VMS Software

VSI X.500 Directory Service Programming Reference

Operating System and Version: VSI OpenVMS Alpha Version 8.4-2L1 or higher
VSI OpenVMS IA-64 Version 8.4-1H1 or higher

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

VSI X.500 Directory Service Programming Reference

VMS Software

Copyright © 2026 VMS Software, Inc. (VSI), Boston, Massachusetts, USA

Legal Notice

Confidential computer software. Valid license from VSI required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for VSI products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. VSI shall not be liable for technical or editorial errors or omissions contained herein.

HPE, HPE Integrity, HPE Alpha, and HPE Proliant are trademarks or registered trademarks of Hewlett Packard Enterprise.

Table of Contents

Preface ..... **v**
1. About VSI v
2. Intended Audience v
3. Related Documents v
4. VSI Encourages Your Comments v
5. OpenVMS Documentation v
6. Typogharical Conventions v

Chapter 1. The XDS Programming Interface **1**
1.1. C Language Binding 1
1.2. C Naming Conventions 1
1.3. Function Return Values 2
1.4. Compiling and Linking 3

Chapter 2. Interface Description **5**
2.1. Abstract Services and Interface Functions 5
2.2. Negotiation Sequence 6
2.3. Session 7
2.4. Context 7
2.5. Function Arguments 8
2.5.1. Attribute and AVA 8
2.5.2. Entry Info Selection 8
2.5.3. Name 9
2.6. Function Results 9
2.6.1. Status 9
2.6.2. Result 9
2.6.3. Invoke-ID 10
2.7. Synchronous and Asynchronous Operation 10
2.7.1. Synchronous 10
2.7.2. Asynchronous 10

Chapter 3. Interface Class Definitions **13**
3.1. Access Point 14
3.2. Address 14
3.3. Attribute 15
3.4. Attribute List 15
3.5. AVA 15
3.6. Common Results 16
3.7. Compare Result 16
3.8. Context 17
3.9. Continuation Reference 19
3.10. Distinguished Name 20
3.11. Entry Information 21
3.12. Entry Information Selection 21
3.13. Entry Modification 22
3.14. Entry Modification List 22
3.15. Extension 23
3.16. Filter 23
3.17. Filter Item 24
3.18. List Information 26
3.19. List Information Item 27

3.20. List Result 27
3.21. Name 27
3.22. Operation Progress 28
3.23. Partial Outcome Qualifier 29
3.24. Presentation Address 29
3.25. Read Result 30
3.26. Relative Distinguished Name 30
3.27. Relative Name 31
3.28. Search Information 31
3.29. Search Result 31
3.30. Session 32

Chapter 4. Interface Functions 35

4.1. X/OPEN Directory Services (XDS) Functions 35

Chapter 5. Errors 91

5.1. Error 91
5.2. Abandon Failed 92
5.3. Attribute Error 92
5.4. Attribute Problem 92
5.5. Communications Error 93
5.6. Library Error 94
5.7. Name Error 95
5.8. Referral 96
5.9. Security Error 96
5.10. Service Error 97
5.11. System Error 98
5.12. Update Error 98

Chapter 6. Directory Class Definitions 101

6.1. Selected Attribute Types 101
6.2. Selected Object Classes 108
6.3. OM Class Hierarchy 109
6.3.1. Algorithm Identifier 110
6.3.2. Certificate 110
6.3.3. Certificate List 111
6.3.4. Certificate Pair 112
6.3.5. Certificate Sublist 112
6.3.6. Facsimile Phone Number 113
6.3.7. DL Submit Permission 113
6.3.8. Postal Address 114
6.3.9. Search Criterion 115
6.3.10. Search Guide 115
6.3.11. Signature 116
6.3.12. Teletex Terminal Identifier 116
6.3.13. Telex Number 117

Preface

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. Intended Audience

This manual provides reference material for writers of applications that use the XDS interface to the X.500 Directory Service. It assumes that you are familiar with the X.500 Directory Service, as described in *VSI X.500 Directory Service Management*, and understand the X.500 API concepts, as described in *VSI X.500 Directory Service Programming*.

The implementation of XDS described in this document is based on Version 1 of the X/Open™ CAE Specification *API to Directory Services (XDS)*.

Note that although you can also use the XDS interface to access a Distributed Computing Environment (DCE) Cell Directory Service (CDS), this book deals primarily with using XDS in an X.500 environment. If you are using XDS to develop a CDS application, you should refer to the DCE documentation.

3. Related Documents

You may also need the following related documents:

- *VSI X.500 Directory Service Programming*
- *DEC X.500 Directory Service Management*
- *VSI X.500 Directory Service Problem Solving*
- *OSI-Abstract-Data Manipulation*

4. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: `<docinfo@vmssoftware.com>`. Users who have VSI OpenVMS support contracts through VSI can contact `<support@vmssoftware.com>` for help with this product.

5. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

6. Typogharical Conventions

The term X.500 API is used to describe XDS. For details of the XDS interface and DCE CDS, refer to the DCE documentation.

The following conventions are used in this book:

Convention	**Meaning**
italics	Indicates an attribute value
`this typeface`	Indicates an example of code

Chapter 1. The XDS Programming Interface

1.1. C Language Binding

The C binding specifies identifiers for all the elements of the X.500 API so that applications written in C can access the directory. These elements include function names, typedef names and constants.

The definitions of the C identifiers appear in several header files:

- xom.h and xomi.h contain definitions for the associated OM API.
- xds.h contains definitions for the Directory Service.
- xdsbdcp.h contains definitions for the Basic Directory Contents Package.
- xdssap.h contains definitions for the Strong Authentication Package.
- xdsmdup.h contains definitions for the MHS Directory User Package. It includes xmhp.h.
- xdsdec.h contains definitions of OM attributes and Directory attributes specific to DSA.

Note

If you have the DCE software installed on your system you will also have other header files specific to DCE.

1.2. C Naming Conventions

The X.500 API uses part of the C public namespace for its facilities. All identifiers start with the letters ds or DS. The interface reserves all identifiers starting with the letters ds P for private use by implementations of the X.500 API. It also reserves all identifiers starting with the letters dsX or DSX for vendor-specific extensions to the X.500 API. You must not use any identifiers starting with these letters. The table below presents details of the C naming conventions.

Table 1.1. Naming Conventions for C Identifiers

Item	Prefix
Reserved for implementors	ds P_
Reserved for interface extensions	dsX_
Reserved for interface extensions	DSX_
Functions	ds_
Error values	DS_E_
OM class names	DS_C_
OM value length limits	DS_VL_
OM value number limits	DS_VN_

Item	Prefix
Other constants	DS_
Attribute Type	DS_A_
Object Class	DS_O_

See *OSI-Abstract-Data Manipulation* for information about the naming conventions used in the OM API.

Certain C identifiers are documented within this book. These are as follows:

- OM class names are written in English. The equivalent C identifier follows the English name and is spelt entirely in upper case and starts with DS_C_, for example, Read Result and DS_C_READ_RESULT.
- Constants that represent errors are treated as a special case. Values of the OM attribute Problem that are a subclass of the OM class Error are written in English. The equivalent C identifier follows the English name and is spelt entirely in upper case, starts with DS_E_ and is enclosed in parenthesis, for example, Alias Problem (DS_E_ALIAS_PROBLEM).
- Enumeration constants, the names of OM attributes, and all other constants except errors are written in English. The equivalent C identifier follows the English name and is spelt entirely in upper case, starts with DS_ and is enclosed in parenthesis, for example, Information Type (DS_INFO_TYPE).

The remaining C identifiers are derived as follows:

- All hyphens and spaces are translated to underscores.
- Function names are spelt entirely in lower case and prefixed by ds_. For example, Receive-Result becomes ds_receive_result. The exception to this rule is the Trace-Object function (a Digital extension), which becomes dsX_trace_object.
- Enumeration tags are derived by prefixing the name of the corresponding OM syntax with DS_ and replacing hyphens (-) with underscores (_). The case of the letters is not changed. For example, Enum(Limit-Problem) becomes DS_Limit_Problem.
- C function parameters are derived from the names of the function arguments and results, and are spelt entirely in lower case. The names of results have _return appended. For example, the argument Name becomes name while the result Operation-Status becomes operation_status_return.
- Where the value of the upper limit in either the Value Length or Number of Values columns of the OM class definition tables is not equal to one, a C identifier can be used to represent this upper limit. The identifier is the OM attribute name in upper case and prefixed by either DS_ VL_ for Value Length, or DS_VN_ for Number of Values. For example, the upper limit of the string length of the Postal Address attribute of the Postal Address object (30), can be represented by the identifier DS_VL_POSTAL_ADDRESS.

1.3. Function Return Values

The return value of a C function is bound to the Status result of the language- independent description. Functions return a value of the type DS_status, indicating whether any errors occurred during the function call. If the function succeeds, then its value is the constant Success. If a function returns a status other than this, then it has not updated the return parameters. The value of the status in this case is an error object as described in *Chapter 5, "Errors"*.

1.4. Compiling and Linking

All application programs that use the X.500 API must include the xom.h and xds.h header files, in that order. After you include these headers you can include any of the optional header files. See *VSI X.500 Directory Service Programming* for details of these header files and information about linking your application program.

Chapter 2. Interface Description

This chapter describes the X.500 API functions and the Object Management (OM) classes that are used as arguments and results of these functions. These functions and objects are based on the standard X.500 Abstract Services.

2.1. Abstract Services and Interface Functions

This section describes the X.500 Abstract Services available and explains how they map to the interface functions. The Abstract Services are used to interact with the X.500 Directory and each maps to a single function call.

The Abstract Services are:

- Abandon
- Add Entry
- Compare
- DirectoryBind
- DirectoryUnbind
- List
- ModifyEntry
- ModifyRDN
- Read
- RemoveEntry
- Search

Each of the abstract services maps to a single function of the same name, with the following exceptions:

- DirectoryBind maps to Bind
- DirectoryUnbind maps to Unbind

The X.500 API functions are summarized in Table 2–1 and are fully described in Chapter 4. Not all these functions are applicable to a CDS Directory; see your DCE documentation for information about using the XDS interface with CDS. The column headed Asynchronous in Table 2–1 indicates whether the function can execute asynchronously.

Table 2.1. Interface Functions

Name	Asynchronous	Description
Abandon	no	Abandons an outstanding operation.

Name	Asynchronous	Description
Add-Entry	yes	Adds an entry to the Directory Information Tree (DIT).
Bind	no	Opens a session with the directory.
Compare	yes	Compares a given attribute value with the attribute value stored in the directory for a particular entry.
Initialize	no	Initializes the interface.
List	yes	Lists the immediate subordinate entries of a particular entry.
Modify-Entry	yes	Modifies a directory entry.
Modify-RDN	yes	Changes the Relative Distinguished Name (RDN) of an entry.
Read	yes	Queries information on an entry by name.
Receive-Result	no	Retrieves the result of an asynchronously executed function.
Remove-Entry	yes	Removes an entry from the DIT.
Search	yes	Finds entries of interest in a portion of the DIT. Shutdown no Shuts down the interface.
Unbind	no	Closes a directory session.
Version	no	Negotiates features of the interface and service.
Trace-Object	no	VSI extension. Displays an explanation of the content of an object.

2.2. Negotiation Sequence

This section describes the initialization and shutdown sequence of the interface and the optional features that this allows. The sequence involves the following functions:

Initialize

The Initialize function returns a workspace. This workspace supports only the standard Directory Service Package (xds.h). It does not support any other packages or extensions. See *Chapter 3, "Interface Class Definitions"* for details of the Directory Service Package.

The workspace can be extended to support the optional Basic Directory Contents Package (xdsbdcp.h), the Strong Authentication Package (xdssap.h) or the MHS Directory User package (xdsmdup.h). See *Chapter 6, "Directory Class Definitions"* for details of these optional packages.

The workspace can also be extended to include any vendor-specific extension. Digital extensions are described in *VSI X.500 Directory Service Programming*.

Version

Extensions to the workspace are supplied as arguments to the Version function. When a workspace with the required features has been set up in this way, the application can use it as required to create and manipulate objects using the OM functions and to start one or more sessions using the Bind function.

Shutdown

When the application has completed all its tasks, it must end all its sessions by calling Unbind once for each call made to Bind. Then the application must release all its objects by calling the OM-Delete function for each top-level object, and release all resources associated with the workspace by calling the Shutdown function.

You retain access to service-generated public objects after the Shutdown function has been called. You can also start another cycle by calling the Initialize function if required by the application.

2.3. Session

This section describes what a session is and how it is started and ended. It also explains the setting of session parameters and the defaults.

An object of OM class Session contains the address of the Directory System Agent (DSA) a particular directory operation is sent to. A Session object contains some DirectoryBind arguments, for example, the Distinguished Name of the requesting DUA. The Session object is passed as the first argument to most of the interface functions.

You can create an object of the OM class Session and set appropriate attribute values using the OM functions. A directory session is then started by calling the Bind function and ended by calling the Unbind function. A session with default attributes can be started by passing the constant Default-Session as the session argument to the Bind function. You can also use the DUA defaults file to specify some of the attribute values. See *VSI X.500 Directory Service Programming* for information about Default-Session and the DUA defaults file.

