales_record attribute to region2 of an clearinghouse entry named /.:/Q1_records, follow these steps:

1 Read the **cds_attributes** file to check that the attribute sales_record is listed, as shown in the following display:

OID	LABEL	SYNTAX
1.3.22.1.3.66	sales_record	char

2 Enter the following command to assign the value region2 to the attributesales_record of an clearinghouse entry named **/.:/Q1_records**.

```
dcecp > clearinghouse modify /.:/Q1_records -add {sales_record region2}
dcecp  >
```

clearinghouse operations

Returns a list of the operations supported by the **clearinghouse** object. The syntax is as follows:

clearinghouse operations

The **clearinghouse operations** command takes no arguments and returns a list of the available operations for the **clearinghouse** object. The order of the elements is alphabetical with the exception that **help** and **operations** are listed last.

Privileges Required

No special privileges are needed to use the **clearinghouse operations** command.

Examples

```
dcecp> clearinghouse operations
catalog create delete disable initiate modify repair show verify help operations
```

clearinghouse repair

Repairs a specific problem on a specified clearinghouse on the local machine. The syntax is:

clearinghouse repair *clearinghouse_name_list* **-timestamps**

Options

-timestamps

Analyzes and repairs invalid timestamps found in a clearinghouse.

Use the **clearinghouse repair** operation to fix various problems that can occur in a clearinghouse. The required *clearinghouse_name_list* argument is a list of one or more names of clearinghouses you want to initiate repair actions on. Currently, only invalid timestamps can be repaired. This operation returns an empty string on success.

Privileges Required

You need write permission on the clearinghouse server and administer permission

on the cell root directory. The server principal needs read, write, and administer

permission on the cell root directory.

Examples

The following command repairs invalid timestamps in a clearinghouse named **/.:/blech_ch** on the local system.

```
dcecp > clearinghouse repair /.:/blech_ch -timestamps
```

clearinghouse show

Displays attribute and counter information associated with specified clearinghouses on local or remote machines. The syntax is as follows:

clearinghouse show *clearinghouse_name_list* [**-counters** | **-attributes** | **-all][-schema]**

Options

-attributes

Returns the attributes for the clearinghouse (default).

-counters

Returns the counters for the clearinghouse.

-schema

Indicates whether attributes are singlevalued or multivalued.

The **clearinghouse show** operation displays attribute and counter information associated with the clearinghouses specified by *clearinghouse_name_list*, which is a list of one or more names of the clearinghouses. If more than one clearinghouse is specified, the attributes of all the clearinghouses are concatenated into one list. The order of the returned attributes is the lexical order of the object identifiers (OIDs) of each attribute for each clearinghouse.

If you supply no options, **clearinghouse show** returns the attributes associated with the specified clearinghouse.

Privileges Required

You must have **r** (**read**) permission to the clearinghouse. If you specify a wildcard clearinghouse name, you also need **r** (**read**) permission to the cell root directory.

Examples

The following command displays the attributes of the clearinghouse **/.:/drkstr_ch**.

```
dcecp> clearinghouse show /.:/drkstr_ch
{CDS_CTS 1994-06-18-20:16:22.150-05:00I0.000/00-00-c0-f7-de-56}
{CDS_UTS 1994-06-19-17:17:43.911-05:00I0.000/00-00-c0-f7-de-56}
{CDS_ObjectUUID 0066ccea-d978-1db3-8259-0000c0f7de56}
{CDS_AllUpTo 1994-07-01-21:30:18.948-05:00I0.000/00-00-c0-f7-de-56}
{CDS_DirectoryVersion 3.0}
{CDS_CHName /.../terrapin/drkstr_ch}
{CDS_CHLastAddress
 {Tower ncacn_ip_tcp 130.105.5.16}
 {Tower ncadg_ip_udp 130.105.5.16}}
{CDS_CHState on}
{CDS_CHDirectories
 {{Dir_UUID 00146037-d97b-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin}}
 {{Dir_UUID 0043797a-d991-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys}}
 {{Dir_UUID 004faa42-d992-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/HP}}
 {{Dir_UUID 004fa65a-d993-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/HP/sample-apps}}
 {{Dir_UUID 004b1130-d994-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/dce}}
 {{Dir_UUID 00498a0e-d995-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/subsys/dce/sec}}
 {{Dir_UUID 003ed80c-d996-1db3-8259-0000c0f7de56}
```

```
  {Dir_Name /.../terrapin/subsys/dce/dfs}}
 {{Dir_UUID 003d4d8e-d997-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/hosts}}
 {{Dir_UUID 003bc522-d998-1db3-8259-0000c0f7de56}
  {Dir_Name /.../terrapin/hosts/drkstr}}
 {{Dir_UUID 0089ee8c-44e0-1dbe-929b-0000c0f7de56}
  {Dir_Name /.../terrapin/help}}
 {{Dir_UUID 001c6cea-00fb-1dc5-929b-0000c0f7de56}
  {Dir_Name /.../terrapin/test_1}}
 {{Dir_UUID 00440fe8-02a1-1dc5-929b-0000c0f7de56}
  {Dir_Name /.../terrapin/dirmod}}}
{CDS_ReplicaVersion 3.0}
{CDS_NSCellname /.../terrapin}
```

The following command displays the current values of the counters associated with the **/.:/Chicago1_CH** clearinghouse:

```
dcecp> clearinghouse show /.:/Chicago1_CH -counters
{corruptions 0}
{disables 0}
{enables 1}
{failedupgrades 0}
{missingentries 0}
{reads 2336}
{returnedrefs 2}
{rootunreachables 0}
{skulkfailures 0}
{writes 68}
```

clearinghouse verify

Verifies the consistency of the specified clearinghouse on the local machine. The syntax is:

clearinghouse verify *clearinghouse_name_list*

The **clearinghouse verify** operation verifies the consistency of the specified clearinghouse by checking internal attributes. The required *clearinghouse_name_list* argument is a list of one or more names of clearinghouses you want to verify. This operation returns an empty string on success.

