

Contents

Active Directory Domain Services

 About Active Directory Domain Services

 Core Concepts of Active Directory Domain Services

 Attributes

 Containers and Leaves

 Object Names and Identities

 Naming Contexts and Directory Partitions

 About Application Directory Partitions

 Domain Trees

 Forests

 Active Directory Servers and Dynamic DNS

 Replication and Data Integrity

 Replication Model in Active Directory Domain Services

 Features of the Replication Model for Active Directory Domain Services

 Why Active Directory Domain Services Uses This Replication Model

 A Programmer's Model of Replication in Active Directory Domain Services

 Replication Behavior in Active Directory Domain Services

 Impact on Directory-Enabled Applications

 Detecting and Avoiding Replication Latency

 What Can You Know, and When Can You Know It?

 Temporal Locality

 Out-of-Band Signaling

 Effective Date and Time

 Checksums and Object Counts

 Consistency GUIDs

 Versioning and Fallback Strategies

 Active Directory Domain Services Architecture

 Directory System Agent

 Data Model

[Schema](#)

[Administration Model](#)

[Global Catalog](#)

[Security in Active Directory Domain Services](#)

[Object and Attribute Protection](#)

[Delegation](#)

[Inheritance](#)

[Active Directory Schema](#)

[About the Active Directory Schema](#)

[Read-Only DCs and the Active Directory Schema](#)

[Schema Implementation](#)

[Object Identifiers](#)

[Characteristics of Object Classes](#)

[Structural, Abstract, and Auxiliary Classes](#)

[Class Inheritance in the Active Directory Schema](#)

[Object Class and Object Category](#)

[Characteristics of Attributes](#)

[Syntaxes for Attributes in Active Directory Domain Services](#)

[Indexed Attributes](#)

[Including Attributes in the Global Catalog](#)

[Linked Attributes](#)

[The Abstract Schema](#)

[Using Active Directory Domain Services](#)

[Binding to Active Directory Domain Services](#)

[Serverless Binding and RootDSE](#)

[Example Code for Getting the Distinguished Name of the Domain](#)

[Binding to the Global Catalog](#)

[Example Code for Searching a Forest](#)

[Using objectGUID to Bind to an Object](#)

[Example Code for Creating a Bindable String Representation of a GUID](#)

[Reading an Object's objectGUID and Creating a String Representation of the GUID](#)

[Binding to Well-Known Objects Using WKGUID](#)

[Example Code for Binding to Well Known Objects](#)

[Binding to an Object Using a SID](#)

[Enabling Rename-Safe Binding with the otherWellKnownObjects Property](#)

[Example Code for Creating a Container Object](#)

[Authentication](#)

[Searching in Active Directory Domain Services](#)

[Deciding What to Find](#)

[Example Code for Searching for Users](#)

[Deciding Where to Search](#)

[Searching Domain Contents](#)

[Searching the Global Catalog](#)

[Choosing the Search Technology](#)

[Creating a Query Filter](#)

[Finding Objects by Class](#)

[Finding Objects by Name](#)

[Finding a List of Attributes To Query](#)

[Checking the Query Filter Syntax](#)

[How to Specify Comparison Values](#)

[Deciding Which Attributes to Retrieve for Each Object Found](#)

[Retrieving the objectClass Attribute](#)

[Binding to the Search Start Point](#)

[Specifying the Search Scope](#)

[Specifying Other Search Options](#)

[Processing the Search Results](#)

[Effects of Security on Searching](#)

[Creating Efficient Queries](#)

[How Tuple Indexing Works](#)

[Initial, Medial, and Final Search Strings](#)

[Referrals](#)

[When Referrals Are Generated](#)

[Creating an External Referral](#)

[Example Code for Creating an External crossRef Object](#)

[Enumerating Domain Controllers](#)

[Creating and Deleting Objects in Active Directory Domain Services](#)

[Retrieving Deleted Objects](#)

[Restoring Deleted Objects](#)

[Moving Objects](#)

[Example Code for Moving an Object](#)

[Reading and Writing Attributes of Objects in Active Directory Domain Services](#)

[Determining an Attribute Type](#)

[Controlling Access to Objects in Active Directory Domain Services](#)

[How Access Control Works in Active Directory Domain Services](#)

[Controlling Access to Objects and Their Properties](#)

[Access Rights for Active Directory Domain Services Objects](#)

[Security Contexts and Active Directory Domain Services](#)

[How Security Affects Operations in Active Directory Domain Services](#)

[Access Control and Read Operations](#)

[Access Control and Write Operations](#)

[Access Control and Object Creation](#)

[Access Control and Object Deletion](#)

[APIs for Working with Security Descriptors](#)

[Using IADs to Get a Security Descriptor](#)

[Using IDirectoryObject to Get a Security Descriptor](#)

[Security Descriptor Components](#)

[Retrieving an Object's DACL](#)

[Retrieving an Object's SACL](#)

[Reading an Object's Security Descriptor](#)

[Example Code for Creating a Security Descriptor](#)

[Example Code for Enumerating the ACL of an Object in Active Directory Domain Services](#)

[Setting Access Rights on an Object](#)

[Example Code for Setting an ACE on a Directory Object](#)

[Setting Access Rights on the Entire Object](#)

[Example Code for Setting Read Property Rights on an Object](#)

[Setting Permissions to a Specific Property](#)

Setting Permissions on a Group of Properties

Example Code for Setting Permissions on a Group of Properties

Setting Permissions on Child Object Operations

Example Code for Setting Permissions on Child Object Operations

How Security Descriptors are Set on New Directory Objects

Creating a Security Descriptor for a New Directory Object

Inheritance and Delegation of Administration

Access Control Inheritance

Setting Rights to Specific Types of Objects

Example Code for Setting Rights to Specific Types of Objects

Setting Rights to Specific Properties of Specific Types of Objects

Protecting Objects from the Effects of Inherited Rights

Example Code for Setting and Removing SACL and DACL Protection

Default Security Descriptor

Reading the defaultSecurityDescriptor for an Object Class

Example Code for Reading defaultSecurityDescriptor

Modifying the defaultSecurityDescriptor for an Object Class

Control Access Rights

Creating a Control Access Right

Example Code for Creating a Control Access Right

Setting a Control Access Right ACE in an Object's ACL

Example Code for Setting a Control Access Right ACE

Checking a Control Access Right in an Object's ACL

Example Code for Checking a Control Access Right in an Object's ACL

Reading a Control Access Right Set in an Object's ACL

Example Code for Checking for a Control Access Right in an ACE

Using DsAddSidHistory

Controlling Object Visibility

Null DACLs and Empty DACLs

Extending the Schema

Guidelines for Binding to the Schema

Reading the Abstract Schema

[Example Code for Enumerating Schema Classes, Attributes, and Syntaxes](#)

[Reading attributeSchema and classSchema Objects](#)

[Example Code for Searching for Schema Objects](#)

[What You Must Know Before Extending the Schema](#)

[Impact of Schema Changes](#)

[When to Extend the Schema](#)

[Restrictions on Schema Extension](#)

[Querying for Category 1 or 2 Schema Objects](#)

[How to Extend the Schema](#)

[Naming Attributes and Classes](#)

[Disabling Existing Classes and Attributes](#)

[Obtaining a Link ID](#)

[Obtaining an Object Identifier](#)

[Obtaining a Root OID from an ISO Name Registration Authority](#)

[Obtaining an Object Identifier from Microsoft](#)

[Integrating Schema Extensions with the User Interface](#)

[Defining a New Attribute](#)

[Example Code for Creating an Attribute](#)

[Choosing a Syntax](#)

[Defining a New Class](#)

[Example Code for Creating a Class](#)