The Bind function must be called before the Session object can be used as an argument to any other directory interface function. The Bind function returns the Bound Session object, which is a parameter to other directory functions. After calling the Unbind function you must call Bind again if you want to start another session. You may re-use an existing Session object if you call Bind after calling Unbind.

2.4. Context

An object of OM class Context defines the characteristics of the interaction with the Directory Service. These are specific to a particular directory operation but often remain unchanged for many operations. It is unlikely that a user will want to change the parameters that determine these characteristics often. The Context object is supplied as the second argument to many Directory requests. This reduces the number of arguments passed to each function, as it has the same effect as passing a group of additional arguments on every function call.

The value of each attribute of the Context object is determined when the function is called and remains fixed throughout the operation. All OM attributes in the Context object have default values and the

constant Default- Context can be passed as the value of the Context argument to cause all these system defaults to be taken. You can also use the DUA defaults file to specify some of the values. Values given in the DUA defaults file override the system defaults. If you supply a Context object and not the Default-Context constant, the values in the DUA defaults file are not ignored. See *VSI X.500 Directory Service Programming* for information about Default-Context and the DUA defaults file.

2.5. Function Arguments

This section describes arguments to operations and how these are mapped to function arguments. When a function argument is an instance of a particular OM class, it can also be an instance of any subclass of that class. For example, most functions have a Name argument that accepts values of the OM class Name or an instance of the subclass DS-Name. All arguments that are objects can be supplied to the function as either public objects or private objects.

2.5.1. Attribute and AVA

Each directory attribute is represented by an object of the OM class Attribute. This OM class has attributes that represent the attribute type and attribute values. See Section 3.3 for further information on these attributes.

The representation of the attribute value depends on the attribute type and is determined as set out in the list below:

- If the attribute type and the representation of the corresponding values are defined in a package, the attributes are represented as specified in the package. The selected attribute types defined in *Section 6.1, "Selected Attribute Types"* are an example of this (but note that this XDS implementation does not support any of these packages).
- If the value is an ASN.1 simple type, then the representation is the corresponding type specified in *OSI-Abstract-Data Manipulation*.
- If the value is an ASN.1 structured type, then the value is represented in Basic Encoding Rules (BER) with an OM syntax String (Encoding).

An ***attribute-value-assertion (AVA)*** is an assertion about the value of an attribute of a particular entry. It consists of an attribute type and a single value.

An AVA is represented by an instance of the OM class AVA, which is a subclass of Attribute.

2.5.2. Entry Info Selection

The Selection arguments of the Read and Search functions tailor the function results to obtain just part of the required entry. Information about all attributes, no attributes, or about a named set of attributes can be chosen. Attribute types are always returned, but the attribute values do not have to be.

The value of the Selection argument is an instance of the OM class Entry-Info- Selection. In a simple case you can use one of the following constants:

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

To choose a particular set of attributes you must create a new instance of the OM class Entry-Info-Selection. See *VSI X.500 Directory Service Programming* for more information.

2.5.3. Name

Most functions take a Name argument to specify the target entry of the function. The name is represented by an instance of one of the subclasses of the OM class Name.

The Name specified may contain aliases. The Dont-Dereference-Alia ses control determines whether such aliases are dereferenced. You can set the control in the Context parameter. By default the control is *false* for directory interrogations and *true* for modifications.

A Relative Distinguished Name is represented by an instance of a subclass of the OM class Relative-Name. This subclass is DS-RDN.

2.6. Function Results

All functions return the C function result Status. If a function was invoked asynchronously, it also returns an Invoke-ID that identifies the particular invocation of that function. The interrogation functions return a result. The Invoke-ID and Result are returned using pointers that are supplied as arguments of the C functions. These function results are described in the following sections.

All objects returned by functions, including results and errors, are private objects in the workspace of the session object used by that function.

2.6.1. Status

Every function returns a Status value that is either one of the constants Success and No-Workspace, or a pointer to a private object that describes an error. Errors are represented by private objects whose OM class is a subclass of the OM class Error. Details of all errors are in *Chapter 5, "Errors"*.

2.6.2. Result

Directory interrogation operations return a result if they succeed. Any errors from all operations are reported in the Status parameter described in *Section 2.6.1, "Status"*. The value of Result is Success if it is returned by a function call that succeeds in invoking an operation. The value of Result is an error if it is returned by a function call that fails to invoke an operation. The result of an asynchronous operation is returned by a call to the Receive-Result function.

The result of an interrogation is returned in a private object whose OM class is appropriate to the particular operation. The result of a single operation is returned in a single object. The components of the result of an operation are represented by OM attributes in the operation's Result object. All the information contained in the Abstract Service result is available to the application. You can read the result using the functions provided by the OM API.

Only interrogation operations produce results. Each type of interrogation has a specific OM class of object for its result. These OM classes are defined in *Chapter 3, "Interface Class Definitions"* and are called:

- Compare-Result
- List-Result
- Read-Result

- Search-Result

2.6.3. Invoke-ID

All functions that invoke an asynchronous directory operation return an Invoke-ID. This is an integer that uniquely identifies the particular invocation of an operation. It is used to receive the result and status of asynchronous operations or to abandon them. It is not relevant to synchronous operations.

The Invoke-ID of an asynchronous operation is unique amongst the Invoke-IDs of outstanding operations in a given session. The Invoke-ID of a synchronous operation (including all CDS operations), or of a call that fails to invoke an operation, is unspecified.

Asynchronous operations are described in *Section 2.7, "Synchronous and Asynchronous Operation"* and *VSI X.500 Directory Service Programming*.

2.7. Synchronous and Asynchronous Operation

This section describes how the interface provides synchronous and asynchronous modes of operation, and the features of each.

For further information on synchronous and asynchronous operations, see *VSI X.500 Directory Service Programming*.

2.7.1. Synchronous

This is the default mode of operation. The value of the Asynchronous parameter in the Context object is set to *false*. In synchronous mode, all functions wait until the operation is complete before returning. The thread of control is blocked within the interface after calling a function until a result or error is returned. An application is able to make use of the result as soon as the function returns it.

All errors that occur during a synchronous operation are reported when the function returns.

If you call a synchronous function (other than Receive-Result or Abandon) when there is one or more outstanding asynchronous operation, some directory services may return the Library-Error error, *mixed-synchronous*. However the X.500 API allows you to call a function synchronously even if there are outstanding asynchronous operations. To do this you must call the Receive-Result function after each synchronous function call.

2.7.2. Asynchronous

In asynchronous mode, the functions are able to complete before the actual operation is complete. The application is then able to continue with other processing while the operation is being executed by the Directory. You access the result of the function by calling the Receive-Result function.

An application can initiate several concurrent asynchronous operations on the same session before receiving any of the results. The number of outstanding asynchronous operations must not exceed the constant value Max-Outstanding-Operations. This constant is defined in the header file xds.h.

If an error is detected before an asynchronous request is submitted to the Directory, then the function returns immediately and no asynchronous operation is generated. Other errors are notified by the Receive-Result function when the result of the operation is returned.

You must ensure that there are no outstanding asynchronous operations on a session when a call to the Unbind function is made on that session. When the Unbind function is called there is no way to determine whether any outstanding asynchronous operations succeed, or even if they were sent to the Directory.

Chapter 3. Interface Class Definitions

This chapter defines the OM classes that make up the Directory Service Package, as defined in the xds.h header file that is described in *VSI X.500 Directory Service Programming*. The errors defined in *Chapter 5, "Errors"* also belong to this package.

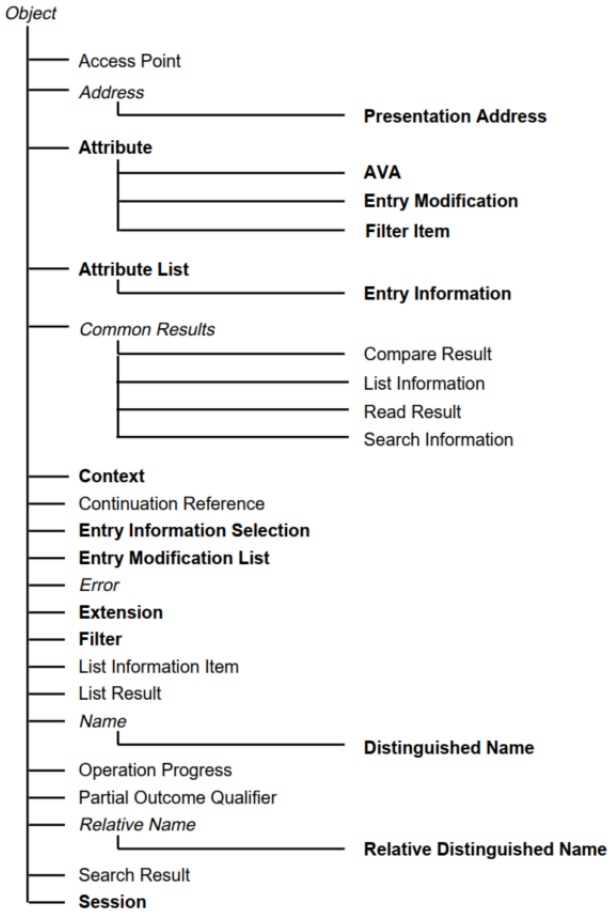
This chapter explains the hierarchical organization of these classes. *Figure 3.1, "Hierarchy of Directory Object Classes"* shows the hierarchy of the classes with the names of abstract classes in italics. An application is not permitted to create or modify instances of some classes because they are returned by the X.500 API, but never supplied to it. Classes that you can create instances of are shown in bold type in *Figure 3.1, "Hierarchy of Directory Object Classes"*. The object class Object is defined in *OSI-Abstract-Data Manipulation*. It contains one attribute, Class, which denotes the class of the object.

This chapter contains descriptions of the OM classes in alphabetical order. Each OM class is described in a separate section that identifies the OM attributes specific to that class and lists its superclasses. The attributes in an instance of an OM class include those specific to that class and those inherited from superclasses. For each class, there is a table that defines the class-specific attributes, if there are any. The tables give the following information:

- The name of each OM attribute
- The syntax of each attribute value (including any appropriate object class, enumeration syntax or string type in parentheses)
- The number of values each attribute can have
- The initial value, if any, that the OM-Create function supplies if initialization is requested (initial values are not supplied for all attributes)

These classes cannot be encoded using the OM-Encode and OM-Decode functions. For more information about OM classes and syntaxes, see *OSI-Abstract-Data Manipulation*.

Figure 3.1. Hierarchy of Directory Object Classes



3.1. Access Point

DS_C_ACCESS_POINT

An instance of the OM class Access Point identifies a particular point at which access to a DSA can occur. An application must not create or modify instances of this class. The class has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Address	Object(Add ress)	1
AE Title	Object(Name)	1

Address (DS_ADDRESS)

Indicates the address of the DSA for use in communication with it.

AE Title (DS_AE_TITLE)

Indicates the name of the DSA.

3.2. Address

DS_C_ADDRESS

An instance of the OM class Address represents the address of a particular entity or service, for example, a DSA. It is an abstract class that has the attributes of its superclass, Object, and no others.

An address is an unambiguous name, label, or number that identifies the location of the entity or service. All addresses are represented as instances of some subclass of this class. The only subclass defined in this guide is Presentation Address, which is the presentation address of an OSI application process.

3.3. Attribute

DS_C_ATTRIBUTE

An instance of the OM class Attribute is a component of a directory entry. The OM class Attribute has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Attribute Type	String(Object Identifier)	1
Attribute Values	any	0 or more

Attribute Type (DS_ATTRIBUTE_TYPE)

Indicates the class of information given by this attribute.

Attribute Values (DS_ATTRIBUTE_VALUES)

Indicates the value syntax and the number of values allowed for this attribute. This is determined by the value of the Attribute Type.

3.4. Attribute List

DS_C_ATTRIBUTE_LIST

An instance of the OM class Attribute List is a list of directory attributes. It has the attributes of its superclass, Object, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values
Attributes	Object(Attribute)	0 or more

Attributes (DS_ATTRIBUTES)

Indicates the attributes that will constitute a new entry, or those attributes selected from an existing entry.

3.5. AVA

DS_C_AVA

An instance of the OM class AVA is a proposition concerning the values of a directory entry. It has the attributes of its superclasses, Object and Attribute, and no others. There must be exactly one value of the attribute Attribute Values. The Attribute Type remains single-valued. The value syntax of Attribute Values must conform with the rules set out in *Section 2.5.1, "Attribute and AVA"*.

3.6. Common Results

DS_C_COMMON_RESULTS

An instance of the OM class Common Results contains results that are returned by the directory interrogation operations. It is an abstract class that has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Alias Dereferenced	Boolean	1
Performer	Object(Name)	0 or 1

Alias Dereferenced (DS_ALIAS_DEREFERENCED)

Indicates whether the name of the target entry that was passed as a function argument included an alias that was dereferenced to determine the distinguished name.

Performer (DS_PERFORMER)

When present, this attribute gives the distinguished name of the performer of a particular operation. It can be present when the result is signed and holds the name of the DSA that signed the result. Note that the presence of this attribute does imply that strong authentication is supported. DSA may return this attribute, but does not support strong authentication.