Privileges Required

You need write permission on the clearinghouse server and administer permission on the cell root directory. The server principal needs read, write, and administer permission on the cell root directory.

Examples

The following command verifies the consistency of clearinghouses named **/.:/gumby_ch** and **/.:/pokey_ch**.

```
dcecp > clearinghouse verify {/.:/gumby_ch /.:/pokey_ch}
```

Related Information

Commands: **cdscache(8dce)**, **directory(8dce)**, **link(8dce)**, **object(8dce)**, **clear_clearinghouse(8cds)**, **create_clearinghouse(8cds)**, **delete_clearinghouse(8cds)**, **show_clearinghouse(8cds)**.

directory(8dce)

Name

directory -A **dcecp** object that manages a name service directory

Synopsis

directory add *directory_name_list* **-member** *child_pointer_list* **-clearinghouse** *clearinghouse_name*

directory create *directory_name_list* [**-attribute** *attribute_list* [**-single**]] [[**-replica -clearinghouse** *clearinghouse_name*]| [**-clearinghouse** *clearinghouse_name*]]

directory delete *directory_name_list* [[**-tree**]| [**-replica -clearinghouse** *clearinghouse_name*]]

directory help [*operation* | **-verbose**]

directory list *directory_name_list* [**-directories**][**-objects**][**-links**] [**-simplename** | **-fullname**]

directory merge *source_directory_name* **-into** *destination_directory_name* [**-clearinghouse** *clearinghouse_name*][**-tree**][**-nocheck**]

directory modify *directory_name_list* { [[**-add** *attribute_list*][**-single**]] [[**-remove** *attribute_list* [**-types**]] [**-change** *attribute_list*] [**-propagate [-force**]] [**-upgrade [-tree**]]}

directory operations

directory remove *directory_name_list* **-member** *child_pointer_list*

directory show *directory_name_list* [**-member** *child_pointer_list* | **-replica -clearinghouse** *clearinghouse_name*] [**-schema**]

directory synchronize *directory_name_list*

Arguments

directory_name_list

A list of one or more specific directory names to be operated on. The last simple name can contain wildcard characters.

child_pointer_list

A list of one or more specific child pointers to be operated on.

clearinghouse_name

The name of a single clearinghouse to use during directory operations.

clearinghouse_name_list

A list of one or more clearinghouses to use during the **directory modify** operations.

source_directory_name

The name of one specific directory whose contents are to be copied into a destination directory using a **directory merge** operation.

destination_directory_name

The name of one specific directory that will contain the results of a successful **directory merge** operation.

operation

The name of one specific **directory** operation about which you want to see help information.

attribute_list

A list of one or more legal attributes to use during **directory** operations.

Description

The **directory** object represents Cell Directory Service (CDS) directories. CDS directories are containers for other objects, links, and other directories (as well as clearinghouses). Any of these items that reside in a directory are called children of that directory. Directories also contain attributes that may be viewed or modified.

This object also represents CDS replicas. Replicas are read-only copies of directories stored in other clearinghouses. Several of the supported operations take options to indicate that the command is to operate on a specific replica.

If the **_s(cds)** convenience variable is set, it is treated as the name of a clearinghouse to contact for this operation. This is the only clearinghouse that will be contacted in an attempt to complete the operation. These commands do not set the value of this variable after completion. If a **-clearinghouse** option is used (as described in some commands below), then it overrides the value of **_s(cds)** but the command will not change the setting of **_s(cds)**.

Attributes

The following are the CDS defined attributes that may be present in directories and replicas in CDS:

CDS_AllUpTo

Indicates the date and time of the last successful skulk on the directory. All replicas of the directory are guaranteed to receive all updates whose timestamps are less than the value of this attribute.

CDS_Convergence *value*

Specifies the degree of consistency among replicas. This attribute's value is defined as one of the following:

- **low**

  CDS does not immediately propagate an update. The next skulk distributes all updates that occurred since the previous skulk. Skulks occur at least once every 24 hours.

- **medium**

  CDS attempts to immediately propagate an update to all replicas. If the attempt fails, the next scheduled skulk makes the replicas consistent. Skulks occur at least once every 12 hours.

- **high**

  CDS attempts to immediately propagate an update to all replicas. If the attempt fails (for example, if one of the replicas is unavailable), a skulk is scheduled for within one hour. Skulks usually occur at least once every 12 hours. Use this setting temporarily and briefly, because it uses extensive system resources.

By default, every directory inherits the convergence setting of its parent at creation time. The default setting on the root directory is **medium**.

CDS_CTS

Specifies the creation timestamp (CTS) of the CDS directory.

CDS_DirectoryVersion

Specifies the current version of the directory (derived from the **CDS_DirectoryVersion** attribute of the clearinghouse in which the directory was created). Multiple directory versions are supported in a cell.

CDS_Epoch

A Universal Unique Identifier (UUID) that identifies a particular incarnation of the directory.

CDS_GDAPointers

A set-valued attribute that is only present in the root directory of a cell. This attribute contains location information about registered Global Directory Agents (GDAs) for that cell, similar to the CDS_Replicas attribute. It is created and only used by a GDA.