[Installing Schema Extensions](#)

[Documenting Schema Extensions](#)

[What the Installation Must Do](#)

[Prerequisites for Installing a Schema Extension](#)

[Recommendations for Schema Extension Applications](#)

[Supported Installation Mechanisms](#)

[Extending the User Interface for Directory Objects](#)

[About the User Interfaces of Active Directory Domain Services](#)

[Display Specifiers](#)

[DisplaySpecifiers Container](#)

[Class and Attribute Display Names](#)

[Class Icons](#)

[Viewing Containers as Leaf Nodes](#)

[Modifying Existing User Interfaces](#)

[User Interface Extension for New Object Classes](#)

[Property Pages for Use with Display Specifiers](#)

[Implementing the Property Page COM Object](#)

[Example Code for Implementation of the Property Sheet COM Object](#)

[Registering the Property Page COM Object in a Display Specifier](#)

[Context Menus for Use with Display Specifiers](#)

[Implementing the Context Menu COM Object](#)

[Example Code for Implementation of the Context Menu COM Object](#)

[Registering the Context Menu COM Object in a Display Specifier](#)

[Registering a Static Context Menu Item](#)

[Example Code for Installing a Static Context Menu Item](#)

[Object Creation Wizards](#)

[Implementing the Object Creation Extension COM Object](#)

[Registering the Object Creation Extension](#)

[Invoking Creation Wizards from Your Application](#)

[Using Standard Dialog Boxes to Handle Objects in Active Directory Domain Services](#)

[Directory Object Picker](#)

[Domain Browser](#)

[Container Browser](#)

[Active Directory Users and Computers Property Sheets](#)

[Administrative Notification Handlers](#)

[Distributing User Interface Components](#)

[How Applications Should Use Display Specifiers](#)

[Debugging an Active Directory Extension](#)

[Managing Users](#)

[Users in Active Directory Domain Services](#)

[Security Principals](#)

[What is a User?](#)

[Example Code for Binding to the User's Container](#)

User Object Attributes

User Naming Attributes

User Security Attributes

User Address Book Attributes

Creating a User

Example Code for Creating a User

Enumerating Users

Querying for Users

Example Code for Using the Global Catalog to Find Users in a Forest

Managing Users on Member Servers and Windows 2000 Professional

Enumerating Users on Member Servers and Windows 2000 Professional

Example Code for Enumerating Users on a Member Server

Creating Users on Member Servers and Windows 2000 Professional

Example Code for Creating Users on a Member Server or Windows 2000 Professional

Deleting Users on Member Servers and Windows 2000 Professional

Example Code for Deleting Users on a Member Server or Windows 2000 Professional

Values for the countryCode and c Properties

Managing Groups

Group Objects

How Security Groups are Used in Access Control

Creating Groups in a Domain

Example Code for Creating a Group

Adding Members to Groups in a Domain

Example Code for Adding a Member to a Group

Example Code for Converting an objectSid into a Bindable String

Removing Members from Groups in a Domain

Example Code for Removing a Member from a Group

Nesting a Group in Another Group

Nesting in Native Mode

Nesting in Mixed Mode

Common Errors

[Determining a User's or Group's Membership in a Group](#)

[Example Code for Checking for Membership in a Group](#)

[Enumerating Groups](#)

[Enumerating Groups in a Domain](#)

[Searching for Groups by Scope or Type in a Domain](#)

[Enumerating Members in a Group](#)

[Example Code for Displaying Members of a Group](#)

[Enumerating Groups That Contain Many Members](#)

[Querying for Groups in a Domain](#)

[Example Code for Searching for Groups in a Domain](#)

[Changing a Group's Scope or Type](#)

[Example Code for Changing the Scope of a Group](#)

[Getting the Domain Account-Style Name of a Group](#)

[Groups on Member Servers and Windows 2000 Professional](#)

[Enumerating Local Groups](#)

[Example Code for Enumerating Local Groups](#)

[Creating Local Groups](#)

[Example Code for Creating a Local Group](#)

[Deleting Local Groups](#)

[Example Code for Deleting a Local Group](#)

[Adding Domain Objects to Local Groups](#)

[Example Code for Adding a Domain User or Group to a Local Group](#)

[What Application and Service Developers Need to Know About Groups](#)

[Tracking Changes](#)

[Overview of Change Tracking Techniques](#)

[Change Notifications in Active Directory Domain Services](#)

[Example Code for Receiving Change Notifications](#)

[Polling for Changes Using the DirSync Control](#)

[Example Code Using ADS_SEARCHPREF_DIRSYNC](#)

[Polling for Changes Using USNChanged](#)

[Example Code to Retrieve Changes Using USNChanged](#)

[Service Publication](#)

[About Service Publication](#)

[Security Issues for Service Publication](#)

[Connection Points](#)

[Publishing with Service Connection Points](#)

[Where to Create a Service Connection Point](#)

[Publishing Under a Computer Object](#)

[Publishing in a Domain System Container](#)

[Service Connection Points for Replicated, Host-based, and Database Services](#)

[Service Connection Point Properties](#)

[Creating and Maintaining a Service Connection Point](#)

[Script Samples for Managing Service Connection Points](#)

[Code Samples for Managing Service Connection Points](#)

[Publishing with the RPC Name Service \(RpcNs\)](#)

[Publishing with Windows Sockets Registration and Resolution](#)

[Example Code for Installing an RnR Service Class](#)

[Example Code for Implementing a Winsock Service with an RnR Publication](#)

[Example Code for Publishing the RnR Connection Point](#)

[Example Code for Removing the RnR Connection Point](#)

[Example Code for Locating a Service Using an RnR Query](#)

[Publishing COM+ Services](#)

[Service Logon Accounts](#)

[About Service Logon Accounts](#)

[Guidelines for Selecting a Service Logon Account](#)

[Using a Local User Account as a Service Logon Account](#)

[Using a Domain User Account as a Service Logon Account](#)

[Using the LocalSystem Account as a Service Logon Account](#)

[Setting up a Service's User Account](#)

[Installing a Service on a Host Computer](#)

[Granting Logon as Service Right on the Host Computer](#)

[Testing Whether Running on a Domain Controller](#)

[Granting Access Rights to the Service Logon Account](#)

[Enabling Service Account to Access SCP Properties](#)

[Logon Account Maintenance Tasks](#)

[Changing the Password on a Service's User Account](#)

[Changing the Password on a Service's User Account](#)

[Enumerating the Replicas of a Service](#)

[Converting Domain Account Name Formats](#)

[Mutual Authentication Using Kerberos](#)

[About Mutual Authentication Using Kerberos](#)

[Security Providers](#)

[Integrity and Privacy](#)

[Limitations of Mutual Authentication with Kerberos](#)

[Service Principal Names](#)

[Name Formats for Unique SPNs](#)

[How a Service Composes its SPNs](#)

[How a Service Registers its SPNs](#)

[How Clients Compose a Service's SPN](#)

[Mutual Authentication in a Windows Sockets Service with an SCP](#)

[How a Client Authenticates an SCP-based Windows Sockets Service](#)

[Composing and Registering SPNs for an SCP-based Windows Sockets Service](#)

[Composing the SPNs for a Service with an SCP](#)

[Registering the SPNs for a Service](#)

[How a Windows Sockets Service Authenticates a Client](#)

[Mutual Authentication in RPC Applications](#)

[Composing SPNs for an RpcNs Service](#)

[Mutual Authentication in Windows Sockets Applications](#)

[Backing Up and Restoring an Active Directory Server](#)

[Considerations for Active Directory Domain Services Backup](#)

[Backing Up an Active Directory Server](#)

[Restoring an Active Directory Server](#)

[Storing Dynamic Data](#)