3.7. Compare Result

DS_C_COMPARE_RESULT

An instance of the OM class Compare Result contains the result of a successful call to the Compare function. An application must not create or modify nstances of this class. The class has the attributes of its superclasses, Object and Common Results, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
From-Entry	Boolean	1
Matched	Boolean	1
Object-Name	Object(Name)	0 or 1

From Entry (DS_FROM_ENTRY)

Indicates whether the assertion was tested against the specified object's entry rather than a copy of it.

Matched (DS_MATCHED)

Indicates whether the assertion specified as an argument was true.

Object Name (DS_OBJ ECT_NAME)

Indicates the Distinguished Name of the target object of the operation. This attribute will be present if the OM attribute Alias Dereferenced is *true*.

3.8. Context

DS_C_CONTEXT

An instance of the OM class Context contains arguments that are accepted by most of the interface functions. Some of these attributes define the service controls that determine how an operation is carried out. The Context object has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values	Initial Value
Aliased RDNs	Integer	0 or 1	–
Extensions	Object(Extension)	0 or more	–
Operation Progress	Object(Operation Progress)	1	*Operation Not Started*
Chaining Prohibited	Boolean	1	*false* [1]
Dont Dereference Aliases	Boolean	1	*false*[1]
Dont Use Copy	Boolean	1	*false*[1]
Local Scope	Boolean	1	*true*[1]
Prefer Chaining	Boolean	1	*true*[1]
Priority	Enum(Priority)	1	*Medium*[1]
Scope Of Referral	Enum(Scope-Of-Referral)	0 or 1	*DMD*[1]
Size Limit	Integer	0 or 1	0[1]
Time Limit	Integer	0 or 1	0[1]
Automatic Continuation	Boolean	1	*true*
Asynchronous	Boolean	1	*false*

[1]This can be overridden by information in the DUA defaults file (see *Section 2.4, "Context"*).

Applications can assume that an object of OM class Context, created with default values of all its OM attributes, will work with all the interface functions. You should ensure that this is the case. The constant Default Context (DS_DEFAULT_CONTEXT) can be used as an argument to interface functions instead of creating an OM object with default values. You can also specify information in the DUA defaults file. See *VSI X.500 Directory Service Programming* for information about setting application defaults.

Aliased RDNs (DS_ALIASED_RDNS)

Indicates how many of the RDNs in the target name have been produced by dereferencing an alias. Its value is zero if no aliases have been dereferenced. This value should be used in the Context parameter of any continued operation.

Extensions (DS_EXT)

Indicates any future standardized extensions that should be applied to the directory.

Operation Progress (DS_OPERATION_PROGRESS)

Indicates the state that the Directory Service is to assume at the start of the operation. It normally takes its default value of *Operation Not Started*.

Chaining Prohibited (DS_CHAINING_PROHIB)

Indicates that chaining the request to other DSAs is prohibited.

Dont Dereference Aliases (DS_DONT_DEREFERENCE_ALIASES)

Indicates that any aliases used to identify the target entry of an operation are not dereferenced. This allows interrogation of alias entries.

Dont Use Copy (DS_DONT_USE_COPY)

Indicates that the request is to be satisfied only by access to entries and not by use of copy entries.

Local Scope (DS_LOCAL_SCOPE)

Indicates that the directory request is to be satisfied locally. The meaning of this option is determined by the DSA implementation but typically it restricts the request to the DSA in which the request originated (as is the case with DSAs) or to that Directory Management Domain.

Prefer-Chaining (DS_PREFER_CHAINING)

Indicates that chaining is preferred to referrals. A DSA does not have to follow this preference and can return a referral even if this parameter has the value *true*. If this parameter has the value *true*, a DSA will always use chaining, rather than returning a referral, provided that authentication is likely to be accepted by the other DSA. See *DEC X.500 Directory Service Management* for more information about chaining and referrals.

Priority (DS_PRIORITY)

Indicates the priority at which the Directory is to try to satisfy the request relative to other directory requests. It is not a guaranteed service as there is no Directory-wide prioritization. Its value is one of:

- *Low* (DS_LOW)
- *Medium* (DS_MEDIUM)
- *High* (DS_HIGH)

VSI X.500 Directory Service does not recognize the Priority settings. All requests sent to a DSA are treated with equal priority.

Scope of Referral (DS_SCOPE_OF_REFERRAL)

Indicates the portion of the Directory to which referrals are to be limited. This includes Referral errors and Partial Outcome Qualifiers. Its value is one of the following:

- *DMD* (DS_DMD)

  DSAs within the Directory Management Domain (DMD) in which the request originates.

- *country* (DS_COUNTRY)

  DSAs in any DMD within the country in which the request originates.

If its value is not set to one of the above, no limit is applied. DSA ignores the setting of this control and does not impose any limit on the scope of referrals.

Size Limit (DS_SIZE_LIMIT)

Indicates the maximum number of objects about which list or search operations should return information. If the limit is exceeded then information is returned about exactly this number of objects. Which objects are chosen is unspecified.

Time Limit (DS_TIME_LIMIT)

Indicates the maximum time, in seconds, within which the service should be provided. If the limit is reached a Service Error, Time Limit Exceeded, is returned by all operations except for List and Search. In this case, List and Search return an arbitrary selection of accumulated results.

Automatic-Continuation (DS_AUTOMATIC_CONTINUATION)

Indicates the requestor's requirement for continuation reference handling, including referrals and those in partial outcome qualifiers. The value is one of the following:

- *false* (OM_FALSE)

  The interface returns all continuation references to the application, which then processes them.

- *true* (OM_TRUE)

  The continuation references are automatically processed and the subsequent results returned to the application whenever practical. This is a much simpler process unless the application has special requirements. The continuation references can still be returned to the application if, for example, the relevant DSA cannot be contacted.

VSI X.500 Directory Service API does not support the Automatic-Continuation service. All referrals are passed to the application, regardless of the setting of this parameter.

Asynchronous (DS_ASYNCHRONOUS)

Indicates whether this service call should operate asynchronously, or not. The value is one of the following:

- *false* (OM_FALSE).

  The operation is performed synchronously (sequentially), with control not returning from the function until a result or error is returned.

- *true* (OM_TRUE).

  The operation is performed asynchronously (non-blocking). The application can perform multiple concurrent asynchronous operations, and can associate a result with the original operation. This result is returned by calling the Receive-Result function. The maximum number of outstanding asynchronous operations is defined by the constant Maximum Outstanding Operations (DS_MAX_OUTSTANDING_OPERATIONS) (defined in xds.h - see *VSI X.500 Directory Service Programming*). The value of this constant is defined by the DSA implementation.

3.9. Continuation Reference

DS_C_CONTINUATION_REF

An instance of the OM class Continuation Reference contains the information that enables a partially completed directory request to be continued, for example, following a referral. An application must not create or modify instances of this class. The class has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Access Points	Object(Access Point)	1 or more
Aliased RDNs	Integer	1
Operation Progress	Object(Operation Progress)	1
RDNs Resolved	Integer	0 or 1
Target Object	Object(Name)	1

Access Points (DS_ACCESS_POINTS)

Consists of the names and presentation addresses of the DSAs where the directory request should be continued.

Aliased RDNs (DS_ALIASED_RDNS)

Indicates how many of the RDNs in the target name have been produced by dereferencing an alias. Its value is zero if no aliases have been dereferenced. This value should be used in the Context parameter of any continued operation.

Operation Progress (DS_OPERATION_PROGRESS)

The state at which the directory request must be continued. This value should be used in the Context parameter of any continued operation.

RDNs Resolved (DS_RDNS_RESOLVED)

Indicates how many RDNs in the supplied object name have been resolved.

Target Object (DS_TARGET_OBJ ECT)

The name of the object upon which the continuation is to be carried out.

3.10. Distinguished Name

DS_C_DS_DN

An instance of the OM class Distinguished Name represents the name of a directory object. It has the attributes of its superclasses, Object and Name, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values
RDNs	Object(Relative Distinguished Name)	0 or more

RDNs (DS_RDNS)

The sequence of RDNs that define a path through the Directory Information Tree (DIT) from its root to the object that the Distinguished Name denotes. The order of the values is significant; the first value is closest to the root and the last value is the RDN of the object.

3.11. Entry Information

DS_C_ENTRY_IN FO

An instance of the OM class Entry Information contains selected information from a single entry. It has the attributes of its superclasses, Object and Attribute List, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
From Entry	Boolean	1
Object Name	Object(Name)	1

From Entry (DS_FROM_ENTRY)

Indicates whether the information was extracted from a specified object's entry or a copy of the entry.

Object Name (DS_OBJ ECT_NAME)

The object's Distinguished Name.

3.12. Entry Information Selection

DS_C_ENTRY_IN FO_SELECTION

An instance of the OM class Entry Information Selection identifies the information to be extracted from an entry. It has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values	Initial Value
All Attributes	Boolean	1	*true*
Attributes Selected	String(Object Identifier)	0 or more	-
Information Type	Enum(Information-Type)	1	*types-and-values*

All Attributes (DS_ALL_ATTRIBUTES)

Indicates which attributes are of interest. Its value is one of the following:

- *false* (OM_FALSE).

  Information is requested about just those attributes listed in Attributes Selected.

- *true* (OM_TRUE).

  Information is requested about all attributes in the entry, apart from operational attributes which must be requested specifically using the Attributes Selected attribute.

Attributes Selected (DS_ATTRIBUTES_SELECTED)

Lists the types of the attributes from which information is to be extracted. If you supply an empty list, no attribute data is returned. This can be used to verify the existence of an entry, or return the Distinguished

Name of an entry. If you require information from operational attributes, you must specifically select those attributes using the Attributes Selected attribute.

Information Type (DS_INFO_TYPE)

Identifies what information is to be extracted from each identified attribute. Its value must be one of:

- *Types-Only* (DS_TYPES_ONLY)

  Only the attribute types of the selected attributes in the entry are to be returned.

- *Types-And-Values* (DS_TYPES_AND_VALUES)

  Attribute types and values of the selected attributes in the entry are to be returned.

3.13. Entry Modification

DS_C_ENTRY_MOD

An instance of the OM class Entry Modification describes one modification to a specified attribute of an entry. It has the attributes of its superclasses, Object and Attribute, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values	Initial Value
Modification Type	Enum(Modification-Type)	1	*add-attribute*

Modification Type (DS_MOD_TYPE)

Identifies the type of modification. Its value must be one of the following:

- *add-attribute* (DS_ADD_ATTRIBUTE)

  The specified attribute is absent and is to be added with the specified values.

- *add-values* (DS_ADD_VALUES)

  One or more specified values are to be added to an existing specified attribute.

- *remove-attribute* (DS_REMOVE_ATTRIBUTE)

  An existing specified attribute is to be removed. Any values in the attribute Attribute-Values are ignored.

- *remove-values* (DS_REMOVE_VALUES)

  One or more specified values are to be removed from an existing specified attribute.

The attribute type to be modified, and the associated values, are specified in the OM attributes Attribute-Type and Attribute-Values, which are inherited from the Attribute superclass.

3.14. Entry Modification List

DS_C_ENTRY_MOD_LIST

An instance of the OM class Entry Modification List contains an ordered sequence of changes to be made to an entry. It has the attributes of its superclass, Object, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values
Changes	Object(Entry Modification)	1 or more

Changes (DS_CHANGES)

Identifies the modifications to be made to the entry of the specified object in the order specified.

3.15. Extension

DS_C_EXT

An instance of the OM class Extension denotes a standardized extension to the Directory Service. It has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values	Initial Value
Critical	Boolean	1	*false*
Identifier	Integer	1	–
Item Parameters	–	1	–

Critical (DS_CRIT)

Has a value of one of the following:

- *false* (OM_FALSE)

  The operation can be performed even if the extension is not available.

- *true* (OM_TRUE)

  The extended operation must be performed or an error must be reported.

Identifier (DS_IDENT)

Identifies the service extension.

Item Parameters (DS_ITEM_PARAMETERS)

Supplies the parameters of the extension. Its syntax is determined by the Identifier.

3.16. Filter

DS_C_FILTER

An instance of the OM class Filter is an assertion about the existence or value of information contained in an entry. A filter is a collection of other filters and filter items connected with Boolean operators.

The possible values of a filter are:

- *false*

- *true*
- *undefined*

The value of a filter is determined by evaluating each of the nested components and combining their values using the Boolean operators. Components whose values are *undefined* are ignored. The filter value is *undefined* if all the component filters and filter items are *undefined*.

An entry is selected if it satisfies the conditions defined in the filter, that is, if the filter value is *true* for that entry.

An instance of the OM class Filter has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values	Initial Value
Filter Items	Object(Filter Item)	0 or more	–
Filter	Object(Filter)	0 or more	–
Filter Type	Enum(Filter-Type)	1	*and*

Filter Items (DS_FILTER_ITEMS)

A collection of assertions each relating to one attribute of an entry.

Filters (DS_FILTERS)

A collection of filters.

Filter Type (DS_FILTER_TYPE)

The filter's type. Its value is one of the following:

- *and* (DS_AND)

  The filter is the logical connection of its components. The filter is *true* if all nested filters or filter items are *true* or if there are no nested filter items.

- *or* (DS_OR)

  The filter is *true* if any of the nested filters or filter items are *true*. The filter is *false* if there are no nested components.