CDS_InCHName

Indicates whether a directory or any of its descendants can store clearinghouse names. If this value is **true**, the directory can store clearinghouse names. If it is **false**, the directory cannot store clearinghouse names.

CDS_LastSkulk

Records the timestamp of the last skulk performed on this directory.

CDS_LastUpdate

Records the timestamp of the most recent change to any attribute of a directory replica, or any change to an entry in the replica.

CDS_ObjectUUID

Specifies the unique identifier of the directory. This is read-only to the user and is set by the system at creation time.

CDS_ParentPointer

Contains a pointer to this directory's parent in the namespace.

CDS_Replicas

Specifies the address, UUID, and name of every clearinghouse where a copy of this directory is located. This attribute also specifies whether the replica in a particular clearinghouse is a master or read-only replica.

CDS_ReplicaState

Specifies whether a directory replica can be accessed.

CDS_ReplicaType

Indicates whether a directory replica is a master or read-only replica.

CDS_ReplicaVersion

Specifies the version of a replica of the directory.

CDS_RingPointer

Specifies the UUID of a clearinghouse containing another replica of this directory.

CDS_UpgradeTo

A single-valued attribute used to control the upgrading of a directory from one version of CDS to another. By modifying this attribute, the process of upgrading a directory to a newer version of CDS may be initiated.

CDS_UTS

Specifies the timestamp of the most recent update to an attribute of the directory.

Operations

directory add

Creates a child pointer in the parent directory. The syntax is as follows:

directory add *directory_name_list* **-member** *child_pointer_list*
-clearinghouse *clearinghouse_name*

Options

-member *child_pointer_list*

This required option names the child pointers to be added to parent directories in the clearinghouse named by the required **-clearinghouse** option.

-clearinghouse *clearinghouse_name*

This required option names the clearinghouse where the child pointers are to be added.

The **directory add** operation creates a child pointer in the parent directory. The *directory_name_list* argument is a list of one or more names of parent directories to have child pointers added to them. The value of the required **-member** option is a list of names of child pointers to be added to each of the directories listed in the argument. Each child pointer name entered should contain only the last relative distinguished name (RDN) of the name. The child object must exist or the command returns an error. The full name of a clearinghouse that holds a replica of the child directory is given as the value to the required **-clearinghouse** option.

This option may only have one value and is used for each of the values of the **-member** option. The operation returns an empty string on success. If a child pointer of the same name already exists an error is returned. This command is needed only to recreate a child pointer that was accidentally deleted, such as in a troubleshooting situation. Normally child pointers are created internally by CDS when creating directories with the **directory create** command.

Privileges Required

You must have **i** (**insert**) permission to the parent directory.

Examples

```
dcecp > directory add /.: -member foo -clearinghouse /.:/darkstr_ch
```

directory create

Creates a new directory of the specified name. The syntax is as follows:

directory create *directory_name_list* [**-attribute** *attribute_list* [**-single**]] [[**-replica -clearinghouse** *clearinghouse_name*]|
[**-clearinghouse** *clearinghouse_name*]]

Options

-attribute *attribute_list*

Allows you to specify the CDS_Convergence attribute or the CDS_UpgradeTo attribute in an attribute list. The format is as follows:

-attribute {{*attr value***}{***attr value***}}**

See the **ATTRIBUTES** section of this reference page for descriptions of the **CDS_Convergence** attribute and the **CDS_UpgradeTo** attribute.

-single

Valid only with the **-attribute** option, this specifies that attribute values are single-valued. Otherwise, attributes are multivalued.

-replica

This option specifies that the directory created is a replica of an existing directory. If you use the **-replica** option, you must specify a clearinghouse by using the -clearinghouse option.

-clearinghouse *clearinghouse_name*

Required with the **-replica** option and optional when **-replica** option is not present, the **-clearinghouse** option names the clearinghouse to which the child pointers are to be added.

The **directory create** operation creates a new directory of the specified name. The directory_name_list argument is a list of names of directories to be created.

An optional **-attribute** option specifies a list of attributes to be included in each created directory. The attribute values are multivalued unless the **-single** option is specified, in which case all attributes are single-valued. The **-single** option is valid only if the **-attribute** option is specified.

The **-clearinghouse** option specifies one clearinghouse to create all the directories in. If this option is not specified, the new directories are created in the master clearinghouse as the parent directory. The **directory create** command also takes a **-replica** option which indicates that a directory replica is created; when this option is used, the **-clearinghouse** option is required. The operation returns an empty string on success.

Privileges Required

You must have the following permissions in order to create a directory: **r** (**read**) and **i** (**insert**) permission to the parent directory, and **w** (**write**) permission to the clearinghouse in which the master replica of the new directory is to be stored.

In addition, the server principal must have **r** (**read**) and **i** (**insert**) permission to the parent directory.

Examples

The following command creates a directory named **/.:/sales**.

```
dcecp > directory create /.:/sales
```

directory delete

Deletes a directory. The syntax is as follows:

directory delete *directory_name_list*
[[**-tree**]| [**-replica -clearinghouse** *clearinghouse_name*]]

Options

-tree

Removes the directory and everything (all directories, objects, links, and clearinghouses) beneath it.

-replica

Specifies that the directory to delete is a replica of an existing directory. If you use the **-replica** option, you must specify a clearinghouse using the **-clearinghouse** option.

-clearinghouse

Required with the **-replica** option, the **-clearinghouse** option names one clearinghouse (not a list of clearinghouses) from which the replica is to be deleted.