[Dynamic Objects](#)

[Creating Dynamic Objects](#)

[Refreshing a Dynamic Object](#)

[Configuration of TTL Limits](#)

[Dynamic Auxiliary Classes](#)

[Dynamically Linked Auxiliary Classes](#)

[Statically Linked Auxiliary Classes](#)

[Determining the Classes Associated With an Object Instance](#)

[Adding an Auxiliary Class to an Object Instance](#)

[Application Directory Partitions](#)

[Application Directory Partition Replication](#)

[Application Directory Partition Security](#)

[Creating an Application Directory Partition](#)

[Example Code for Creating an Application Directory Partition](#)

[Example Code for Locating the Partitions Container](#)

[Deleting an Application Directory Partition](#)

[Example Code for Deleting an Application Directory Partition](#)

[Enumerating Application Directory Partitions in a Forest](#)

[Locating an Application Directory Partition Host Server](#)

[Example Code for Locating an Application Directory Partition Host Server](#)

[Adding or Deleting an Application Directory Partition Replica](#)

[Enumerating Replicas of an Application Directory Partition](#)

[Modifying Application Directory Partition Configuration](#)

[Detecting the Operation Mode of a Domain](#)

[Example Code for Determining the Operation Mode](#)

[Active Directory Domain Services Reference](#)

[Constants in Active Directory Domain Services](#)

[GUIDs of User Interface Elements](#)

[Messages Communicated through User Interfaces](#)

[CQPM_CLEARFORM](#)

[CQPM_ENABLE](#)

[CQPM_GETPARAMETERS](#)

[CQPM_HANDLERSPECIFIC](#)

[CQPM_HELP](#)

[CQPM_INITIALIZE](#)

[CQPM_PERSIST](#)

[CQPM_RELEASE](#)

[CQPM_SETDEFAULTPARAMETERS](#)

[CFSTR_DS_DISPLAY_SPEC_OPTIONS](#)
[CFSTR_DS_MULTISELECTPROPPAGE](#)
[CFSTR_DS_PARENTHWND](#)
[CFSTR_DS_PROPSHEETCONFIG](#)
[CFSTR_DSOBJECTNAMES](#)
[CFSTR_DSOP_DS_SELECTION_LIST](#)
[CFSTR_DSPROPERTYPAGEINFO](#)
[CFSTR_DSQUERYPARAMS](#)
[CFSTR_DSQUERYSCOPE](#)
BFT Constants

Structures in Active Directory Domain Services

Admin Structures in Active Directory Domain Services

[DSA_NEWOBJ_DISPINFO](#)

Display Structures in Active Directory Domain Services

[CQFORM](#)

[CQPAGE](#)

[DSA_SEC_PAGE_INFO](#)

[DOMAINDESC](#)

[DOMAINTREE](#)

[DSBITEM](#)

[DSBROWSEINFO](#)

[DSCLASSCREATIONINFO](#)

[DSCOLUMN](#)

[DSDISPLAYSPECOPTIONS](#)

[DSOBJECT](#)

[DSOBJECTNAMES](#)

[DSPROPERTYPAGEINFO](#)

[DSQUERYCLASSTLIST](#)

[DSQUERYINITPARAMS](#)

[DSQUERYPARAMS](#)

[OPENQUERYWINDOW](#)

[PROPSHEETCFG](#)

MMC Property Page Structures in Active Directory Domain Services

ADSPROPERROR

ADSPROPINITPARAMS

Domain Controller and Replication Management Structures

DS_DOMAIN_CONTROLLER_INFO_1

DS_DOMAIN_CONTROLLER_INFO_2

DS_DOMAIN_CONTROLLER_INFO_3

DS_NAME_RESULT

DS_NAME_RESULT_ITEM

DS_REPL_ATTR_META_DATA

DS_REPL_ATTR_META_DATA_2

DS_REPL_ATTR_META_DATA_BLOB

DS_REPL_ATTR_VALUE_META_DATA

DS_REPL_ATTR_VALUE_META_DATA_2

DS_REPL_ATTR_VALUE_META_DATA_EXT

DS_REPL_CURSOR

DS_REPL_CURSOR_2

DS_REPL_CURSOR_3

DS_REPL_CURSOR_BLOB

DS_REPL_CURSORS

DS_REPL_CURSORS_2

DS_REPL_CURSORS_3

DS_REPL_KCC_DSA_FAILURE

DS_REPL_KCC_DSA_FAILURES

DS_REPL_KCC_DSA_FAILUREW_BLOB

DS_REPL_NEIGHBOR

DS_REPL_NEIGHBORS

DS_REPL_NEIGHBORW_BLOB

DS_REPL_OBJ_META_DATA

DS_REPL_OBJ_META_DATA_2

DS_REPL_OP

DS_REPL_OPW_BLOB

DS_REPL_PENDING_OPS
DS_REPL_QUEUE_STATISTICSW
DS_REPL_VALUE_META_DATA
DS_REPL_VALUE_META_DATA_2
DS_REPL_VALUE_META_DATA_BLOB
DS_REPL_VALUE_META_DATA_BLOB_EXT
DS_REPL_VALUE_META_DATA_EXT
DS_REPSYNCALL_ERRINFO
DS_REPSYNCALL_SYNC
DS_REPSYNCALL_UPDATE
DS_SCHEMA_GUID_MAP
DS_SITE_COST_INFO
SCHEDULE
SCHEDULE_HEADER

Directory Service Structures

DOMAIN_CONTROLLER_INFO
DS_DOMAIN_TRUSTS
DSROLE_OPERATION_STATE_INFO
DSROLE_PRIMARY_DOMAIN_INFO_BASIC
DSROLE_UPGRADE_STATUS_INFO

Directory Backup Structures

EDB_RSTMAP

Object Picker Dialog Box Structures

DS_SELECTION
DS_SELECTION_LIST
DSOP_FILTER_FLAGS
DSOP_INIT_INFO
DSOP_SCOPE_INIT_INFO
DSOP_UPLEVEL_FILTER_FLAGS

Enumerations in Active Directory Domain Services

DS_KCC_TASKID
DS_MANGLE_FOR

DS_NAME_ERROR
DS_NAME_FLAGS
DS_NAME_FORMAT
DS_REPL_INFO_TYPE
DS_REPL_OP_TYPE
DS_REPSYNCALL_ERROR
DS_REPSYNCALL_EVENT
DS_SPN_NAME_TYPE
DS_SPN_WRITE_OP
DSROLE_MACHINE_ROLE
DSROLE_OPERATION_STATE
DSROLE_PRIMARY_DOMAIN_INFO_LEVEL
DSROLE_SERVER_STATE

Functions in Active Directory Domain Services

Display Functions in Active Directory Domain Services

BFFCallBack
CQAddFormsProc
CQAddPagesProc
CQPageProc
DsBrowseForContainer
DSEnumAttributesCallback
DsGetFriendlyClassName
DsGetIcon

MMC Property Page Functions in Active Directory Domain Services

ADsPropCheckIfWritable
ADsPropCreateNotifyObj
ADsPropGetInitInfo
ADsPropSendErrorMessage
ADsPropSetHwnd
ADsPropSetHwndWithTitle
ADsPropShowErrorDialog

Domain Controller and Replication Management Functions

DsAddSidHistory
DsBind
DsBindingSetTimeout
DsBindToISTG
DsBindWithCred
DsBindWithSpn
DsBindWithSpnEx
DsClientMakeSpnForTargetServer
DsCrackNames
DsCrackSpn
DsCrackUnquotedMangledRdn
DsFreeDomainControllerInfo
DsFreeNameResult
DsFreePasswordCredentials
DsFreeSchemaGuidMap
DsFreeSpnArray
DsGetDomainControllerInfo
DsGetRdnW
DsGetSpn
DsInheritSecurityIdentity
DsIsMangledDn
DsIsMangledRdnValue
DsListDomainsInSite
DsListInfoForServer
DsListRoles
DsListServersForDomainInSite
DsListServersInSite
DsListSites
DsMakePasswordCredentials
DsMakeSpn
DsMapSchemaGuids
DsQuerySitesByCost