- *not* (DS_NOT)

  The result of the filter is reversed. There must be exactly one nested filter or filter item. The filter is *true* if the enclosed filter or filter item is *false*. The filter is *false* if the enclosed filter or filter item is *true*.

3.17. Filter Item

DS_C_FILTER_ITEM

An instance of class Filter Item is a component of a filter. It is an assertion about the existence or values of a single attribute type in an entry. The DSA determines the value of the Filter Item. The value is *undefined* if any of the following are true:

- The Attribute Type is not known.
- Any of the Attribute Values do not conform to the attribute syntax defined for that attribute type.
- The Filter Item Type uses a matching rule that is not defined for the attribute syntax.

It has the attributes of its superclasses, Object and Attribute, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Filter Item Type	Enum(Filter-Item-Type)	1
Final Substring	String[1]	0 or 1
Initial Substring	String[1]	0 or 1

[1]The string cannot be of zero length

Filter Item Type (DS_FILTER_ITEM_TYPE)

Identifies the type of the Filter Item and therefore the nature of the filter. Its value must be one of the following:

- *approximate-match* (DS_APPROXIMATE_MATCH)

  The filter item is *true* if the entry contains at least one value of the specified type that is approximately equal to the specified value. There must be exactly one value of the attribute Attribute Values.

  The matching rules for *approximate-match* are defined by the DSA implementation and vary according to the attribute syntax.

- *equality* (DS_EQUALITY)

  The filter item is *true* if the entry contains at least one value of the specified type that is equal to the specified value. There must be exactly one value of the attribute Attribute Values.

  The matching rules for *equality* are defined by the DSA implementation and vary according to the attribute syntax.

- *greater-or-equal* (DS_GREATER_OR_EQUAL)

  The filter item is *true* if at least one value of the attribute is greater than or equal to the supplied value. There must be exactly one value of the attribute Attribute Values.

  The matching rules for *greater-or-equ al* are defined by the DSA implementation and vary according to the attribute syntax.

- *less-or-equal* (DS_LESS_OR_EQUAL)

  The filter item is *true* if at least one value of the attribute is less than or equal to the supplied value. There must be exactly one value of the attribute Attribute Values.

  The matching rules for *less-or-equal* are defined by the DSA implementation and vary according to the attribute syntax.

- *present* (DS_PRESENT)

The filter item is *true* if the entry contains an attribute of the specified type. Attribute Values are ignored.

The matching rules for *present* are defined by the DSA implementation and vary according to the attribute syntax.

- *substrings* (DS_SUBSTRINGS)

  The filter item is *true* if the entry contains at least one value of the specified type that contains all of the specified substrings in the given order. There can be any number of substrings given as values of Attribute Values, including none. Substrings must not overlap but they can be separated from each other or the ends of the attribute value by zero or more string elements.

  The matching rules for *substrings* are defined by the DSA implementation and vary according to the attribute syntax.

Final Substring (DS_FINAL_SUBSTRING)

The substring that must match the last portion of an attribute value in the entry.

Initial Substring (DS_INITIAL_SUBSTRING)

The substring that must match the first portion of an attribute value in the entry.

3.18. List Information

DS_C_LIST_INFO

An instance of the OM class List Information is a portion of the results of a List function call.
An application must not create or modify instances of this class. The class has the attributes of its superclasses, Object and Common Results, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Object Name	Object(Name)	0 or 1
Partial Outcome Qualifier	Object(Partial Outcome Qualifier)	0 or 1
Subordinates	Object(List Information Item)	0 or more

Object Name (DS_OBJ ECT NAME)

The Distinguished Name of the target object entry of the operation. It is present if the Alias Dereferenced attribute is *true*.

Partial Outcome Qualifier (DS_PARTIAL_OUTCOME_QUAL)

Is present if the list of subordinates is incomplete. It contains details of why the search was not completed and which areas of the Directory were not searched.

Subordinates (DS_SUBORDINATES)

Is information about zero or more subordinate objects identified by the List function.

3.19. List Information Item

DS_C_LIST_INFO_ITEM

An instance of the OM class List Information Item contains details of one subordinate object returned by the List function. An application must not create or modify instances of this class. The class has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Alias Entry	Boolean	1
From Entry	Boolean	1
RDN	Object(Relative Name)	1

Alias Entry (DS_ALIAS_ENTRY)

Indicates whether the subordinate object is an alias.

From Entry (DS_FROM_ENTRY)

Indicates whether information about the object was obtained from its entry or a copy of the entry.

RDN (DS_RDN)

The RDN of the object. If this is the name of an alias entry, it is not dereferenced.

3.20. List Result

DS_C_LIST_RESULT

An instance of the OM class List Result contains the results of a successful call to the List function. An application must not create or modify instances of this class. The class has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
List Information	Object(List Information)	0 or 1
Uncorrelated List Information	Object(List Result)	0 or more

List Information (DS_LIST_INFO)

The result of the List function, or a portion of it.

Uncorrelated List Information (DS_UNCORRELATED_LIST_INFO)

The information returned when the DUA has requested a protection request of Signed. This can consist of a number of sets of results originating from and signed by different components of the Directory.

DSA does not support this service.

3.21. Name

DS_C_NAME

An instance of the OM class Name represents a name of an entry in the Directory or part of that name. It is an abstract class that has the attributes of its superclass, Object, and no others.

A name unambiguously distinguishes the object from all other objects whose entries appear in the DIT. However, an object may have more than one real- world name, that is, a name need not be unique. A distinguished name is unique; there are no other distinguished names that identify the same object. A relative distinguished name is a part of a name, and only distinguishes the object from others that are its siblings. Most of the interface functions take a name argument, the value of which must be an instance of one of the subclasses of this OM class. Thus, this OM class serves to collect together all possible representations of names.

This guide defines one subclass of this OM class, Distinguished Name, which provides a single representation for names, including distinguished names.

3.22. Operation Progress

DS_C_OPERATION_PROGRESS

An instance of the OM class Operation Progress specifies the progress or processing state of a directory request. An application must not create or modify instances of this class. The class has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Name Resolution Phase	Enum(Name-Resolution-Phase)	1
Next RDN To Be Resolved	Integer	0 or 1

The target name mentioned below is the name upon which processing of the directory request is currently focu sed.

Name Resolution Ph ase (DS_NAME_RESOLUTION_PHASE)

Indicates what phase has been reached in handling the target name. Its value is one of:

- *Completed* (DS_COMPLETED)

  The DSA holding the target object has been reached.

- *Not-Started* (DS_NOT_STARTED)

  A DSA with a naming context containing the initial RDN(s) of the name has not yet been reached.

- *Proceeding* (DS_PROCEEDING)

  The initial part of the name has been recognized but the DSA holding the target object has not yet been reached.

Next RDN To Be Resolved (DS_NEXT_RDN_TO_BE_RESOLVED)

Indicates to the DSA which of the RDNs in the target object is next to be resolved. It is an integer that ranges from one to the number of RDNs in the name. This attribute only has a value if the value of Name Resolution Phase is *Proceeding*.

The constant Operation Not Started (DS_OPERATION_NOT_STARTED) may be used in the Context of an operation instead of an instance of this OM class (see *Section 3.8, "Context"*).

3.23. Partial Outcome Qualifier

DS_C_PARTIAL_OUTCOME_QUAL

An instance of the OM class Partial Outcome Qualifier explains how incomplete the results of a call to the List or Search functions are and why. An application must not create or modify instances of this class. This class has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Limit Problem	Enum(Limit-Problem)	1
Unavailable Critical Extensions	Boolean	1
Unexplored	Object(Continuation Reference)	0 or 1

Limit Problem (DS_LIMIT_PROBLEM)

If present, explains why the results are incomplete. Its value is one of the following:

- *Administrative-Limit-Exceeded* (DS_ADMIN_LIMIT_EXCEEDED)

  An administrative limit was reached.

- *No-Limit-Exceeded* (DS_NO_LIMIT_EXCEEDED)

  There was no limit problem.

- *Size-Limit-Exceeded* (DS_SIZE_LIMIT_EXCEEDED)

  The maximum number of objects specified as a service control was reached.

- *Time-Limit-Exceeded* (DS_TIME_LIMIT_EXCEEDED)

  The maximum number of seconds specified as a service control was reached.

Unavailable Critical Extensions (DS_UNAVAILABLE_CRIT_EXT)

If *true*, indicates that some part of the Directory cannot provide a requested critical service extension. The user requested one or more standard service extensions, by including values of the OM attribute Extensions in the Context supplied for the operation and additionally, indicated some of them to be essential by setting the OM attribute Critical in the extension to be *true*.

Unexplored (DS_UNEXPLORED)

Identifies any parts of the Directory that were not explored. This allows the directory request to be continued. Only continuation references within the scope specified by the Scope Of Referral Context attribute are included.

3.24. Presentation Address

DS_C_PRESENTATION_ADDRESS

An instance of the OM class Presentation Address is a presentation address of an OSI application entity used for OSI communications. It has the attributes of its superclasses, Object and Address, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
N-Addresses	String(Octet)	1 or more
P-Selector	String(Octet)	0 or 1
S-Selector	String(Octet)	0 or 1
T-Selector	String(Octet)	0 or 1

N-Addresses (DS_N_ADDRE SSES)

The network address of the application entity.

P-Selector (DS_P_SELECTOR)

The presentation selector.

S-Selector (DS_S_SELECTOR)

The session selector.

T-Selector (DS_T_SELECTOR)

The transport selector.

3.25. Read Result

DS_C_READ_RESULT

An instance of the OM class Read Result contains the result of a successful call to the Read function. An application must not create or modify instances of this class. The class has the attributes of its superclasses, Object and Common Results, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values
Entry	Object(Entry Information)	1

Entry (DS_ENTRY)

The information extracted from the entry of the target object.

3.26. Relative Distinguished Name

DS_C_DS_RDN

An instance of the OM class Relative Distinguished Name is an RDN. An RDN uniquely identifies an immediate subordinate of an object whose entry is in the DIT. It has the attributes of its superclasses, Object and Relative Name, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values
AVAs	Object(AVA)	1 or more

AVAs (DS_AVAS)

The OM attribute AVAs indicates the AVAs marked by the Directory as components of the entry's RDN. The assertions are true of the object but not of its siblings. The values they contain are in the object's entry. The order of the AVAs is not significant.

3.27. Relative Name

DS_C_RELATIVE_NAME

An instance of the OM class Relative Name represents the RDN of objects in the Directory. It is an abstract class that has the attributes of its superclass, Object, and no others.

It has a single subclass, Relative Distinguished Name, which provides a representation for RDNs.

3.28. Search Information

DS_C_SEARCH_INFO

An instance of the OM class Search Information contains part of the result of a successful call to the Search function. An application must not create or modify instances of this class. The class has the attributes of its superclasses, Object and Common Results, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Entries	Object(Entry Information)	0 or more
Object Name	Object(Name)	0 or 1
Partial Outcome Qualifier	Object(Partial Outcome Qualifier)	0 or 1

Entries (DS_ENTRIES)

Contains information about zero or more objects foun d by the Search function that match the given selection criteria.

Object Name (DS_OBJ ECT_NAME)

The Distinguished Name of the target object of the operation. It is present if the attribute Alias Dereferenced is *true*.

Partial Outcome Qualifier (DS_PARTIAL_OUTCOME_QUAL)

This is only present if the list of entries is incomplete. It contains details of why the search was not completed and which areas of the Directory were not searched.

3.29. Search Result

DS_C_SEARCH_RESULT

An instance of the OM class Search Result contains the result of a successful call to the Search function. An application must not create or modify instances of this class. The class has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Search Information	Object(Search Information)	0 or 1
Uncorrelated Search Information	Object(Search Result)	0 or more

Search Information (DS_SEARCH_INFO)

The result of the Search function, or a portion of it.

Uncorrelated Search Information (DS_UNCORRELATED_SEARCH_ INFO)

The information returned when the DUA has requested a protection request of Signed. This can consist of a number of sets of results originating from and signed by different components of the Directory.

3.30. Session

DS_C_SESSION

An instance of the OM class Session identifies a particular link from the application to the DUA. It has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name Value Syntax

Number of

Values Initial Value

OM Attribute Name	Value Syntax	Number of Values	Initial Value
DSA Address	Object(Address)	0 or 1	none[1]
DSA Name	Object(Name)	0 or 1	none[1]
File Descriptor [b]	Integer	1	No_Valid_File_Descriptor
Requestor	Object(Name)	0 or 1	none[1]
Password	String(Object)	0 or 1	none[1]

[1]You can use the DUA defaults file to configure this. See *Section 2.3, "Session"* for more information.

[b]On an OpenVMS system, the value of the File Descriptor parameter is always No-Valid-FileDescriptor. No-Valid-File-Descriptor is also the value if you are bound only to a CDS directory.

DSA Address (DS_DSA_ADDRE SS)

Indicates the address of the default DSA named by DSA Name.

DSA Name (DS_DSA_NAME)

Indicates the Distinguished Name of the DSA that is used by default to service directory requests.

File Descriptor (DS_FILE_DESCRIPTOR)

Indicates the file descriptor associated with the session.

Requestor (DS_REQUESTOR)

The Distinguished Name of the user of this directory session.