The **directory delete** operation deletes a directory from the CDS name service. The *directory_name_list* argument is a list of names of directories to be deleted. If the directory is not empty, the command returns an error unless the **-tree** option is used. The **-tree** option, which takes no value, removes the directory and everything (all directories, objects, links, and clearinghouses) beneath it.

The **-replica** and **-clearinghouse** options (they must be used together) let you delete a replica instead of a directory. The **-clearinghouse** option specifies the clearinghouse that contains the replica; only one value can be specified, not a list. The operation returns an empty string on sucess. If a specified directory does not exist, an error is generated.

The **-replica** and **-clearinghouse** options cannot be used with the **-tree** option.

Privileges Required

You must have (**delete**) permission to the directory and **w** (**write**) permission to the clearinghouse that stores the master replica of the directory. The server principal needs **A** (**Admin**) permission to the parent directory or **d** (**delete**) permission to the child pointer that points to the directory you intend to delete.

Examples

The following command deletes the directory **/.:/eng** from the namespace:

```
dcecp> directory delete /.:/eng
```

The following command tries to delete a nonempty directory **/.:/depts/phrenology** and gets an error. The second attempt uses the **-tree** option to delete the directory and all the directories and objects beneath it.

```
dcecp> dir delete /.:/depts/phrenology
```

Error: Directory must be empty to be deleted

```
dcecp> dir delete /.:/depts/phrenology -tree
```

directory help

Returns help information about the directory object and operations. The syntax is as follows:

directory help [*operation* | **-verbose**]

Options

-verbose

Displays information about the **directory** object.

The optional operations argument returns information about the operation and its options. Used without an argument or option, the **help** command returns brief information about each **directory** operation.

Privileges Required

No special privileges are needed to use the **directory help** command.

Examples

```
dcecp> directory help
add                 Creates a child pointer in the specified directory.
create              Creates the named directory.
delete              Deletes the named directory.
list                Lists the descendants of a directory.
merge               Merges the contents of one directory into another.
modify              Adds, removes or changes attributes in the named directory.
remove              Removes a child pointer in the specified directory.
show                Returns the attributes of a directory or replica.
synchronize         Skulks the named directory.
help                Prints a summary of command-line options.
operations          Returns a list of the valid operations for this command.
```

directory list

Returns a list of the names of all the descendants of a directory. The syntax is as follows:

directory list *directory_name_list* [**-directories**][**-objects**][**-links**] [-simplename] | **-fullname]**

Options

-directories

This option lists the names of all descendent directories.

-objects

This option lists the names of all descendent objects.

-links

This option lists the names of all descendent softlinks.

-simplename

Returns just the RDN of the name.

-fullname

Returns the entire name.

The **directory list** operation returns a list of the names of all the descendents of a directory. Descendents can include all directories, objects, links, and clearinghouses of the directory. The *directory_name_list* argument is a list of names of directories to be operated on. This command returns only the names of descendents, so there is no way to tell the class of each name unless by convention (for instance, most clearinghouses end with **_ch**). Use the following options to specify the types of descendents to return: **-directories**,

-objects, **-links**. The options take no values and can be used in combination. By default or if the **-fullname** option is specified, fullnames are returned. Use the **-simplename** option to return merely the last RDN of the name.

Privileges Required

You must have **r** (**read**) permission to the directory named in the argument.

Examples

The following example lists a softlink in the **/.:/depts/administration** directory that points to a server entry in the **/.:/depts/phrenology/applications** directory:

```
dcecp> dir list /.:/depts/administration -links
/.../ward_cell.osf.org/depts/administration/bump_server1
```

directory merge

Copies the contents of one directory into another directory. The syntax is as follows:

directory merge *source_directory_name* **-into** *destination_directory_name* [**-clearinghouse** *clearinghouse_name*][**-tree**][**-nocheck**]

Options

-tree

Copies the contents of child directories (as well as the child directories themselves) into the destination directory.

-into *destination_directory_name*

The argument to this required option specifies the name of the destination directory. The destination directory must exist.

-clearinghouse *clearinghouse_name*

Places the new objects (the resulting merged directory) in a clearinghouse other than that of the destination directory.

-nocheck

Lets the **directory merge** operation proceed without checking first for object name collisions or access control list (ACL) problems. Use this option to save time when you're sure problems don't exist.

The **directory merge** operation copies the contents of one directory into another. The argument is the name of the source directory. This command takes a required **-into** option to specify the destination directory which must exist. For example, if **/.:/a** has two child objects **/.:/a/b** and **/.:/a/c**, then **directory merge /.:/a -into /.:/x** would result (assuming no errors) in the following objects: **/.:/x/b** and **/.:/x/c**.

Normally only the immediate contents of the directory are merged. This means all objects, links, and directories, but not the contents of child directories. To merge these as well, use the **-tree** option.

By default, the new objects are placed in the destination directory's master clearinghouse, and all children (no matter how many levels down) are placed in the same clearinghouse. To place any descendant directories in another clearinghouse, use the **-clearinghouse** option with a value. There can only be one clearinghouse specified for all directories involved in the merge operation. To specify more than one, either change this after the merge has happened, or use separate commands.

This command first checks for any collisions or ACL problems before beginning to merge any objects. If there are any problems encountered, an error is generated (not immediately; all objects are checked first), and the names of all problem objects, links or directories are returned in a list. The administrator should then address these problems and rerun the merge command. If the **-nocheck** option is specified the check is not performed. This way time can be saved when trying a known nonproblematic merge. This is not an atomic operation and other changes to the involved objects can cause problems. This command should be issued when others are not modifying the involved directories. Changing ACLs can be done to ensure this. If an error does occur during the actual merging process, it is generated and the operation aborts immediately.