DsQuerySitesFree
DsQuoteRdnValue
DsRemoveDsDomain
DsRemoveDsServer
DsReplicaAdd
DsReplicaConsistencyCheck
DsReplicaDel
DsReplicaFreeInfo
DsReplicaGetInfo
DsReplicaGetInfo2
DsReplicaModify
DsReplicaSync
DsReplicaSyncAll
DsReplicaUpdateRefs
DsReplicaVerifyObjects
DsServerRegisterSpn
DsUnBind
DsUnquoteRdnValue
DsWriteAccountSpn
SyncUpdateProc

Directory Service Functions

DsAddressToSiteNames
DsAddressToSiteNamesEx
DsDeregisterDnsHostRecords
DsEnumerateDomainTrusts
DsGetDcClose
DsGetDcName
DsGetDcNext
DsGetDcOpen
DsGetDcSiteCoverage
DsGetForestTrustInformationW
DsGetSiteName

[DsMergeForestTrustInformationW](#)

[DsRoleFreeMemory](#)

[DsRoleGetPrimaryDomainInformation](#)

[DsValidateSubnetName](#)

[Directory Backup Functions](#)

[DsBackupClose](#)

[DsBackupEnd](#)

[DsBackupFree](#)

[DsBackupGetBackupLogs](#)

[DsBackupGetDatabaseNames](#)

[DsBackupOpenFile](#)

[DsBackupPrepare](#)

[DsBackupRead](#)

[DsBackupTruncateLogs](#)

[DsIsNTDSOnline](#)

[DsRestoreEnd](#)

[DsRestoreGetDatabaseLocations](#)

[DsRestorePrepare](#)

[DsRestoreRegister](#)

[DsRestoreRegisterComplete](#)

[DsSetAuthIdentity](#)

[DsSetCurrentBackupLog](#)

[Interfaces for Active Directory Domain Services](#)

[Admin Interfaces in Active Directory Domain Services](#)

[IDsAdminCreateObj](#)

[IDsAdminCreateObj::CreateModal](#)

[IDsAdminCreateObj::Initialize](#)

[IDsAdminNewObj](#)

[IDsAdminNewObj::GetPageCounts](#)

[IDsAdminNewObj::SetButtons](#)

[IDsAdminNewObjPrimarySite](#)

[IDsAdminNewObjPrimarySite::Commit](#)

[IDsAdminNewObjPrimarySite::CreateNew](#)

[IDsAdminNewObjExt](#)

[IDsAdminNewObjExt::AddPages](#)

[IDsAdminNewObjExt::GetSummaryInfo](#)

[IDsAdminNewObjExt::Initialize](#)

[IDsAdminNewObjExt::OnError](#)

[IDsAdminNewObjExt::SetObject](#)

[IDsAdminNewObjExt::WriteData](#)

[IDsAdminNotifyHandler](#)

[IDsAdminNotifyHandler::Begin](#)

[IDsAdminNotifyHandler::End](#)

[IDsAdminNotifyHandler::Initialize](#)

[IDsAdminNotifyHandler::Notify](#)

Display Interfaces in Active Directory Domain Services

[ICommonQuery](#)

[ICommonQuery::OpenQueryWindow](#)

[IDsBrowseDomainTree](#)

[IDsBrowseDomainTree::BrowseTo](#)

[IDsBrowseDomainTree::FlushCachedDomains](#)

[IDsBrowseDomainTree::FreeDomains](#)

[IDsBrowseDomainTree::GetDomains](#)

[IDsBrowseDomainTree::SetComputer](#)

[IDsDisplaySpecifier](#)

[IDsDisplaySpecifier::EnumClassAttributes](#)

[IDsDisplaySpecifier::GetAttributeADsType](#)

[IDsDisplaySpecifier::GetClassCreateInfo](#)

[IDsDisplaySpecifier::GetDisplaySpecifier](#)

[IDsDisplaySpecifier::GetFriendlyAttributeName](#)

[IDsDisplaySpecifier::GetFriendlyClassName](#)

[IDsDisplaySpecifier::GetIcon](#)

[IDsDisplaySpecifier::GetIconLocation](#)

[IDsDisplaySpecifier::IsClassContainer](#)

[IDsDisplaySpecifier::SetLanguageID](#)

[IDsDisplaySpecifier::SetServer](#)

[IPersistQuery](#)

[IPersistQuery::Clear](#)

[IPersistQuery::ReadInt](#)

[IPersistQuery::ReadString](#)

[IPersistQuery::ReadStruct](#)

[IPersistQuery::WriteInt](#)

[IPersistQuery::WriteString](#)

[IPersistQuery::WriteStruct](#)

[IQueryForm](#)

[IQueryForm::AddForms](#)

[IQueryForm::AddPages](#)

[IQueryForm::Initialize](#)

Object Picker Dialog Box Interfaces

[IDsObjectPickerCredentials](#)

[IDsObjectPickerCredentials::SetCredentials Method](#)

[IDsObjectPicker](#)

[IDsObjectPicker::Initialize](#)

[IDsObjectPicker::InvokeDialog](#)

Messages in Active Directory Domain Services

[WM_ADSPROP_NOTIFY_APPLY](#)

[WM_ADSPROP_NOTIFY_CHANGE](#)

[WM_ADSPROP_NOTIFY_ERROR](#)

[WM_ADSPROP_NOTIFY_EXIT](#)

[WM_ADSPROP_NOTIFY_FOREGROUND](#)

[WM_ADSPROP_NOTIFY_PAGEHWND](#)

[WM_ADSPROP_NOTIFY_PAGEINIT](#)

[WM_ADSPROP_NOTIFY_SETFOCUS](#)

[WM_ADSPROP_SHEET_CREATE](#)

[WM_DSA_SHEET_CLOSE_NOTIFY](#)

[WM_DSA_SHEET_CREATE_NOTIFY](#)

[Return Values for Functions in Active Directory Domain Services](#)

[Success](#)

[Backup Errors in Active Directory Domain Services](#)

[System Errors in Active Directory Domain Services](#)

[Directory Manager Errors](#)

[Logging and Recovery Errors in Functions in Active Directory Domain Services](#)

[User Interface Mappings in Active Directory Domain Services](#)

[Mappings for the Active Directory Users and Computers Snap-in](#)

[Computer Object User Interface Mapping](#)

[Domain Object User Interface Mapping](#)

[Group Object User Interface Mapping](#)

[Object Property Sheet](#)

[Organizational Unit User Interface Mapping](#)

[Printer Object User Interface Mapping](#)

[Shared Folder Object User Interface Mapping](#)

[User Object User Interface Mapping](#)

[WMI Provider Reference for Active Directory Domain Services](#)

[WMI Provider Classes in Active Directory Domain Services](#)

[ReplicationProvider1](#)

[MSAD_DomainController](#)

[ExecuteKCC Method of the MSAD_DomainController Class](#)

[MSAD_NamingContext](#)

[MSAD_ReplCursor](#)

[MSAD_ReplNeighbor](#)

[SyncNamingContext Method of the MSAD_ReplNeighbor Class](#)

[MSAD_ReplPendingOp](#)

Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Purpose

Microsoft Active Directory Domain Services are the foundation for distributed networks built on Windows 2000 Server, Windows Server 2003 and Microsoft Windows Server 2008 operating systems that use domain controllers. Active Directory Domain Services provide secure, structured, hierarchical data storage for objects in a network such as users, computers, printers, and services. Active Directory Domain Services provide support for locating and working with these objects.