Password (DSX_PASSWORD)

This is the password that allows the user access to the directory. This is a Digital extension to XDS.

Applications can assume that an object of OM Class Session, created with default values of all its OM attributes, will work with all the interface functions. You should ensure that this is the case. Such a session can be created by passing the constant Default Session (DS_DEFAULT_S ESSION) as an argument of the Bind function. *Section 2.3, "Session"* contains information about the Default Session constant and defining default values.

Chapter 4. Interface Functions

This chapter describes, in alphabetic order, the X.500 API functions. It explains what the functions do, describes the function arguments and return values, and lists the errors that can occur for each function.

Call these functions to use the services provided by the X.500 API. See *VSI X.500 Directory Service Programming* for further information on using the functions with an X.500 directory. The DCE Notes section of the function descriptions gives brief details of how the function is used with CDS. See *DCE for DEC OSF/1 AXP Product Guide* for information about using this interface with CDS.

The behavior of the interface is undefined if an argument to a function has an invalid value. This only applies if a specific error condition is not defined. *Chapter 5, "Errors"* contains details of the errors returned by the interface functions.

Note that for many of the example programs shown, the return code of a function call is often not checked. This omission, and others, are intentional to ensure clarity and brevity within the example programs. The programs have been written to show as much important detail as possible, while remaining correct and concise.

The next section is a general introduction to the interface functions; the rest of this chapter contains descriptions of the interface functions.

4.1. X/OPEN Directory Services (XDS) Functions

ds_intro

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

Format

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

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

Description

This reference page lists the XDS interface functions supported in this 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.

Function	Description
ds_list	Enumerates the immediate subordinates of a particular directory entry.
ds_modify_entry	Perform s 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 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 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 ServiceError *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

ds_abandon — This function abandons an outstanding asynchronous operation.

Format

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

Argument	Data Type	Access
Session	OM_private_object	read
Invoke-ID	Integer	read
Status	DS_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.

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, even if the function returns an error.

DCE Notes

The DCE XDS interface does not support asynchronous operations.

Return Values

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

```
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");
    }

}
```

ds_add_entry

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

Format

`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

Argument	Data Type	Access
Invoke-ID	Integer	write
Status	DS_status	

C Binding

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

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-Alia ses 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 conform s 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_OBJ ECT_CLASS attribute of an object is single valued in CDS and multivalued in X.500. If the Entry argument contains a DS_A_ OBJ ECT_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_OBJ ECT_ 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_OBJ ECT_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 Values

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-con text*, *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.

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;
            }
        }
    }
}
```

Example 1 shows:

- How to define a private object containing context parameters.
- How to define a public object (cp ub_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 (cp ub_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.

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");
    }
}
```

Example 2 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

ds_bind — Opens a session with the directory service.

Format

`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, 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 Values

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

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

Format

`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-Alia ses.

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 foun d 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 between operations 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_OBJ ECT_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 Values

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 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-con text*, *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.

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;
        }
    }
}
```

Example 1 shows:

- How to define an attribute value assertion and use that in the Compare function.
- How to define an AVA OM public object (cp ub_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 (cp ub_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.

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 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 exactly as shown.

ds_initialize

ds_initialize — Initializes the interface.

Format

`Workspace = ds_initialize (void)`

Argument	Data Type
Workspace	OM_workspace

C Binding

```
OM_workspace ds_initialize (void)
```

Description

This function perform s 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 which is distinct from other workspaces created by the Initialize function but not yet deleted by the Shutdown function.

Return Values

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

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

Format

`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-Alia ses.

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 Values

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-con text*, *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.

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;
     }
  }
}
```

Example 1 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.

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
    {...}
}
```

Example 2 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

ds_modify_entry — Perform s an modification on an entry.

Format

`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 conform s 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 Values

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-con text*, *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 non-existent attribute type
- To use Remove-Attribute to remove a non-existent attribute or non-existent 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 "Abacu s 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.

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;
            }
        }
    }
}
```

Example 1 shows the following:

- How to define an Entry-Modification-List OM public object (cp ub_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 (cp ub_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.

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");
    }
}
```

Example 2 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

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

Format

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

Argument	Data Type	Access
Session	OM_private_object	read
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 ServiceError *unwilling-to-perform*.

Return Values

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 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-con text*, *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-m ultiple-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

ds_read — Queries information in a particular entry.

Format

`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-Alia ses.

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-An d-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 Values

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 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- attribute*, *bad-con text*, *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.

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;
            }
        }
    }
}
```

Example 1 shows:

- How to define a private object containing a distinguished name.

- How to define a DS-DN OM public object (cp ub_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.

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");
     }
}
```

Example 2 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.

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");
    }
}
```

Example 3 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_TYPELE SS_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_TYPELE SS_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

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

Format

`Status = ds_receive_result (Session, Completion-Flag, Operation-Status, Resul`

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 has completed and its result is available.
- Outstanding-Operations. There are outstanding asynchronous operations but none has 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 *VSI X.500 Directory Service Programming* for information about calling functions asynchronously.

DCE Notes

The DCE XDS interface does not support asynchronous operations.

Return Values

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*10

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

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

Format

`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-Alia ses 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 Values

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 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-con text*, *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.

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;
                }
            }
        }
    }
```

Example 1 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.

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;
}
```

Example 2 shows a synchronous call to the Remove Entry function. Theoperation being performed is the same as that shown in Example 1, theonly difference being that this operation completes immediately and theresult is contained in the Status argument. The Invoke-ID argument is notneeded and is therefore set to NULL.

ds_search

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

Format

`Status = ds_search (Session, Context, Name, Subset, Filter, Search_Aliases, S`

Argument	Data Type	Access
Session	OM_private_object	read
Context	OM_private_object	read
Name	OM_object	read
Subset	Integer	read
Filter	OM_object	read
Search_Aliases	OM_boolean	read

Argument	Data Type	Access
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 to 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

A Filter OM object, specifying a filter 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*.

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-An d-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.

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

DCE Notes

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

Return Values

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 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-con text*, *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 foun d 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.

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;
        }
    }
}
```

Example 1 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.

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
    {...}
}
```

Example 2 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

ds_shutdown — Shuts down the interface and closes the workspace.

Format

`Status = ds_shutdown (Workspace)`

Argument	Data Type	Access
Workspace	OM_workspace	read
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 Values

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_shutdown

ds_shutdown — This function closes a directory session.

Format

`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 theSession OM private object that was returned by the Bind function, identifyingthe directory session. If the function succeeds, the value of the File-DescriptorOM attribute is *No-Valid-File-Descriptor*. The other OM attributes areunchanged.

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 Values

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*.

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

ds_version — Negotiates the features of the interface and service.

Format

`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 which 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. VSI's implementation of this function does not support these packages, 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 Values

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 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 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