The merge command actually recreates the objects with the same writable attributes of the source objects. This means that some read-only attributes will change between the source and destination. For example, the creation timestamp attribute (**CDS_CTS**) changes.

The resulting merged directory inherits its ACLs from the destination directory's Initial Container or Initial Object ACLs. Consequently, the ACLs of the destination objects are likely to differ from the ACLs of the source objects.

Privileges Required

You must have **r** (**read**) and **i** (**insert**) permission to the destination directory.

Examples

The following command merges the directories but not the contents of the **/.:/depts/phrenology** directory into the **/.:/depts/radiology** directory:

```
dcecp> dir list /.:/depts/phrenology -simple
applications services staff users
dcecp> directory merge /.:/depts/phrenology -into /.:/depts/radiology
dcecp > dir list /.:/depts/radiology -simple
applications services staff users
```

directory modify

Adds, removes, or changes a directory's attributes and their values. The syntax is as follows:

directory modify *directory_name_list*
{[[**-add** *attribute_list*][**-single**]]
[[**-remove** *attribute_list* [**-types**]] [**-change** *attribute_list*]
[**-propagate [-force**]] [**-upgrade [-tree**]]}

Options

-add

Adds a value to a modifiable, set-valued attribute (including application-defined attributes) of a directory. If you enter a byte data type, you must enter an even number of digits in length. You can only enter pairs of hexadecimal values for user-defined attributes.

-single

Used with the **-add** option to specify that the attributes to be added are to be single-valued. Normally all user-defined attributes are defined to be multivalued, even if only one value is specified. This option is not legal without the **-add** option.

-remove

Removes a value from a set-valued or single-valued attribute (including application-defined attributes) of a directory. If you do not specify a value, the command removes the entire attribute. This command can delete attributes created with the **-add** and **-change** options.

-types

Used with the **-remove** option to specify that the value of the **-remove** option is a list of attribute types. The entire attribute is to be removed, not just a value. (This option is not legal without the **-remove** option.)

-change

Changes the value of a modifiable, single-valued attribute of a directory. You can specify an application-defined attribute or the following attribute, which specifies the degree of consistency among replicas:

{CDS_Convergence *value***}**

Specify one of the following for *value*:

- low

  CDS does not immediately propagate any updates. The next skulk distributes all updates that occurred since the previous skulk. Skulks occur at least once every 24 hours.

- medium

  CDS attempts to immediately propagate an update to all replicas. If the attempt fails, the software lets the next scheduled skulk make the replicas consistent. Skulks occur at least once every 12 hours.

- high

  CDS attempts to immediately propagate an update to all replicas. If that attempt fails (for example, if one of the replicas is unavailable), a skulk is scheduled for within one hour. Background skulks occur at least once every 12 hours. Use this setting temporarily and briefly because it uses extensive system resources.

-propagate

Changes the replica set of the child directories to match that of the parent directory. This recursive call modifies the replica set of all child directories found beneath the parent directory. Refer to the *Gradient DCE for Tru64 UNIX Product Guide* for information about cell migration.

-force

Used with the **-propagate** option to specify that the child's replica set be synchronized with the parent's replica set even if the current replica set of the child directories contains a different master replica and does not contain the same number of replica sets.

-upgrade

Upgrades the directories to CDS directory version 4.0. This option is provided as part of a migration tool for upgrading from a Digital DCE V1.3 (OSF DCE R1.0.3+) to Digital DCE V2.0 (OSF DCE R1.1+) base. Refer to the *Gradient DCE for Tru64 UNIX Installation and Configuration Guide* for information about cell migration.

-tree

Used with the **-upgrade** option to specify recursive upgrading of all directories and child directories found beneath the directories named in the operation argument.

The **directory modify** operation adds, removes, or changes a directory's attributes and their values. The argument is a list of one or more names of directories to be operated on. Attribute options are not supported; use one or more of the **-add**, **-remove**,or **-change** options, each of which takes an attribute list as an argument.

Use the **-remove** option to remove a value from an attribute. You can use the **-types** option along with the **-remove** option to remove an entire attribute or list of attributes.

Some attributes in CDS are multivalued. For instance, the **CDS_Replicas** attribute can specify the locations and names of several clearinghouses that maintain copies of a directory. The **-add** operation needs to be instructed whether to operate on single-valued or multivalued attributes. Omit the **-single** option to operate on multivalued attributes. Include the **-single** option to operate on single-valued attributes.

The **-upgrade** and **-tree**, options support cell migration to Digital DCE Version 2.0 (based on OSF DCE Release 1.1). For information about cell migration, refer to the *Gradient DCE for Tru64 UNIX Installation and Configuration Guide*.

The **-propogate** and **-force** options support synchronization and creation of child directory replica sets. For information about synchronizing replica sets, refer to the *Gradient DCE for Tru64 UNIX Product Guide*.

Most attributes are usually managed by the client application. See the *OSF DCE Administration Guide* for more information about attributes. All modifications are made to each directory listed in the argument. An error in any one causes the command to abort immediately and generate an error. The operation returns an empty string on success.

Privileges Required

You must have **w** (**write**) permission to the directory to add, remove, or change attributes.