This guide provides an overview of Active Directory Domain Services and sample code for basic tasks, such as searching for objects and reading properties, to more advanced tasks such as service publication.

Windows 2000 Server and later operating systems provide a user interface for users and administrators to work with the objects and data in Active Directory Domain Services. This guide describes how to extend and customize that user interface. It also describes how to extend Active Directory Domain Services by defining new object classes and attributes.

NOTE

The following documentation is for computer programmers. If you are an end-user trying to debug a printing error or home network issue, see the [Microsoft community forums](#).

Where applicable

Network administrators write scripts and applications that access Active Directory Domain Services to automate common administrative tasks, such as adding users and groups, managing printers, and setting permissions for network resources.

Independent software vendors and end-user developers can use Active Directory Domain Services programming to directory-enable their products and applications. Services can publish themselves in Active Directory Domain Services; clients can use Active Directory Domain Services to find services, and both can use Active Directory Domain Services to locate and work with other objects on a network.

Developer audience

Applications that access data in Active Directory Domain Services can be written using the [Active Directory Service Interfaces API](#), [Lightweight Directory Access Protocol API](#), or the [System.DirectoryServices](#) namespace.

Run-time requirements

Active Directory Domain Services run on Windows 2000 and later domain controllers. However, client applications can be written for and run on Windows Vista, Windows Server 2003, Windows XP, Windows 2000, Windows NT 4.0, Windows 98, and Windows 95.

In this section

[About Active Directory Domain Services](#)

General information about Active Directory Domain Services.

[Using Active Directory Domain Services](#)

Active Directory Domain Services programming guide.

[Active Directory Domain Services Reference](#)

Active Directory Domain Services programming reference.

About Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Writing Powerful Applications that Use Active Directory Domain Services

This guide provides essential information for integrating Active Directory Domain Services in distributed applications designed for operating systems that support Active Directory Domain Services, including:

- Windows Server 2008
- Windows Server 2008 R2
- Windows Server 2012
- Windows Server 2012 R2
- Windows Server 2016

NOTE

These topics are for software developers. For support issues, see [Microsoft Support](#). For information about administering Active Directory, see [Active Directory Domain Services](#) at TechNet.

Fundamental Directory Features

A directory service is a fundamental service for distributed applications. A directory service must provide the features listed in the following table.

FEATURE	DESCRIPTION
Location transparency	Able to find user, group, networked service, or resource, data without the object address
Object data	Able to store user, group, organization, and service data in a hierarchical tree
Rich query	Able to locate an object by querying for object properties
High availability	Able to locate a replica of the directory at a location that is efficient for read/write operations

Advanced Features of Active Directory Domain Services

Active Directory Domain Services provides the features listed in the following table.

FEATURE	DESCRIPTION
Support for Internet standards	Active Directory Domain Services implements its features in accordance with published Internet standards such as LDAP and DNS.

FEATURE	DESCRIPTION
Tightly integrated and flexible security	<p>Advantages include:</p> <ul style="list-style-type: none"> • Choice of authentication packages. Kerberos, Secure Sockets Layer (SSL), or a combination; for example, establish an SSL channel for encryption and then use Kerberos for authentication. • Central management of service and resource access by using the users and groups in Active Directory Domain Services. • Delegation of administration so that central administrators can delegate administrative tasks such as password changing or specific object creation and deletion. • The Active Directory server uses the same access control mechanisms used on file systems in the Windows operating systems. Thus, the same tools that manage access control on a file system work for Active Directory Domain Services. • Comprehensive Public Key infrastructure. The Microsoft Certificate Server and Smart Card support are integrated with Active Directory Domain Services to provide Smart Card logon and Certificate management.
Easily programmable	<p>The Active Directory server can be programmatically accessed and administered using the Active Directory Service Interfaces API, Lightweight Directory Access Protocol API, or the System.DirectoryServices namespace.</p>
Directory enabled system services	<p>Your client application can be easily deployed to distributed desktops by creating a Windows Installer package and using the application deployment feature available in the Windows operating systems.</p>
Key application integration	<p>Key distributed applications, such as Exchange, are integrated with Active Directory Domain Services. Thus, companies can reduce the number of directory services to be managed.</p>
Rich and extensible schema	<p>The schema defines what objects and properties can be written and read from a directory service. The Active Directory Schema is rich. Most of the objects and properties a service requires are available. If not, a distributed application can extend the schema to support the application requirements.</p>

For more information about Active Directory Domain Services, see:

- [Core Concepts of Active Directory Domain Services](#)
- [Active Directory Architecture](#)
- [Active Directory Schema](#)

Core Concepts of Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following topics are core concepts of Active Directory Domain Services:

- [Attributes](#)
- [Containers and Leaves](#)
- [Object Names and Identities](#)
- [Naming Contexts and Directory Partitions](#)
- [Domain Trees](#)
- [Forests](#)
- [Active Directory Servers and Dynamic DNS](#)
- [Replication and Data Integrity](#)

Attributes (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

Each object in Active Directory Domain Services contains a set of attributes that define the characteristics of the object. Each attribute is described by an [attributeSchema](#) object in the schema container that defines the attribute. The attribute definition includes a variety of data, for example, what object types that the attribute applies to and the syntax type of the attribute. For more information about attribute schema definitions, see [Characteristics of Attributes](#).

The following list lists the type of attributes that are stored in Active Directory Domain Services.

Domain-replicated, stored attributes

Some attributes are stored in the directory (such as [cn](#), [nTSecurityDescriptor](#), and [objectGUID](#)) and replicated to all domain controllers in a domain. A subset of these attributes is also replicated to the global catalog. If you enumerate attributes of an object from the global catalog, only the attributes replicated to the global catalog are returned. Some attributes are also indexed because including an indexed property in a query improves the query performance.

Non-replicated, locally stored attributes

Non-replicated attributes, such as [badPwdCount](#), [Last-Logon](#), and [Last-Logoff](#) are stored on each domain controller, but are not replicated. The non-replicated attributes are attributes that pertain to a particular domain controller. For example, [Last-Logon](#) attribute is the last date and time that the user's network logon was validated by that particular domain controller that returned the property. These attributes can be retrieved in the same way as the domain-wide attributes described previously. However, for these attributes, each domain controller stores only values that pertain to that particular domain controller. For example, to obtain the last time that a user logged on to the domain, retrieve the [Last-Logon](#) attribute for the user from every domain controller in the domain and find that latest date and time.

Non-stored, constructed attributes

A user object also has constructed attributes that are not stored in the directory, but are calculated by the domain controller, such as [canonicalName](#) and [allowedAttributes](#).

Containers and Leaves

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services contain a hierarchy of objects in which every object instance, except the root of the directory hierarchy, is contained by some other object. The structure of this hierarchy is more flexible than a file system of directories and files. Instead, rules, in the [Active Directory Schema](#), determine which object classes can contain instances of which other object classes. For example, the default schema definition of the [User](#) object class includes the [Organizational-Unit](#) and [Container](#) object classes as possible superiors; that is, possible parent-objects or containers of a [User](#) object instance. This means that an [Organizational-Unit](#) object can contain a [User](#) object, but a [User](#) object cannot contain another [User](#) object, unless the schema definition of the [User](#) class is changed.