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

Format

(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

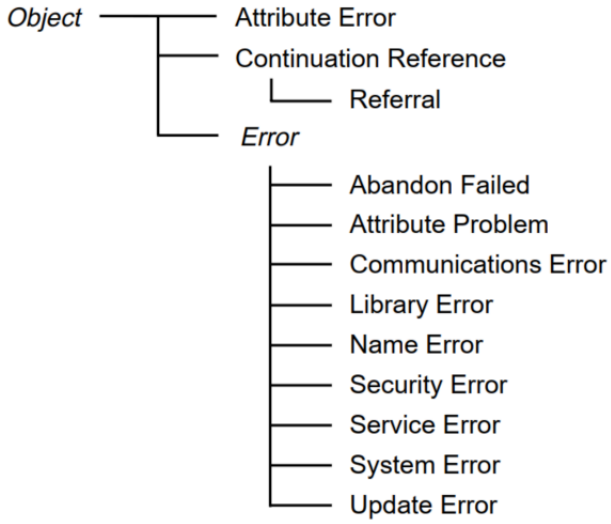
The following code extract shows an example call to the Trace Object function:

```
{
  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 5. Errors

This chapter defines the errors that can arise in the use of the X.500 API and the method used to report them. It also describes the hierarchical organization of the Object Management (OM) classes defined in this chapter and shows how the classes inherit OM attributes from superclasses. *Figure 5.1, "Hierarchy of Error Object Classes"* shows the hierarchy of error object classes with the names of abstract OM classes in italics.

Figure 5.1. Hierarchy of Error Object Classes



The object class Object is defined in the *OSI-Abstract-Data Manipulation*. It contains one attribute, Class, which denotes the class of the object.

The following sections contain descriptions of the OM classes used to define errors. Each OM class is described in a separate section that identifies the OM attributes specific to that class and lists its superclasses. For each class, there is a table that defines the class-specific attributes, if there are any. The tables give the following information:

- The name of each attribute
- The syntax of each attribute value (including any appropriate object class, enumeration syntax or string type in parenthesis)
- The number of values each attribute can have
- The initial value, if any, that the OM-Create function supplies (if initialization is requested)

Applications must not create or modify instances of any of these classes.

5.1. Error

DS_C_ERROR

This class contains the parameters common to all errors. Error is an abstract class that has the attributes of its superclass, Object, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values
Problem	Enum(Problem)	1

Problem (DS_PROBLEM)

Provides details of the error. A number of possible values are defined, each of which is described under the appropriate error class.

5.2. Abandon Failed

DS_C_ABANDON_FAILED

An instance of the class Abandon Failed reports an error during an attempt to abandon an operation. It has the attributes of its superclass, Object and Error, and no other attributes.

The OM attribute Problem inherited from the superclass Error has the following values:

- *Cannot Abandon* (DS_E_CANNOT_ABANDON)

  The operation does not allow an abandon operation on it, or the operation could not be abandoned for an unspecified reason.

- *No Such Operation* (DS_E_NO_SUCH_OPERATION)

  The Directory has no knowledge of the operation that was the subject of the abandon request.

- *Too Late* (DS_E_TOO_LATE)

  The operation has already completed.

5.3. Attribute Error

DS_C_ATTRIBUTE_ERROR

An instance of the class Attribute Error reports attribute-related directory errors. It has the attributes of its superclass, Object, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Object Name	Object(Name)	1
Problems	Object(Attribute Problem)	1 or more

Object Name (DS_OBJ ECT_NAME)

The name of the directory entry to which the operation was being applied when the failure occurred.

Problems (DS_PROBLEMS)

Lists the attribute-related problems that occurred. An Attribute Error can report several problems at once, all related to the named object.

5.4. Attribute Problem

DS_C_ATTRIBUTE_PROBLEM

An instance of the class Attribute Problem describes a problem that is attribute-related. It has the attributes of its superclasses, Object and Error, and the following attributes:

OM Attribute Name	Value Syntax	Number of Values
Attribute Type	String(Object Identifier)	1
Attribute Value	–	0 or 1

Attribute Type (DS_ATTRIBUTE_TYPE)

Identifies the type of the attribute associated with the problem.

Attribute Value (DS_ATTRIBUTE_VALUE)

The attribute value associated with the problem. It is present only if required to avoid ambiguity. Its syntax is determined by Attribute Type.

The OM attribute Problem inherited from the superclass Error has one of the following values:

- *Attribute or Value Exists* (DS_E_ATTRIBUTE_OR_VALUE_EXISTS) The attribute or attribute value to be added is already present in the specified entry.
- *Constraint Violation* (DS_E_CONSTRAINT_VIOLATION)

  The attribute or attribute value does not conform to constraints imposed by the Directory or by the attribute definition.
- *Inappropriate Matching* (DS_E_INAPPROP_MATCHING) The matching rule used is not defined for the attribute type.
- *Invalid Attribute Syntax* (DS_E_INVALID_ATTRIBUTE_SYNTAX)

  A value presented as an argument does not conform to the attribute syntax of the attribute type.
- *No Such Attribute Or Value* (DS_E_NO_SUCH_ATTRIBUTE_OR_VALUE)

  The specified attribute or attribute value was not foun d in the specified entry. This is reported by the Read or Search functions if an explicit list of attributes is specified in the Selection argument, and none of them are present in the entry.
- *Undefined Attribute Type* (DS_E_UNDEFINED_ATTRIBUTE_TYPE)

  The attribute type, which was supplied as an argument to either the Add Entry or Modify Entry function, is undefined.

5.5. Communications Error

DS_C_COMMUNICATIONS_ERROR

An instance of the class Communications Error reports errors occurring in other OSI services supporting the Directory Service. Communications errors include those occurring in Remote Operation, Association Control, Presentation, Session and Transport. It has the attributes of its superclasses, Object and Error, and the attributes listed below.

DSX_INVOKE_ID
DSX_COMM_PROBLEM

DSX_OSAK_STATUS1
DSX_OSAK_STATUS2
DSX_TRANSPORT_STATUS1
DSX_TRANSPORT_STATUS2
DSX_ABORT_REASON
DSX_RE J ECT_REASON
DSX_UNEXPECTED_EVENT_CODE
DSX_PDU

The attribute Problem inherited from the superclass Error indicates the type of error. For most problem codes there are attributes that contain further information. The following table shows the problem codes and the attributes that contain further information:

Problem Code	**Related Attributes**	**Attribute Type**
DS_E_COMMUNICATIONS_PROBLEM	DSX_OSAK_STATUS1 DSX_OSAK_STATUS2 DSX_TRANSPORT_STATUS1 DSX_TRANSPORT_STATUS1	Integer
DSX_E_TRANSMIT_PROBLEM	DSX_OSAK_STATUS1 DSX_OSAK_STATUS2 DSX_TRANSPORT_STATUS1 DSX_TRANSPORT_STATUS1	Integer
DSX_E_RECEIVE_PROBLEM	DSX_OSAK_STATUS1 DSX_OSAK_STATUS2 DSX_TRANSPORT_STATUS1 DSX_TRANSPORT_STATUS1	Integer
DSX_E_OSAK_ABORT	DSX_ABORT_REASON	Integer
DSX_E_OSAK_REJECT	DSX_REJECT_REASON	Integer
DSX_E_UNEXPECTED_OSAK_EVENT	DSX_UNEXPECTED_EVENT_ CODE	Integer
DSX_E_DECODE_FAILED	DSX_PDU	Integer
DSX_E_BAD_INVOKE_ID	DSX_INVOKE_ID	Pointer to PDU
DSX_E_NO_MEMORY	none	OM_sint
DSX_E_ROSE_REJECT	DSX_REJECT_REASON	Integer
DSX_E_ROSE_INVOKE_PROBLEM	DSX_REJECT_REASON	Integer

For information about these OSAK™ events, see the *OSI Applications Kernel Programming Reference* manual.

5.6. Library Error

DS_C_LIBRARY_ERROR

An instance of the class Library Error reports errors detected by the interface function library. It has the attributes of its superclasses, Object and Error, and no other attributes. Each function has several possible errors that can be detected by the library and are returned directly by the subroutine. These errors occur

when the library cannot perform an action in submitting a service request or in deciphering a response from the Directory.

The OM attribute Problem inherited from the superclass Error has one of the following values:

- *Bad Argument* (DS_E_BAD_ARGUMENT)

  An invalid argument, other than Name, was supplied. Supplying an instance of OM class Attribute with no values of the OM attribute Attribute-Values as an input argument to an X.500 API function always results in this error, because directory attributes always have at least one value.

- *Bad Class* (DS_E_BAD_CLASS)

  The class of an argument is not supported for this operation.

- *Bad Context* (DS_E_BAD_CONTEXT)

  An invalid context argument was supplied.

- *Bad Name* (DS_E_BAD_NAME)

  An invalid name argument was supplied.

- *Bad Session* (DS_E_BAD_SESSION)

  An invalid session argument was supplied.

- *Miscellaneous* (DS_E_MISCELLANEOUS)

  A miscellaneous error occurred during interaction with the Directory. This error is returned if the interface cannot clear a transient system error by retrying the affected system call.

- *Missing Type* (DS_E_MISSING_TYPE)

  The attribute type was not included in an AVA that was passed as part of a Distinguished Name argument.

- *Not Supported* (DS_E_NOT_SUPPORTED)

  The requested, optional functionality is not available in this implementation.

- *Too Many Operations* (DS_E_TOO_MANY_OPERATIONS)

  No more directory operations can be performed until at least one asynchronous operation has completed.

- *Too Many Sessions* (DS_E_TOO_MANY_S ESSIONS)

  No more directory sessions can be started.

5.7. Name Error

DS_C_NAME_ERROR

An instance of the class Name Error reports name-related directory errors. It has the attributes of its superclasses, Object and Error, and the following attribute:

OM Attribute Name	Value Syntax	Number of Values
Matched	Object(Name)	1

Matched (DS_MATCHED)

The first part of the name up to, but not including, the first unrecognized RDN. This name is either the one that was supplied or the one resulting from the dereferencing of an alias. This OM Name object names the lowest entry or alias entry in the DIT that was matched.

The OM attribute Problem inherited from the superclass Error has one of the following values:

- *Alias Dereferencing Problem* (DS_E_ALIAS_DEREFERENCING_PROBLEM)

  An alias was foun d where aliases are not allowed. This could be one of the following:

  - An alias in a modification operation
  - An alias when the Dont Dereference Aliases Context parameter is set
  - An alias pointing to another alias

- *Alias Problem* (DS_E_ALIAS_PROBLEM)

  An alias has been dereferenced that names an object that does not exist.

- *Invalid Attribute Value* (DS_E_INVALID_ATTRIBUTE_VALUE)

  The attribute value, in an AVA in an RDN in the name, does not conform to the attribute syntax for the attribute type in the AVA.

- *No Such Object* (DS_E_NO_SUCH_OBJ ECT)

  The specified name does not match the name of any object in the Directory.

5.8. Referral

DS_C_REFERRAL

An instance of the class Referral reports a failure to perform an operation and redirects the requestor to one or more access points better equipped to perform the request. A Referral has the attributes of its superclasses, Object and Continuation Reference, and no other attributes.

5.9. Security Error

DS_C_SECURITY_ERROR

An instance of the class Security Error reports security-related Directory errors. It has the attributes of its superclasses, Object and Error, and no other attributes.

The OM attribute Problem inherited from the superclass Error has one of the following values:

- *Inappropriate Authentication* (DS_E_INAPPROP_AUTHENTICATION)

  The level of security attached to the requestor's credentials is inconsistent with the level of protection requested.

- *Insufficient Access Rights* (DS_E_INSUFFICIENT_ACCESS_RIGHTS) The requestor is not permitted to perform the operation.
- *Invalid Credentials* (DS_E_INVALID_CREDENTIALS) The requestor's credentials are invalid.
- *Invalid Signature* (DS_E_INVALID_SIGNATURE) The signature joined to the request is invalid.
- *No Information* (DS_E_NO_INFO)

  The request has produced a security error but no other information is available.
- *Protection Required* (DS_E_PROTECTION_REQUIRED)

  The Directory cannot perform the operation because it was not signed.

5.10. Service Error

DS_C_SERVICE_ERROR

An instance of the class Service Error reports a service-related Directory error. It has the attributes of its superclasses, Object and Error, and no other attributes.

The OM attribute Problem inherited from the superclass Error has one of the following values:

- *Administrative Limit Exceeded* (DS_E_ADMIN_LIMIT_EXCEEDED)

  The operation could not be performed within the administrative constraints on the Directory, and no partial results are available.
- *Busy* (DS_E_BUSY)

  Some part of the Directory is temporarily too busy to perform the operation but might be able to perform it soon.
- *Chaining Required* (DS_E_CHAINING_REQUIRED)

  Chaining is required for the operation to be performed but is prohibited by the Chaining Prohibited Context parameter.
- *DIT Error* (DS_E_DIT_ERROR)

  An inconsistency has been detected in the DIT. This is possibly localized in an entry or set of entries.
- *Invalid Reference* (DS_E_INVALID_REF)

  The DSA was unable to perform the request as directed through the Operations Progress Context parameter. This is possibly because of an invalid referral.
- *Loop Detected* (DS_E_LOOP_DETECTED)

  The DSA detected a loop within the Directory.
- *Out Of Scope* (DS_E_OUT_OF_SCOPE)

  The Directory cannot provide a Referral or Partial Outcome Qualifier within the required scope.
- *Time Limit Exceeded* (DS_E_TIME_LIMIT_EXCEEDED)

The operation could not be performed within the specified time limit and no partial results are available.

- *Unable To Proceed* (DS_E_UNABLE_TO_PROCEED)

  A DSA was asked to resolve a name within a particular naming context over which it has no administrative authority.

- *Unavailable* (DS_E_UNAVAILABLE)

  Some part of the Directory is not currently available.

- *Unavailable Critical Extension* (DS_E_UNAVAILABLE_CRIT_EXT)

  One, or more, critical extensions were requested but were not available.

- *Unwilling To Perform* (DS_E_UNWILLING_TO_PERFORM)

  Some part of the Directory will not perform the operation because it requires excessive resources, or because it would violate administrative policy.

5.11. System Error

DS_C_SYSTEM_ERROR

An instance of the class System Error reports errors that occur in the operating system. It has the attributes of its superclasses, Object and Error, and no other attributes.

The OM attribute Problem inherited from the superclass Error has the same value as *errno* defined in the C programming language.

The current release of XDS does not report any system errors.

5.12. Update Error

DS_C_UPDATE_ERROR

An instance of the class Update Error reports a modification-related Directory error. It has the attributes of its superclasses, Object and Error, and no other attributes.

The OM attribute Problem inherited from the superclass Error has one of the following values:

- *Affects Multiple DSAs* (DS_E_AFFECTS_MULTIPLE_DSAS)

  The modification would affect several DSAs and is not allowed. Local agreement among DSAs might allow such modifications and the problem would not then be reported.

- *Entry Exists* (DS_E_ENTRY_EXISTS)

  The name passed to the Add Entry function already exists.

- *Naming Violation* (DS_E_NAMING_VIOLATION)

  The modification would leave the DIT improperly structured.

- *Not Allowed On Non Leaf* (DS_E_NOT_ALLOWED_ON_NON_LEAF)

  The modification would be to an interior entry of the DIT and this is not allowed.

- *Not Allowed On RDN* (DS_E_NOT_ALLOWED_ON_RDN)

  The modification would alter an entry's RDN.

- *Object Class Modification Prohibited* (DS_E_OBJ ECT_CLASS_MOD_ PROHIB)

  The modification would alter an entry's Object Class attribute.