Examples

The following command sets the **CDS_Convergence** attribute on the **/.:/depts/radiology** directory to a value of low:

```
dcecp> directory modify /.:/depts/radiology -change {CDS_Convergence low}
```

To add the value **ontario** to the attribute **myname** of a directory named **/.:/sales**, read the **cds_attributes** file to verify that the attribute shown in the following display exists:

OID	LABEL	SYNTAX
1.3.22.1.3.91	myname	char

Enter the following command to assign the value **ontario** to the attribute **myname**:

```
dcecp> directory modify /.:/sales -add {myname ontario}
```

To remove the value **1** from the user-defined, set-valued attribute **dirregion** of a directory named **/.:/sales**, follow these steps:

1 Read the **cds_attributes** file to check that the attribute **dirregion** is listed, as shown in the following display:

OID	LABEL	SYNTAX
1.3.22.1.3.66	dirregion	small

2 Enter the following command to remove the value **1** from the attribute **dirregion**:

```
dcecp> directory modify /.:/sales -remove {dirregion 1}
```

directory operations

Returns a list of the operations supported by the **directory** object. The syntax is as follows:

directory operations

The **directory operations** command takes no arguments, and returns a list of the available operations for the **directory** object. The order of the elements is alphabetical with the exception that **help** and **operations** are listed last.

Privileges Required

No special privileges are needed to use the **directory operations** command.

Examples

```
dcecp> directory operations
add create delete list merge modify remove show synchronize help operations
```

directory remove

Deletes a child pointer from the directories specified. The syntax is as follows:

directory remove *directory_name_list* **-member** *child_pointer_list*

Options

-member *child_pointer_list*

This required option names the child pointers to be removed from each directory in the operation argument.

The **directory remove** operation deletes a child pointer from the directories specified. The *directory_name_list* argument is a list of names of one or more directories to be operated on. The required **-member** option allows you to list the child pointers to be removed from each specified directory. The operation returns an empty string on success.

The *child_pointer_list* argument value of the required **-member** option is a list of one or more child pointers (specified as only one RDN each) to be removed from each directory in the argument.

This command is needed only to delete a child pointer that accidentally remains after the child directory is deleted. Normally child pointers are removed internally by CDS when deleting directories with the **directory delete** command.

Privileges Required

You must have **d** (**delete**) permission to the child pointer or **A** (**Admin**) permission to the parent directory.

Examples

The following command deletes the child pointer that accidentally remains after the **/.:/sales/east** directory is deleted:

```
dcecp> directory remove /.:/sales -member east
```

directory show

Returns a list of attributes for the specified directories and optionally, their specified contents. The syntax is as follows:

directory show *directory_name_list*
[**-member** *child_pointer_list* | **-replica -clearinghouse** *clearinghouse_name*]
[**-schema**]

Options

-member *child_pointer_list*

The optional **-member** option takes one required value which is the last RDN of the child pointer in the directory specified by the operation argument. The returned list describes the child pointer information for the specified member stored in the specified directories. This option may not be combined with the **-replica** or **-clearinghouse** option.

-replica

This option specifies that the directory shown is a replica of an existing directory. If you use the **-replica** option, you must specify a clearinghouse with the **-clearinghouse** option.

-clearinghouse *clearinghouse_name*

Required with the **-replica** option, the **-clearinghouse** option names the clearinghouse where the named replica exists.

-schema

This option returns whether an attribute is single or multivalued. This is specific to a directory, meaning that the same attribute can be single-valued on one directory and multivalued on another directory. This option may not be used with other options.

The **directory show** operation returns a list of attributes for the specified directories and optionally, their specified contents. The *directory_name_list* argument is a list of one or more names of directories to be operated on. When used without any options, this command returns the attributes associated with the named directories. If more than one directory is specified, then all the arguments are grouped together in one list. The order of the returned arguments is the lexical order of the object indentifiers (OIDs) of each attribute for each directory.

You can request attributes of specific replicas in specific clearinghouses by using the **-replica** and **-clearinghouse** options. Alternatively, you can request attributes of child pointers by using the **-member** option.

Privileges Required

You must have **r** (**read**) permission to the directories named in the argument list.

Examples

The following example shows the attributes for the **/.:/depts/radiology** directory:

```
dcecp> directory show /.:/depts/radiology
{RPC_ClassVersion
 {01 00}}
{CDS_CTS 1994-07-08-17:01:03.115+00:00I0.000/00-00-c0-8a-df-56}
{CDS_UTS 1994-07-08-19:36:31.719+00:00I0.000/00-00-c0-8a-df-56}
{CDS_ObjectUUID 2df03af4-9a76-11cd-8f2b-0000c08adf56}
{CDS_Replicas
 {{CH_UUID b32648c6-928d-11cd-b4b5-0000c08adf56}
  {CH_Name /.../ward_cell.osf.org/pmin17_ch}
  {Replica_Type Master}
  {Tower ncacn_ip_tcp:130.105.1.227[]}
  {Tower ncadg_ip_udp:130.105.1.227[]}}}
{CDS_AllUpTo 1994-07-08-17:01:05.945+00:00I0.000/00-00-c0-8a-df-56}
{CDS_Convergence medium}
{CDS_ParentPointer
 {{Parent_UUID 8eeb369a-9a4b-11cd-8f2b-0000c08adf56}
  {Timeout
   {expiration 1994-07-09-17:13:31.959}
   {extension +1-00:00:00.000I0.000}}
  {myname /.../ward_cell.osf.org/depts/radiology}}}
{CDS_DirectoryVersion 3.0}
{CDS_ReplicaState on}
{CDS_ReplicaType Master}
{CDS_LastSkulk 1994-07-08-17:01:05.945+00:00I0.000/00-00-c0-8a-df-56}
{CDS_LastUpdate 1994-07-08-19:36:31.719+00:00I0.000/00-00-c0-8a-df-56}
{CDS_RingPointer b32648c6-928d-11cd-b4b5-0000c08adf56}
{CDS_Epoch 2f617aa6-9a76-11cd-8f2b-0000c08adf56}
{CDS_ReplicaVersion 3.0}
```