Except for schema objects, that is, the [classSchema](#) or [attributeSchema](#) objects that define the classes and attributes that can exist in a server forest, any object in Active Directory Domain Services may be a container. Specifically, any object class that appears in the [possSuperiors](#) or [systemPossSuperiors](#) attribute of an object class definition is potentially a container. For more information about containers of a predefined object class, see [Active Directory Domain Services Reference](#). You can bind programmatically to the abstract schema and use the [IADsClass::get_Containment](#) or [IADsClass::get_PossibleSuperiors](#) methods to get the classes that a given class can contain or be contained by. For more information, see [Reading the Abstract Schema](#). You can also read the [possibleInferiors](#) attribute of any object instance to determine the object classes that the object can contain. Be aware that [possibleInferiors](#) is a constructed attribute, which means it is calculated from the [possSuperiors](#)/[systemPossSuperiors](#) values of the other class definitions and is not actually stored in the directory.

Be aware that the Active Directory Schema defines a [Container](#) class. As discussed previously, an object is not required to be an instance of the [Container](#) class to be a container. There is also a [Leaf](#) class, and although subclasses of this class are typically not containers, there is no reason why they cannot be.

Finally, you can set a flag on the display specifier associated with an object class to indicate that user interfaces should always display instances of the class as leaves rather than containers. This helps prevent the user interface from being cluttered by too many containers. For more information, see [Viewing Containers as Leaf Nodes](#).

Object Names and Identities

6/3/2022 • 2 minutes to read • [Edit Online](#)

An object in Active Directory Domain Services has several identities, including the following.

Relative Distinguished Name

The relative distinguished name is the name defined by an object's naming attribute. The **rDnAttID** attribute of a **classSchema** object identifies the naming attribute for instances of the class. Most object classes use the **cn** (Common-Name) attribute as the naming attribute. An object's relative distinguished name must be unique in the container where the object resides. There can be many object instances with the same relative distinguished name, but no two can be in same container. For more information about the **rDnAttID** attribute and **classSchema** objects, see [Characteristics of Object Classes](#).

Distinguished Name

The **distinguished name** is the current name of the object and is contained in the **distinguishedName** attribute of the object. The distinguished name is a string that includes the location of the object and is formed by concatenating the relative distinguished name of the object and each of its ancestors all the way to root. For example, the distinguished name of the Users container in the Fabrikam.com domain would be "CN=Users,DC=Fabrikam,DC=com". Distinguished names are unique within a forest. An object's distinguished name changes when the object is moved or renamed.

Object GUID

The object GUID is a globally unique identifier assigned by Active Directory Domain Services when the object instance is created. The object GUID is contained in the **objectGUID** attribute of the object. A GUID is a 128-bit number guaranteed to be unique in space and time. Object GUIDs never change, so if an object is renamed or moved anywhere in the enterprise forest, the object GUID remains the same. Applications that save references to objects in Active Directory Domain Services must use the object GUID to ensure that the object reference will survive a rename of the object. The distinguished name for an object might change, but the object GUID will not.

Other Identities

Object instances can have many other attributes, and the attributes can be used for identification by applications. For example, security principal objects (instances of the **user**, **computer**, and **group** object classes) have **userPrincipalName**, **sAMAccountName**, and **objectSid** attributes. These attributes are very important "names" for Windows 2000, but these are not part of the object identity from the directory's perspective.

Naming Contexts and Directory Partitions

6/3/2022 • 2 minutes to read • [Edit Online](#)

Each domain controller in a domain forest controlled by Active Directory Domain Services includes *directory partitions*. Directory partitions are also known as *naming contexts*. A directory partition is a contiguous portion of the overall directory that has independent replication scope and scheduling data. By default, the Active Directory Domain Service for an enterprise contains the following partitions:

- *Schema Partition*: The schema partition contains the **classSchema** and **attributeSchema** objects that define the types of objects that can exist in the forest. Every domain controller in the forest has a replica of the same schema partition.
- *Configuration Partition*: The configuration partition contains replication topology and other configuration data that must be replicated throughout the forest. Every domain controller in the forest has a replica of the same configuration partition.
- *Domain Partition*: The domain partition contains the directory objects, such as users and computers, associated with the local domain. A domain can have multiple domain controllers and a forest can have multiple domains. Each domain controller stores a full replica of the domain partition for its local domain, but does not store replicas of the domain partitions for other domains.

Windows Server 2003 introduces the *Application Directory Partition*, which provides the ability to control the scope of replication and allow the placement of replicas in a manner more suitable for dynamic data. For more information about application directory partitions, see [About Application Directory Partitions](#).

For more information about how Active Directory Domain Services maintain consistency between the various replicas of a directory partition, see [Replication and Data Integrity](#).

About Application Directory Partitions

6/3/2022 • 4 minutes to read • [Edit Online](#)

Developers who use ADSI or LDAP to store and access relatively static and globally data in Active Directory Domain Services may also prefer, for the sake of simplicity and uniformity, to continue using ADSI or LDAP access for their dynamic data requirements. Dynamic data changes more frequently than what has been recommended for storing in Active Directory Domain Services. Dynamic data typically changes more frequently than the replication latency involved in propagating the change to all replicas of the data.

In Windows 2000, the support for dynamic data is limited. Storing dynamic data in a domain partition can be complicated. The data is replicated to all domain controllers in the domain which is often unnecessary and can result in inconsistent data due to replication latency. This can adversely impact network performance. In addition, domain partitions are not effective for applications that must replicate data across domain boundaries. Another option in Windows 2000 is to store dynamic data in attributes marked as non-replicated. However, this arrangement is limited in that it has a single point of failure, namely, the single domain controller housing the only copy of the object's non-replicated attributes.

Application directory partitions provide the ability to control the scope of replication and allow the placement of replicas in a manner more suitable for dynamic data. As a result, the application directory partition provides the capability of hosting dynamic data in the Active Directory Server, thus allowing ADSI/LDAP access to it, without significantly impacting network performance.

The Windows 2000 DNS service is an example of a service that can take advantage of application directory partitions. In Windows 2000, if the DNS service is optionally configured to use Active Directory Domain Services, the DNS zone data is stored in the Active Directory Server in a domain partition. That is, the data is replicated to all domain controllers in the domain, regardless of whether a DNS server is configured to run on the domain controller. This is an instance where full domain-wide replication is unnecessary. By storing the DNS zone data in an application directory partition, the service can redefine the scope of replication to only that subset of domain controllers in the domain that actually run the DNS server.

Consider the following scenarios for hosting a replica of an application directory partition:

- A replica of an application directory partition can be created on any Windows Server 2003 operating system and later domain controller in an Active Directory Domain Services forest.
- The set of domain controllers that host replicas of an application directory partition can be specified. Unlike a domain partition, an application directory partition is not required to replicate to all domain controllers in a domain, and it can replicate to domain controllers in different domains of the forest.
- A domain controller with an application directory partition replica can coexist and function in a mixed-mode environment with other computers running Windows 2000 or Windows NT 4.0.

Types of data that can be stored in an application directory partition include:

- An application directory partition can contain instances of any object type except security principals, such as users, computers, or groups.
- Objects in an application directory partition can maintain DN-value references to other objects in the same application directory partition, to objects in the configuration and schema partitions, and to any naming context head (which is the top object of a directory partition, such as the **domainDNS** object at the top of an application directory partition).
- For more information and an example of the type of dynamic data that could be stored in an application directory partition, see [Dynamic Objects](#). Dynamic objects are supported beginning with Windows Server 2003. Dynamic objects have a property that determines the time-to-live, after which the object is

deleted by the Active Directory server.