- *Object Class Violation* (DS_E_OBJ ECT_CLASS_VIOLATION)

  The modification would leave the entry inconsistent with its Object Class definition.

Chapter 6. Directory Class Definitions

This chapter defines the attributes that make up directory entries.

The CCITT X.500 Series of Recommendations define a number of attribute types, attribute syntaxes, attribute sets, and object classes. The attribute types are known as selected attribute types, and the object classes are known as selected object classes. These definitions allow the creation and maintenance of Directory entries for a number of common objects. Therefore, the representation of such objects will be the same throughout the Directory. The definitions include such objects as Country, Organization and Person.

This chapter sets out names for each of these items, and defines OM classes to represent those which are not represented directly by OM syntaxes.

The constants and OM classes defined in this chapter are additional to those described in Chapter 3, since they are not essential to the working of the interface, but instead allow directory entries to be utilized. The definitions are further divided into three packages, each of which is optionally supported.

The first package is the Basic Directory Contents Package (BDCP) and contains all of the definitions except those concerned with strong authentication. The C constants associated with this package are in the xdsbdcp.h header file.

The second package is the Strong Authentication Package (SAP) which contains definitions for strong authentication. The C constants associated with this package are in the xdssap.h header file.

The third package is the MHS Directory User Package (MDUP) which contains definitions to support the use of the directory by 1988 X.400 User Agents and MTAs for the purposes of name resolution, distribution list expansion and capability assessment. The definitions are based upon the attribute types and syntaxes specified in the recommendation X.402, Annex A. The C constants associated with this package are in the xdsmdup.h header file.

Note that VSI's implementation of XDS does not support all these packages. Refer to the *VSI X.500 Directory Service Programming* for more information about XDS header files.

6.1. Selected Attribute Types

This section describes the attribute types defined in the standards for use in directory entries. Each Directory entry is made up of a number of attributes. Each of these attributes has an attribute type and one or more attribute values. The form of each attribute value is determined by the attribute syntax associated with the attribute type.

In the interface, attributes appear as instances of the OM class Attribute. Attribute types are represented as the value of the OM attribute Attribute- Type. Attribute values are represented as the values of the OM attribute Attribute-Values. Each attribute type has an object-identifier, assigned in the CCITT X.500 Standards, that is the value of the OM attribute Attribute-Type. In the interface, these object-identifiers are represented by constants. These constants have the name of the directory attribute prefixed by A_ for ease of identification. Therefore, the C constant starts with DS_A_.

In *Table 6.1, "Representation of Values for Selected Attribute Types"* the matching rules that the Directory uses for the particular attribute are indicated as follows:

- E indicates the rules for determining if two values are equal.
- S indicates the rules for determining if one value is a substring of another.

Table 6.1. Representation of Values for Selected Attribute Types

Attribute Type	Value Syntax	Value Length	Matching Rules	Package
A-Alia sed-Object-Name	Object(Name)	–	E	BDCP
A-Authority-Revoc-List	Object(Revoc-List)	–	–	SAP
A-Bu siness-Category	String(Teletex)	1-128	E,S	BDCP
A-CA-Cert	Object(Cert)	–	–	SAP
A-Cert-Revoc-List	Object(Revoc-List)	–	–	SAP
A-Common-Name	String(Teletex)	1-64	E,S	BDCP
A-Country-Name	String(Printable)	2	E	BDCP
A-Cross-Cert-Pair	Object(Cert-Pair)	–	–	SAP
A-Deliv-Content-Length	Integer	–	–	MDUP
A-Deliv-Content-Types	String(Object-Identifier)	–	–	MDUP
A-Deliv-E ITs	String(Object-Identifier)	–	–	MDUP
A-Descrip tion	String(Teletex)	1-1024	E,S	BDCP
A-Dest-I ndicator	String(Printable)	1-128	E,S	BDCP
A-DL-Member s	Object(OR-Name)	–	–	MDUP
A-DL-S ubmit-Perms	Object(DL-Submit-Perms)	–	–	MDUP
A-Facsimile-Phone-Nbr	Object(Facsimile-Phone-Nbr)	–	–	BDCP
A-I nternat-ISDN-Nbr	String(Numeric)	1-16	–	BDCP
A-Knowledge-I nfo	String(Teletex)	–	E,S	BDCP
A-Locality-Name	String(Teletex)	1-128	E,S	BDCP
A-Member	Object(Name)	–	E	BDCP
A-Message-S tore	Object(DS-DN)	–	–	MDUP
A-Object-Cla ss	String(Object-Identifier)	–	E	BDCP
A-OR-Add resses	Object(OR-Add ress)	–	–	MDUP
A-Org-Name	String(Teletex)	1-64	E,S	BDCP
A-Org-Unit-Name	String(Teletex)	1-64	E,S	BDCP
A-Owner	Object(Name)	–	E	BDCP
A-Phone-Nbr	String(Printable)	1-32	E,S	BDCP
A-Phy-Deliv-Off-Name	String(Teletex)	1-128	E,S	BDCP
A-Pos t-Office-Box	String(Teletex)	1-40	E,S	BDCP
A-Pos tal-Add ress	Object(Postal-Add ress)	–	E	BDCP

Attribute Type	Value Syntax	Value Length	Matching Rules	Package
A-Pos tal-Code	String(Teletex)	1-40	E,S	BDCP
A-Pref-Deliv-Method	Enum(Pref-Deliv-Method)	–	–	BDCP
A-Pref-Deliv-Methods	Enum(Delivery-Mode)	–	E	MDUP
A-Presentation-Add ress	Object(Presentation-Add ress)	– E		BDCP
A-Registered-Add ress	Object(Postal-Add ress)	–	–	BDCP
A-Role-Occu pant	Object(Name)	–	E	BDCP
A-Se arch-Guide	Object(Search-Guide)	–	–	BDCP
A-See-Also	Object(Name)	–	E	BDCP
A-Se rial-Nbr	String(Printable)	1-64	E,S	BDCP
A-S tate-Or-Prov-Name	String(Teletex)	1-128	E,S	BDCP
A-S treet-Add ress	String(Teletex)	1-128	E,S	BDCP
A-S upport-Applic-Context	String(Object-Identifier)	–	E	BDCP
A-S upp-Auto-Action s	String(Object-Identifier)	–	–	MDUP
A-S upp-Content-Types	String(Object-Identifier)	–	–	MDUP
A-S upp-Opt-Attributes	String(Object-Identifier)	–	–	MDUP
A-S urname	String(Teletex)	1-64	E,S	BDCP
A-Teletex-Term-Ide nt	Object(Teletex-Term-Ident)	–	–	BDCP
A-Telex-Nbr	Object(Telex-Nbr)	–	–	BDCP
A-Title	String(Teletex)	1-64	E,S	BDCP
A-User-Cert	Object(Cert)	–	-	SAP
A-User-Pa ssword	String(Octet)	0-128	–	BDCP
A-X121-Add ress	String(Numeric)	1-15	E,S	BDCP

All these attribute types are multi-valued except for the following that are single-valued:

- A-Alia sed-Object-Name
- A-Country-Name
- A-Deliverable-Content-Length
- A-Message-S tore
- A-Pref-Deliv-Methods
- A-Presentation-Add ress

The following list describes all the attribute types from *Table 6.1, "Representation of Values for Selected Attribute Types"*.

- A-Aliased-Object-Name

This attribute occurs only in alias entries. It gives the Distinguished Name of the object that is given an alias by the entry in which this attribute occurs. An alias is an alternative to an object's distinguished name. Any object may (but need not) have one or more aliases. The directory is said to dereference an alias whenever it replaces the alias with the distinguished name associated with it by means of this attribute.

- A-Authority-Revoc-List

  This attribute occurs in entries describing a Certification Authority. It lists all the certificates issued to any Certification Authority known to this authority that were later revoked.

- A-Business-Category

  Describes the businesses in which the object is engaged.

- A-CA-Cert

  The certificate assigned to the object. The object is a Certification Authority.

- A-Cert-Revoc-List

  This attribute occurs in entries that describe a Certification Authority. It is the list of certificates issued by this authority and later revoked.

- A-Common-Name

  The names by which the object is commonly known in the context defined by its position in the DIT.

- A-Country-Name

  Identifies the country in which the object is located or with which it is associated. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

- A-Cross-Cert-Pair

  One or two certificates held in the entry of a Certification Authority. One is the certificate of one Certification Authority guaranteed by a second authority. The other is the certificate of the second authority guaranteed by the first authority.

- A-Deliv-Content-Length

  Identifies the maximum content length of messages that can be delivered to a user.

- A-Deliv-Content-Types

  Identifies the content types of messages that can be delivered to a user.

- A-Deliv-E ITs

  Identifies the Encoded Information Types (EITs) of messages that can be delivered to a user.

- A-Description

  The informal description of the object.

- A-Dest-Indicator

Country-city pairs that enable the object to be reached through the public telegram service. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

- A-DL-Members

  Identifies the members of a distribution list (DL).

- A-DL-Submit-Perms

  Identifies the users and distribution lists (DLs) that are permitted to submit messages to a distribution list.

- A-Facsimile-Phone-Nbr

  Denotes the telephone numbers for facsimile terminals through which the object can be reached or with which it is associated. This attribute can also include the parameters of the facsimile terminal.

- A-Internat-ISDN-Nbr

  The International Integrated Services Digital Network (ISDN) number through which the object can be reached or with which it is associated. The matching rules require that differences caused by the presence of spaces are considered insignificant.

- A-Knowledge-Info

  This attribute occurs in entries that describe a DSA. It provides an informal description of the directory knowledge that the DSA contains.

- A-Locality-Name

  Identifies geographical areas or localities. If used as part of a directory name, it indicates the localities in which the object is located or with which it is associated.

- A-Member

  Indicates names of objects that are considered to be members of the present object (which might be for example, a distribution list for electronic mail).

- A-Message-Store

  Identifies the user’s message store (MS) by name.

- A-Object-Class

  Identifies the object classes to which the object belongs and also identifies their superclasses. This attribute must be present in every entry and must not be modified.

- A-OR-Name

  Identifies the MHS O/R address of a user or a distribution list.

- A-Org-Name

  Identifies an organization with which the object is connected.

- A-Org-Unit-Name

Identifies the part of the organization with which the object is connected.

- A-Owner

  Indicates the names of the objects that are responsible for the object.

- A-Phone-Nbr

  Identifies telephones through which the object can be reached or with which it is associated.

- A-Phy-Deliv-Office-Name

  The names of cities, towns, or villages that contain physical delivery offices through which the object can take delivery of physical mail.

- A-Post-Office-Box

  Identifies post office boxes at which the object can take delivery of physical mail.

- A-Postal-Add ress

  Indicates postal addresses at which the object can take delivery of physical mail. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

- A-Postal-Code

  Indicates postal codes assigned to the area or building where the object can take delivery of physical mail.

- A-Pref-Deliv-Method

  The methods of communication preferred by the object. They are ordered so that the most preferred method comes first.

  - Any-Deliv-Method. No preference.
  - G3-Facsimile-Deliv. Through Group 3 facsimile.
  - G4-Facsimile-Deliv. Through Group 4 facsimile.
  - IA5-Terminal-Deliv. Through IA5 text.
  - MHS-Deliv. Through Electronic Messaging Systems based on the CCITT X.400 series of Recommendations.
  - Physical-Deliv. Through the postal or other physical delivery system.
  - Telephone-Deliv. By telephone.
  - Teletex-Deliv. Through teletex.
  - Telex-Deliv. Through telex.
  - Videotex-Deliv. Through videotex.

- A-Pref-Deliv-Methods

Identifies the methods of delivery preferred by the user. They are ordered so that the most preferred method comes first.

- A-Presentation-Address

  The presentation address of the object which is an application entity. The matching rule for a presented value to match a value stored in the directory is that the P-Selector, S-Selector and T-Selector of the two Presentation Addresses must be equal and the N-Add resses of the presented value must be a subset of those of the stored value.

- A-Registered-Address

  Indicates the mnemonics that enable the object to be reached through the public telegram service. It identifies the object in the context of a particular city and is registered in the country containing that city. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

- A-Role-Occupant

  This attribute occurs in entries that describe an organizational role. It gives the names of objects that fulfill this role.

- A-Search-Guide

  Indicates criteria that can be used to build filters for searches in which the object is the base.

- A-See-Also

  Names of objects that represent other aspects of the real world object represented by the present object.

- A-Serial-Nbr

  Indicates serial numbers of a device.

- A-State-Or-Province-Name

  Specifies a state, province or other geographical region in which the object is located or with which it is associated.

- A-Street-Address

  Identifies the street address where the object is located or with which it is connected.

- A-Support-Appl-Context

  This attribute occurs in entries that describe an application entity. It identifies the application contexts supported by the object.

- A-Supported-Automatic-Actions

  Identifies the automatic actions that a message store fully supports.

- A-Supported-Content-Types

  Identifies the content type of the messages whose syntax and semantics a message store fully supports.

- A-Supported-Optional-Attributes

  Identifies the optional attributes that a message store fully supports.

- A-Surname

  This attribute occurs in entries that describe a person. It is the surname by which the person is commonly known.

- A-Teletex-Term-Ident

  Descriptions of teletex terminals through which the object can be reached or with which it is associated.

- A-Telex-Nbr

  Descriptions of telex terminals through which the object can be reached or with which it is associated.

- A-Title

  Identifies the position or function of a person within an organization.

- A-User-Cert

  The user certificates assigned to the object. This can be any user certificate including a Certification Authority certificate.

- A-User-Password

  The passwords assigned to the object.

- A-X121-Address

  Identifies points on the public data network where the object can be reached or with which it is associated. The matching rules require that differences caused by the presence of spaces are considered insignificant.

6.2. Selected Object Classes

This section lists the object classes that are defined in the standards. Each Directory entry belongs to an object class identified by the attribute Object-Class. The values of this attribute are object identifiers. These are represented in the interface by constants with the same name as the object class prefixed by O_ for ease of reading. The C constants are prefixed by DS_O_.

Table 6.2, "Selected Object Classes" shows the selected object classes, as defined in the standards.