The following example uses the **-schema** option to show whether the attributes for the **/.:/depts/radiology** directory are single-valued or multivalued:

```
dcecp> directory show /.:/depts/radiology -schema
{RPC_ClassVersion multi}
{CDS_CTS single}
{CDS_UTS single}
{CDS_ObjectUUID single}
{CDS_Replicas multi}
{CDS_AllUpTo single}
{CDS_Convergence single}
{CDS_ParentPointer multi}
{CDS_DirectoryVersion single}
{CDS_ReplicaState single}
{CDS_ReplicaType single}
{CDS_LastSkulk single}
{CDS_LastUpdate single}
{CDS_RingPointer single}
{CDS_Epoch single}
{CDS_ReplicaVersion single}
```

directory synchronize

Initiates an immediate skulk of the directories specified. The syntax is as follows:

directory synchronize *directory_name_list*

The **directory synchronize** operation initiates an immediate skulk of the directories specified. The *directory_name_list* argument is a list of names of one or more directories to be operated on. Skulks begin immediately in sequence. The command does not return until all skulks complete. The operation returns an empty string on success.

Privileges Required

You must have **A** (**Admin**), **w** (**write**), **i** (**insert**), or **d** (**delete**) permission to the directory. The server principal needs **A** (**Admin**), **r** (**read**), and **w** (**write**) permission to the directory.

Examples

The following command begins a skulk on the **/.:/admin** directory:

```
dcecp> directory synchronize /.:/admin
```

Related Information

Commands: **dcecp(8dce)**, **clearinghouse(8dce)**, **link(8dce)**, **object(8dce)**, **add directory(8cds)**, **create directory(8cds)**, **delete directory(8cds)**, **list directory(8cds)**, **remove directory(8cds)**, **set directory(8cds)**, **show directory(8cds)**.

ldap_addcell(8dce)

Name

ldap_addcell - Registers DCE cell information in a server that supports LDAP.

Synopsis

ldap_addcell -h *ldap_server* **-a** *authentication_DN* **-p** *password*
[**-o** *object_class*, *object_class*...] | [**-d**]

Options

-h *ldap_server*

The name of the LDAP server targeted to hold the binding.

-a *authentication_DN*

The distinguished name (DN), specified in LDAP name syntax, that is to be authenticated for successive operations to use.

-p *password*

The password that is used to authenticate the distinguished name (DN).

-o *object_class, object_class...*

Value or values of the attribute object_class for the entry being created or modified. If you are listing more than one object_class value, you must separate them with commas. If no object_class value is specified, it is assumed that the object exists and an attempt is made to modify its attributes, **CDS_CELL** and **CDS_REPLICAS**.

If this attempt fails, another attempt is made with the addition of the *object_class* value of *dceCellInfoO*. *"dceCellInfoO" is the recommended auxilliary object class for the* **CDS_CELL** and **CDS_REPLICAS** attributes

-d

Deletes the DCE cell information attributes from the entry in the directory. It does not remove the directory entry.

Description

The **ldap_addcell** command registers DCE cell information in a server that supports LDAP. It returns a zero (0) on success and a one (1) on error.

Privileges Required

You must log in as superuser (root) to run the ldap_addcell command.

Examples

The following ldap_addcell examples assume the following:

- The ldap server machine name is mymachine.mycity.mycompany.com.
- gdatest is a user that has write access to the ldap server.
- gdatest is also the password of the user gdatest.
- An organizational unit is allowed to contain the auxiliary object dceCellInfo.
- The ldap server does schema checking.

Example 1 shows the normal creation of the cell bindings in the ldap server.

```
ldap addcell -h mymachine.mycity.mycompany.com -a \
"cn=gdatest, ou=mycity, o=mycompany, c=mycountry" \
-p "gdatest" -o organizationalUnit, dceCellInfo
```

Example 2 shows the deletion of the **CDSCELL** and **CDSREPLICAS** attributes.

```
ldap addcell -h mycountry.mycity.mycompany.com -a \
"cn=gdatest, ou=mycity, o=mycompany, c=mycountry" \
-p "gdatest" | -d
```

Example 3 shows the changing of the **CDSCELL** and **CDSREPLICAS** attributes in an object that exists.

```
ldap addcell -h mycountry.mycity.mycompany.com -a \
"cn=gdatest, ou=mycity, o=mycompany, c=mycountry" \
-p "gdatest"
```

Environment Variables Used

Each parameter of the **ldap_addcell** command has a corresponding environment variable, which is used when the corresponding parameter is not present on the **ldap_addcell** command invocation. The **ldap_addcell** parameters and the corresponding environment variables are as follows:

-h	LDAP_SERVER
-a	LDAP_AUTH_DN
-p	LDAP_AUTH_DN_PW
-o	LDAP_OPJECT_CLASS

If the cell entry is already registered, the **CDSCELL** and **CDSREPLICAS** attributes are replaced with new values for this cell.

Related Information

None

gdad(8cds)

Name

gdad - Starts the global directory agent (GDA) daemon.

Synopsis

gdad [**-b**][**-d** *routing*][**-D**][**-r** *resolve.conf file*][**-u**][**-w** *route*]\
[**-x**][**-l**]| [**-h** *addr_LDAP_server* [*port*] **-a** *authentication_DN* \
-p *password_for authentication_DN*]| [**-h** *addr_LDAP_server* [*port*]]

Options

-b

Disables the use of DNS as the global directory service.