Some limitations of application directory partitions include:

- Objects in an application directory partition cannot maintain *DN-value references* to objects in other application directory partitions or domain partitions. (DN-value references are persistent references to a distinguished name value such as "CN=someuser,DC=corp,DC=Fabrikam,DC=com"). Likewise, objects in Domain, Configuration, and Schema partitions cannot maintain DN-value references to objects in an application directory partition. A DN-value reference can be maintained to the naming context head.()
- Objects in an application directory partition are not replicated to the Global Catalog. As with any domain controller, a global catalog server can also be configured to contain a full replica of an application directory partition, but the application directory partition data is completely separate from the global catalog data.
- When an application directory partition replica is created, the Domain-Naming FSMO role must be on one of the Windows Server 2003 and later operating system domain controllers. After the application directory partition replica is created, the Domain Naming FSMO role can be assigned back to a Windows 2000 domain controller.
- Application directory partition objects cannot be moved to other Active Directory partitions outside the partition in which they were created.

Other application directory partition features include:

- The security and access control model for the application directory partition is the same as that for other partitions in Active Directory Domain Services.
- The time intervals that control the latency of initiating an originating change notification to replication partners within a site can be configured separately for each application directory partition basis.
- Application directory partitions can be named just as regular domains, attached anywhere in the Active Directory namespace where a domain can, and discovered using DNS even by down-level Windows 2000 systems.
- An application directory partition can be created, its replication scope defined, and its configurable settings adjusted programmatically using standard LDAP and ADSI APIs. For more information about programmatically manipulating application directory partitions, see [Application Directory Partitions](#).

Domain Trees

6/3/2022 • 2 minutes to read • [Edit Online](#)

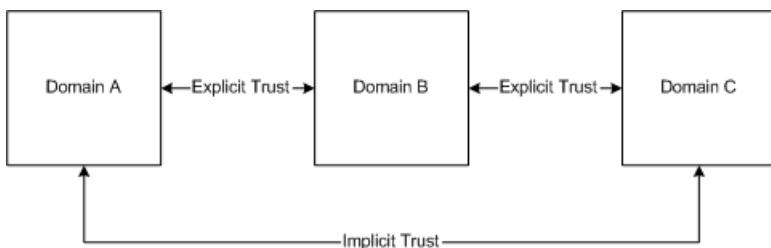
A *domain tree* is made up of several domains that share a common schema and configuration, forming a contiguous namespace. Domains in a tree are also linked together by trust relationships. Active Directory is a set of one or more trees.

Trees can be viewed two ways. One view is the trust relationships between domains. The other view is the namespace of the domain tree.

Viewing Trust Relationships

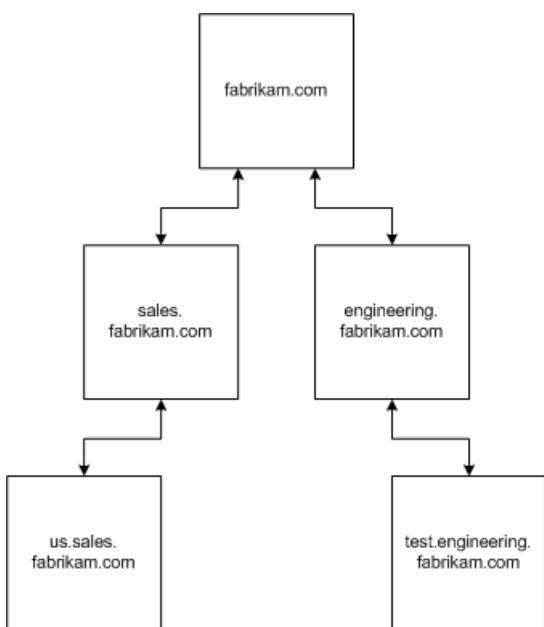
You can draw a diagram of a domain tree based on the individual domains and the existing trust relationship.

Windows 2000 establishes trust relationships between domains based on the Kerberos security protocol. Kerberos trust is transitive and hierarchical—if domain A trusts domain B, and domain B trusts domain C, then domain A trusts domain C.



Viewing the Namespace

You can also draw a diagram of a domain tree based on the namespace. You can determine an object's distinguished name by following the path up the hierarchy of the domain tree namespace. This view is useful for grouping objects into a logical hierarchy. The chief advantage of a contiguous namespace is that a deep search from the root of the namespace searches the entire hierarchy.

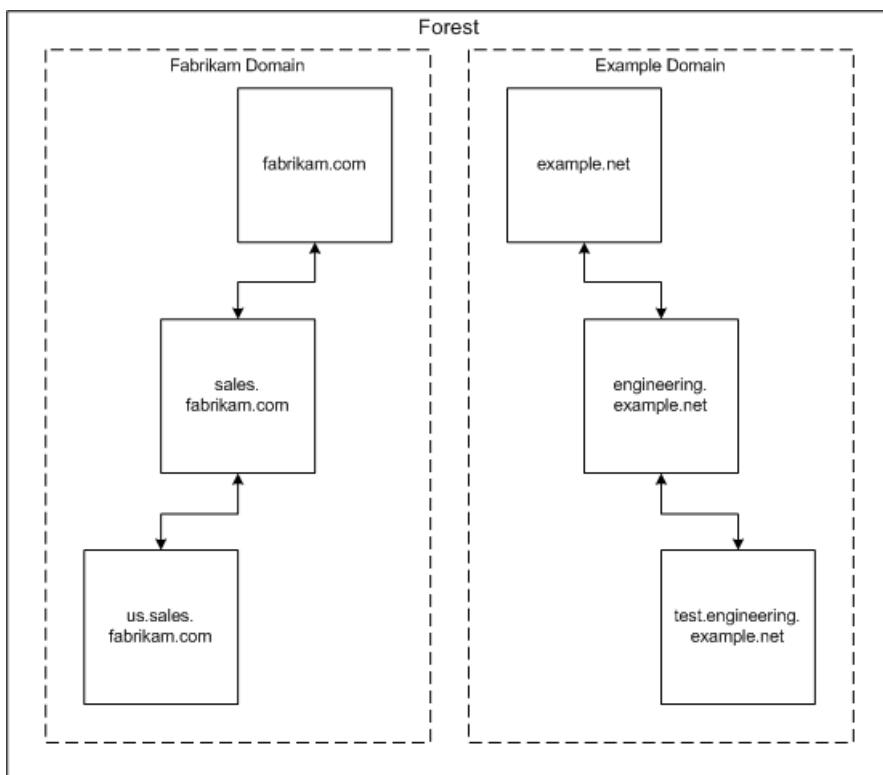


Forests

6/3/2022 • 2 minutes to read • [Edit Online](#)

A *forest* is a set of one or more domain trees that do not form a contiguous namespace. All trees in a forest share a common schema, configuration, and global catalog. All trees in a given forest exchange trust according to transitive hierarchical Kerberos trust relationships. Unlike trees, a forest does not require a distinct name. A forest exists as a set of cross-reference objects and Kerberos trust relationships recognized by the member trees. Trees in a forest form a hierarchy for the purposes of Kerberos trust; the tree name at the root of the trust tree refers to a given forest.

The following figure shows a forest of noncontiguous namespaces.



Active Directory Servers and Dynamic DNS

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory servers publish their addresses such that clients can find them knowing only the domain name. Active Directory servers are published using the Service Resource Records (SRV RRs) in DNS. The SRV RR is a DNS record used to map the name of a service to the address of a server that offers the service. The name of a SRV RR is in this form:

```
<service>.<protocol>.<domain>
```

Active Directory servers offer the LDAP service over the TCP protocol so that published names are "ldap.tcp. <domain>". Thus, the SRV RR for fabrikam.com is "ldap.tcp.fabrikam.com". Additional data about the SRV RR indicates the priority and weight for the server, enabling clients to choose the best server for their needs.