Table 6.2. Selected Object Classes

Object Class	Package	Object Class	Package
O-Alia s	BDCP	O-MHS-Message-Trans-Ag	MDUP
O-Applic-Entity	BDCP	O-MHS-User	MDUP
O-Applic-Process	BDCP	O-MHS-User-Agent	MDUP

Object Class	Package	Object Class	Package
O-Cert-Authority	SAP	O-Org	BDCP
O-Country	BDCP	O-Org-Person	BDCP
O-Device	BDCP	O-Org-Role	BDCP
O-DSA	BDCP	O-Org-Unit	BDCP
O-Group-Of-Names	BDCP	O-Person	BDCP
O-Locality	BDCP	O-Residential-Person	BDCP
O-MHS-Distribution-List	MDUP	O-Strong-Authent-User	SAP
O-MHS-Message-Store	MDUP	O-Top	BDCP

Each object class has zero or more mandatory attributes and zero or more optional attributes. A directory entry must contain all the mandatory attributes associated with its object class and superclasses.

6.3. OM Class Hierarchy

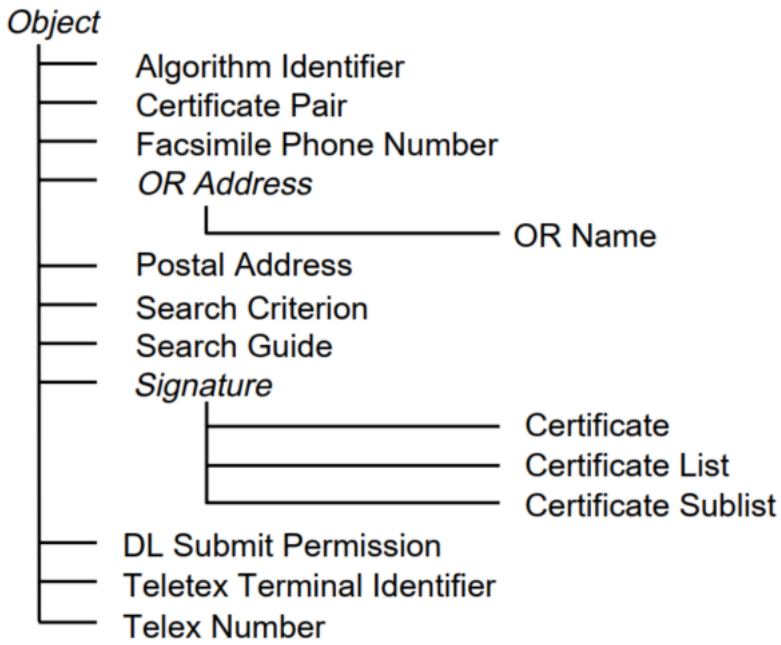
Figure 6.1, "Hierarchy of Selected Object Classes" shows the hierarchical organization of the selected object classes. It shows which OM classes inherit attributes from their superclasses. The name of the abstract OM classes is shown in italics.

Note

VSI's X.500 API does not fully support the object classes in *Figure 6.1, "Hierarchy of Selected Object Classes"*. You cannot pass them as interface objects, only as ASN.1 structures.

The object class Object is defined in the *OSI-Abstract-Data Manipulation*. It contains one attribute, Class, which denotes the class of the object.

Figure 6.1. Hierarchy of Selected Object Classes



The following sections describe the additional OM classes used to represent values of the selected attributes described in Section 6.1. These are described in alphabetical order. Each OM class is described in a separate section that identifies the OM attributes specific to that class and lists its superclasses. The attributes in an instance of an OM class include those specific to that class and those inherited from each of its superclasses. For each class, there is a table that defines the class-specific attributes, if there are any. The tables give the following information:

- The name of each attribute
- The syntax of each attribute value (including any appropriate object class, enumeration syntax or string type in parenthesis)
- The restrictions, if any, on the length of each attribute value
- The number of values each attribute can have
- The initial value, if any, that the OM-Create function supplies (if initialization is requested)

These classes cannot be encoded using the OM-Encode and OM-Decode functions.

6.3.1. Algorithm Identifier

DS_C_ALGORITHM_IDENT

An instance of the OM class Algorithm Identifier records the encryption algorithm used by an object to sign messages. It also contains the algorithm parameters. It has the attributes of its superclass, Object, and the OM attributes listed in the following table:

OM Attribute Name	Value Syntax	Number of Values
Algorithm	String(Object Identifier)	1
Algorithm Parameters	Any	0-1

Algorithm (DS_ALGORITHM)

An object identifier that uniquely identifies an algorithm used by an object.

Algorithm P arameters (DS_ALGORITHM_PARAMETERS)

The values of the algorithm parameters that the object uses. The syntax is determined by the algorithm.

6.3.2. Certificate

DS_C_CERT

An instance of the OM class Certificate is a Certificate containing a user's Distinguished Name, public key, and additional information. These are all signed by the issuing Certification Authority so that the certification cannot be forged. It has the attributes of its superclasses, Object and Signature, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Value Length	Number of Values	Initial Value
Serial Number	Integer	–	1	–

OM Attribute Name	Value Syntax	Value Length	Number of Values	Initial Value
Subject	Object(Name)	–	1	–
Subject Algorithm	Object(Algorithm Identifier)	–	1	–
Subject Public Key	String(Bit)	–	1	–
Validity Not After	String(UTC-Time)	0-17	1	–
Validity Not Before	String(UTC-Time)	0-17	1	–
Version	Enum(Ver sion)	–	1	*v1988*

Serial Number (DS_SERIAL_NBR)

Distinguishes this certificate from all the others that were or will be issued by the Certification Authority that issued it.

Subject (DS_SUBJ ECT) The subject's name.

Subject Algorithm (DS_SUBJ ECT_ALGORITHM)

The algorithm that is used by the subject for encryption. This is associated with the public key.

Subject Public Key (DS_SUBJ ECT_PUBLIC_KEY)

The public key of the subject. This is associated with the encryption algorithm.

Validity Not After (DS_VALIDITY_NOT_AFTER)

Indicates the last day that the certificate is valid. Note, the upper limit of the value length (17) is contained in the integer constant DS_VL_ VALIDITY_NOT_AFTER.

Validity Not Before (DS_VALIDITY_NOT_BEFORE)

Indicates the first day that the certificate is valid. Note, the upper limit of the value length (17) is contained in the integer constant DS_VL_ VALIDITY_NOT_BEFORE.

Version (DS_VERSION)

Identifies the design of the certificate. Its value is *v1988* meaning that the design is that specified in the 1988 version of the standards.

6.3.3. Certificate List

DS_C_CERT_LIST

An instance of the OM class Certificate List documents the revocation of zero or more certificates. The documentation is provided by a Certification Authority whose signature is associated with the instance. It has the attributes of its superclasses, Object and Signature, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Value Length	Number of Values
Last Update	String(UTC-Time)	0-17	1

OM Attribute Name	Value Syntax	Value Length	Number of Values
Revoked Certificates	Object(Certificate Sublist)	–	0 or more

Last Update (DS_LAST_UPDATE)

Indicates the time at which the revocation list was updated to its current state. Note, the upper limit of the value length (17) is contained in the integer constant DS_VL_LAST_UPDATE.

Revoked Certificates (DS_REVOKED_CERTS)

Identifies the revoked certificates.

6.3.4. Certificate Pair

DS_C_CERT_PAIR

An instance of the OM class Certificate Pair contains one or both copies of a forwar d and reverse certificate to help users build a certification path. It has the attributes of its superclass, Object, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Number of Values
Forward	Object(Certificate)	0-1
Reverse	Object(Certificate)	0-1

Forw ard (DS_FORWARD)

The certificate of one Certification Authority issued by a second Certification Authority.

Reverse (DS_REVERSE)

The certificate of the second Certification Authority issued by the first Certification Authority.

At least one of the above attributes must be present.

6.3.5. Certificate Sublist

DS_C_CERT_SUBLIST

An instance of the OM class Certificate Sublist documents the revocation of zero or more certificates issued by the Certification Authority whose signature is associated with the instance. It has the attributes of its superclasses, Object and Signature, and the OM attributes listed in the following table.

Value

OM Attribute Name	Value Syntax	Value Length	Number of Values
Revocation Date	String(UTC-Time)	0-17	0 or more[1]

OM Attribute Name	Value Syntax	Value Length	Number of Values
Serial Numbers	Integer	–	0 or more[1]

[1]The values of these two attributes parallel one another and must be equal in number

Revocation Date (DS_REVOC_DATE)

Indicates the date when each of the certificates was revoked. The serial numbers of the certificates are the corresponding values of the OM attribute Serial Numbers. Note, the upper limit of the value length (17) is contained in the integer constant DS_VL_REVOC_DATE.

Serial Numbers (DS_SERIAL_NBRS)

The serial numbers assigned to the revoked certificates. The values of the above attributes must be equal in number.

6.3.6. Facsimile Phone Number

DS_C_FACSIMILE_PHONE_NBR

An instance of the OM class Facsimile Phone Number identifies and optionally describes a facsimile terminal. It has the attributes of its superclass, Object, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Value Length	Number of Values
Parameters	Object(G3 Fax NBs)	–	0-1
Phone Number	String(Printable)[1]	1-32	1

[1]As permitted by International Standard E.123

Parameters (DS_PARAMETERS)

If present, identifies the non-basic capabilities of the facsimile terminal. Its value syntax is as defined in the guide *MAILbus 400 Application Program Interface Programming*.

Phone Number (DS_PHONE_NBR)

A telephone number through which the facsimile terminal can be accessed. Note, the upper limit of the value length (32) is contained in the integer constant DS_VL_PHONE_NBR.

6.3.7. DL Submit Permission

DS_C_DL_SUBMIT_PERMS

An instance of the OM class DL Submit Permission characterizes an attribute each of whose value is a submit permission. It has the attributes of its superclass, Object, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Number of Values
Permission Type	Enum(Permission-Type)	1
Individual	Object(OR Name)	0-1

OM Attribute Name	Value Syntax	Number of Values
Member of DL	Object(OR Name)	0-1
Pattern Match	Object(OR Name)	0-1
Member of Group	Object(Name)	0 or more

Permission Type (DS_PERM_TYPE)

Indicates the type of the permission specified by DL-Submit-Permission. Its value is one of:

- *individual* (DS_PERM_INDIVIDUAL)
- *member of dl* (DS_PERM_MEMBER_OF_DL)
- *pattern match* (DS_PERM_PATTERN_MATCH)
- *member of group* (DS_PERM_MEMBER_OF_GROUP)

It also indicates which of the following four attributes is also present.

Individual (DS_INDIVIDUAL)

Indicates the user or unexpanded distribution list whose O/R name is equal to the specified O/R name.

Member of DL (DS_MEMBER_OF_DL)

Indicates each member of the distribution list or nested distribution list whose O/R name is equal to the specified O/R name.

P attern Match (DS_PATTERN_MATCH)

Indicates each user or unexpanded distribution list whose O/R name matches the specified O/R name pattern.

Member of Group (DS_MEMBER_OF_GROUP)

Indicates each member of the group-of-names whose name is specified, or of each nested group-of-names recursively.

6.3.8. Postal Address

DS_C_POSTAL_ADDRESS

An instance of the OM class Postal Address is a postal address. It has the attributes of its superclass, Object, and the following OM attribute:

OM Attribute Name	Value Syntax	Value Length	Number of Values
Postal Address	String(Teletex)	1-30	1-6

P ostal Address (DS_POSTAL_ADDRESS)

Signifies the postal address. Each value of this attribute is one line of address. Note, the upper limit of the value length (30) is contained in the integer constant DS_VL_POSTAL_ADDRESS. Note, the upper limit of the number of values (6) is contained in the integer constant DS_VN_POSTAL_ ADDRESS.

6.3.9. Search Criterion

DS_C_SEARCH_CRITERION

An instance of the OM class Search Criterion is a component of a Search Guide OM object. It has the attributes of its superclass, Object, and the OM attributes listed in the following table:

OM Attribute Name	Value Syntax	Number of Values	Initial Value
Attribute Type	String(Object Identifier)	0-1	–
Criteria	Object(Search Criterion)	0 or more	–
Filter Item Type	Enum(Filter-Item-Type)	0-1	–
Filter Type	Enum(Filter-Type)	1	*item*

Attribute Type (DS_ATTRIBUTE_TYPE)

The attribute type to be used in the suggested Filter-Item. This attribute is only present if the value of Filter-Type is *item*.

Criteria (DS_CRITERIA)

The nested search criteria. This attribute is not present if the value of Filter-Type is *item*.

Filter Item Type (DS_FILTER_ITEM_TYPE)

The type of the suggested filter-item. This OM attribute is only present when the value of Filter-Item is *item*. Its value is one of the following:

approximate-match (DS_APPROXIMATE_MATCH)
equality (DS_EQUALITY)
greater-or-equal (DS_GREATER_OR_EQUAL)
less-or-equal (DS_LESS_OR_EQUAL)
substrings (DS_SUBSTRINGS)

It cannot have the value *present* (DS_PRESENT). Refer to Section 3.17.

Filter Type (DS_FILTER_TYPE)

The type of the suggested filter. Refer to *Section 3.16, "Filter"* for a description of this attribute.

6.3.10. Search Guide

DS_C_SEARCH_GUIDE

An instance of the OM class Search Guide suggests a criterion for searching the Directory for particular entries. It can be used to build a Filter for a Search operation that is based on the object in whose entry the guide is placed. It has the attributes of its superclass, Object, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Number of Values
Object Class	String(Object Identifier)	0-1

OM Attribute Name	Value Syntax	Number of Values
Criteria	Object(Search Criterion)	1

Object Class (DS_OBJ ECT_CLASS)

Identifies the object class of the entries to which the search guide applies. The search guide applies to objects of any class if this attribute is absent.

Criteria (DS_CRITERIA)

The suggested search criteria.

6.3.11. Signature

DS_C_SIGNATURE

The abstract OM class Signature contains the algorithm identifier used to produce a digital signature and the name of the object that produced it. The scope of the signature is any instance of any subclass of Signature. It has the attributes of its superclass, Object, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Number of Values
Issuer	Object(Name)	1
Signature	Object(Algorithm Identifier)	1
Signature Value	String(Octet)	1

Issuer (DS_ISSUER)

The name of the object that produced the digital signature.

Signature (DS_SIGNATURE)

Identifies the algorithm that was used to produce the digital signature and any parameters of the algorithm.

Signature Value (DS_SIGNATURE_VALUE)

An enciphered summary of the information connected with the signature. The summary is produced by means of a one-way hash function. Enciphering is carried out using the secret key of the signer.

6.3.12. Teletex Terminal Identifier

DS_C_TELETEX_TERM_ID ENT

An instance of OM class Teletex Terminal Identifier identifies and describes a Teletex terminal. It has the attributes of its superclass, Object, and the OM attributes listed in the following table.

OM Attribute Name	Value Syntax	Value Length	Number of Values
Parameters	Object(Teletex NBPs)	–	0-1
Teletex Terminal	String(Printable)[1]	1-1024	1

[1] As permitted by the International Standard F.200

Parameters (DS_PARAMETERS)

Identifies the non basic capabilities of the Teletex terminal.

Teletex Terminal (DS_TELETEX_TERM)

Identifies a Teletex terminal. Note, the upper limit of the value length (1024) is contained in the integer constant DS_VL_TELETEX_TERM.

6.3.13. Telex Number

DS_C_TELEX_NBR

An instance of OM class Telex Number identifies and describes a telex terminal. It has the attributes of its superclass, Object, and the OM attributes listed in the following table.

OM Attribute Name	Value Length	Value Syntax	Number of Values
Answerback	String(Printable)	1-8	1
Country Code	String(Printable)	1-4	1
Telex Number	String(Printable)	1-14	1

Answerback (DS_ANSWERBACK)

The code with which the telex terminal acknowledges calls to it. Note, the upper limit of the value length (8) is contained in the integer constant DS_VL_ANSWERBACK.

Country Code (DS_COUNTRY_CODE)

The identifier of the country through which the telex terminal is accessed. Note, the upper limit of the value length (4) is contained in the integer constant DS_VL_COUNTRY_CODE.

Telex Number (DS_TELEX_NBR)

The number by which the telex terminal is addressed. The upper limit of the value length (14) is contained in the integer constant DS_VL_TELEX_NBR.