-d *routing*

Sets the debug routing. This option is for debugging use only.

-D

Sets up a debugging condition that refuses to let the gdad process fork.

-r *resolve.conf file*

Specifies the file name of the resolve configuration file. The default is **/etc/resolv.conf**.

-s *named.ca file*

Indicates the file name of the bind server namec data file. The default is **/etc/named.data.**

-u

Prevents **gdad** from updating the GDA parent pointer on **/.:**. This option is for debugging use only.

-w *route*

Routes serviceability messages.

-l

Disables the use of LDAP as global directory service.

-h *addr_LDAP_server port*

Specifies the address of the LDAP server in which DCE cell information is registered. The value addr_LDAP_server and port is the port on which the LDAP server is listening. This is needed if the default port [389] is not used. Alternatively, the host can be specified as an IP address in dotted decimal format. The **-h** option is required when **-l** is not present.

-a *authentication_DN*

Authenticates a distinguished name (DN).

-p *password_for authentication_DN*

Specifies the password used to authenticate the distinguished name (DN).

-x

Disables X.500 name resolution.

Description

The **gdad** command starts the Global Directory Agent (GDA) daemon. The GDA enables intercell communication, serving as a connection to other cells through the global environment.

Privileges Required

You must log in as **superuser** (**root**).

Notes

This command is ordinarily executed by a DCE configuration or startup script. Use this command interactively only when a **gdad** process fails to start automatically after a reboot, or if you want to restart the GDA daemon after disabling it to perform a backup or do diagnostic work on the host system.

Examples

To start the **gdad** process, follow these steps:

1 Log in to the system as **superuser** (**root**).

2 Verify that the **dced** and **gdad** processes are running.

3 3. Enter the following command to restart the **gdad** process:

gdad

To stop the GDA, enter the following command:

kill *pid*

where pid is the process identifier of the **gdad** process.

Related Information

Books: *DCE Administration Guide - Core Components*

dtsd(8dts)

Name

dtsd - Restarts the DTS daemon

Synopsis

dtsd -s [*server options*][*common options*]
dtsd -c [*common options*]

Server Options

-k courier

Runs **dtsd** as a courier.

-k noncourier

Runs **dtsd** as a noncourier.

-o

When enabling as a server, set the clock immediately. Equivalent to the command **enable set clock true** in **dtscp** or to the command **dcecp dts activate -abruptly**.

-g

Runs **dtsd** as a global server.

-s

Runs **dtsd** as a server. The default is **backup courier** and **local server**. **-s** or **-c** must be on the command line to specify server type.

-c

Runs **dtsd** as a clerk. **-s** or **-c** must be on the command line to specify server type.

Common Options

-d

Debug mode. The command runs in the foreground.

-w *serviceability*

See the **svcroute(5dce)** reference page for the full description of the appropriate format for this entry. Only the three-field format, *severity:how:where* is used. An example is:

FATAL:TEXTFILE:/dev/console

-m

Accept timing messages in DLI. (Default is RPC only.)

Description

The **dtsd** command invokes the DTS daemon (server or clerk process). This command is usually executed as part of the overall DCE startup script, **dcesetup**.

You can enter the command manually under the following conditions:

- If a DTS daemon fails to start automatically upon reboot
- If you want to restart a daemon that you shut down to perform a backup or do diagnostic work

In normal rebooting, the **rc.dce** script automatically provides arguments appropriate to the choice of configuration options.

The command line options shown here can also be provided to **dced** as part of the fixed configuration strings, if **dced** is configured to automatically start **dtsd**.

If **dtsd** is started with no arguments (other than **-d** and **-w**), then the server must be started with **dcecp**. The following example configures a local server:

```
dcecp> dts configure -notglobal
dcecp> dts activate
```

Privileges Required

DTS runs as the host machine principal, which is usually **root**. See the security reference section for information about principals.

Notes

Use **dtsd** interactively only when troubleshooting; otherwise use the **/sbin/rc3.d/S66dce** script to start the DTS daemon. On some systems the superuser is associated with the machine principal.

Examples

To restart the daemon, follow these steps:

1 Log into the system as **superuser** (**root**).

2 Use the **ps** command to make sure that **dced** and **cdsadv** are running. (The DCE daemon provides the endpoint mapping and security services, and **cdsadv** provides CDS.)

3 Use the following command to restart the **dts** daemon as a clerk:

dtsd -c

To restart the **dts** daemon as a server use the following command:

dtsd -s

To restart the **dts** daemon as a global server, setting the clock on startup, use the following command:

dtsd -s -g -o

Related Information

Commands: **dtscp (8dts)**, **dtsdate (8dts)**, **dcecp (8dce)**.

Files: **svcroute(5dce)**.

Books: *DCE Administration Guide - Core Components*

randd(8sec)

Name

randd - Stops and restarts the RAND daemon

Synopsis

randd

Description

The **randd** command is executed as part of the overall DCE startup script, **dcesetup**. Super (root) users can stop and restart the RAND daemon. The daemon, which has a low priority, takes over idle central processing unit (cpu) resources, which it uses to produce random numbers. The random numbers

are stored in memory until requested by processes, often security processes. If the daemon is not present, security daemons calculate the random numbers as they did in the past.

Privileges Required

Superuser (root) priviledge is required.

Examples

To restart the daemon, follow these steps:

1 Log into the system as **superuser** (**root**).

2 Use the **ps** command to check whether the **randd** is running.

ps auxw | grep randd

Related Information

None