When an Active Directory server is installed, it uses Dynamic DNS to publish itself. Because TCP/IP addresses are subject to change, servers periodically verify their registrations to be sure they are correct, and update them if necessary.

Dynamic DNS is a recent addition to the DNS standard. Dynamic DNS defines a protocol for dynamically updating a DNS server with new data. Prior to Dynamic DNS, administrators were required to manually configure the records stored by DNS servers.

Replication and Data Integrity

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services provide *multi-master update*. Multi-master update means that all full replicas of a given partition are writable (the partial replicas on global catalog servers are not writable.) Multi-master update means that updates are not blocked even when some replicas are inoperable. The Active Directory server propagates the changes from the updated replica to all other replicas. Replication is automatic and transparent.

For more information, see the following topics.

- [The Replication Model in Active Directory Domain Services](#)
- [Replication Behavior in Active Directory Domain Services](#)
- [Detecting and Avoiding Replication Latency](#)

Replication Model in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

This section includes the following topics:

- [Features of the Replication Model for Active Directory Domain Services](#)
- [Why Active Directory Domain Services Uses This Replication Model](#)
- [A Programmer's Model of Replication in Active Directory Domain Services](#)

Features of the Replication Model for Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The replication model used in Active Directory Domain Services is called *multi-master loose consistency with convergence*. In this model, the directory can have many replicas; a replication system propagates changes made at any given replica to all other replicas. The replicas are not guaranteed to be consistent with each other at any particular time ("loose consistency"), because changes can be applied to any replica at any time ("multi-master"). If the system is allowed to reach a steady state, in which no new updates are occurring and all previous updates have been completely replicated, all replicas are guaranteed to converge on the same set of values ("convergence").

Why Active Directory Domain Services Uses This Replication Model

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic explains the reasons for the free-form system used by Active Directory Domain Services for a replication model.

Active Directory Domain Services are a free-form system for the following reasons:

- Customers require a highly distributed solution in which parts of the directory can be spread across their networks and administered locally.
- Large customers often grow to millions of objects, hundreds or thousands of replicas, or both.
- Many customer networks provide only intermittent connectivity to some locations; for example, remote oil drilling platforms and ships at sea, so the system must be tolerant of partly connected or disconnected operations.

There is no way to guarantee complete awareness of the current or future state of a distributed system because knowledge of state changes must be propagated and propagation takes time, during which more state changes may occur.

Tightly coupled systems handle uncertainty by attempting to eliminate it. This is accomplished through constraints on updates, requiring all nodes or some majority of nodes to be available before updates can be performed, using distributed locking schemes or single-mastering for critical resources, constraining all nodes to be well-connected, or some combination of these techniques. The more tightly coupled the computing nodes in a distributed system are, the lower the scaling limit.

Free-form systems handle uncertainty by tolerating it. A free-form system enables the nodes to have differing views of the overall system state and provides algorithms for resolving conflicts.

Tightly coupled solutions were rejected as unsuitable for Active Directory Domain Services because of the requirements for local administration, disconnected operation, and scalability to very large numbers of nodes. The loosely coupled model chosen, multi-master loose consistency with convergence, satisfies all requirements.

A Programmer's Model of Replication in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following is a description of the replication model for Active Directory Domain Services.

All updates to the Active Directory Domain Controller (DC) are performed using LDAP requests that create, modify, or delete one object for each request. A single request can set or modify multiple attributes on an object.

An update request is processed as an atomic transaction at some DC. Either the entire update happens or none of it does. If the requester receives a successful response to an update request, that entire request has succeeded (committed). This is called an "originating write." Multiple LDAP requests cannot be grouped into a single larger transaction.

In an originating write, a DC computes a "stamp" for each new or modified attribute value, and attaches this stamp to the value so when the value is replicated, the stamp is replicated too. The new stamp is unique, and in case of an update, the new stamp is larger than the stamp on the old value at that DC.

Occasionally, a DC selects the set of objects that have changed since the last time the DC performed replication. Then, for each object, it sends a single message to all other DCs that contain all the current values of attributes changed since the last time the object was replicated. Replication messages are reliable and are delivered in order, but may require more time to be delivered.

When one DC receives a replication message from another DC, it processes it as follows: For each modified attribute, if the stamp on the value in the replication message is larger than the stamp on the current value, the DC applies the update; otherwise the DC discards the update. Each replication message is applied as an atomic transaction, just like an originating write.

That is the Active Directory Domain Services replication model. Key properties of this model include:

- An originating write to a single object is atomic.
- When replicating changes, either all the changes made by an originating write are sent or none of them are.
- A replicated write to a single object is atomic, but conflicts are resolved attribute-by-attribute.

The model does not guarantee the replication ordering of changes made to different objects. Do not write applications that assume that changes are replicated in originating-write order. The model does not guarantee that if an attribute of an object is changed twice, both values will be replicated; Replication sends only the current value at the time of replication.

The model differs from reality in several ways that only affect performance. For example, the Active Directory Server sends replication messages containing the changes to multiple objects, but processes the contents of such a multi-object message as if it were a series of single-object messages. The Active Directory Server does not perform point-to-point replication as described in the model, but instead performs a more complex and more efficient transitive replication that is functionally equivalent to the model.

Replication Behavior in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Replication behavior is consistent and predictable; given a set of changes to a given replica, the outcome can be predicted—the changes will be propagated to all other replicas. Devising a reliable general model for predicting when the changes will be applied at all other replicas, or at a particular replica, is impossible, because the future state of the distributed system as a whole cannot be known. This is called "nondeterministic latency," and applications that use the directory must understand and allow for it.

The situation is not as complex as it might appear. There are only three states that an application must accommodate:

- **Version Skew:** None of the changes that are applied to a given source replica have propagated to a given destination replica. An application reading the source replica sees the new version of the information, while an application reading the destination sees the old version (or nothing, if the new information was added for the first time). Version skew applies to all directory service consumers.
- **Partial Update:** Some of the changes that are applied to a given source replica have propagated to a given destination replica. An application reading the source replica sees the new information, while an application reading the destination sees a mix of old and new information (or only some of the new information, if the new information was added for the first time). Partial update applies to directory service consumers that use two or more related objects to store their information.
- **Fully Replicated State:** All of the changes that are applied to a given source replica have propagated to a given destination replica. Applications at the source and destination replicas see the same information.

Impact on Directory-Enabled Applications

6/3/2022 • 4 minutes to read • [Edit Online](#)

Version Skew

Version skew occurs when applications read the same objects from different replicas before a change has replicated. Applications reading the remote replica recognize the unchanged object. Version skew is an issue when a given application or set of applications use the data in the directory to interoperate.

For example, an RPC Service can publish its endpoints in the directory using standard RPC APIs (such as [RpcNsBindingExport](#)). Clients connect to the service by looking up the desired endpoint in the directory ([RpcNsBindingLookupBegin](#), [RpcNsBindingLookupNext](#), and so on) and binding to it.

Assume that an RPC Service S₁ publishes endpoint E₁ and subsequently moves to a different computer. The original endpoint E₁ is changed to E_{S2}, reflecting the new computer's address. Clients reading remote replicas of the directory service are unable to connect to the service until the updated endpoint is replicated. However, moving a service from one computer to another is a rare occurrence; therefore, this interruption/delay should rarely be encountered, especially if the movement of the service is well-planned.

Partial Updates

In general, partial update occurs when one application is reading a set of data while another application is writing to that same set of data. This is a situation that can occur with any application in a multi-mastered system. There are many ways this can occur. You could have more than one application writing to the same data set at once. If you look at it this way, the directory replication service is just another application that could potentially be writing to the same data set that another application may be reading. This could be a potential problem for an application. However, the window of time in which a partial update can affect your application is relatively small. This should rarely if ever be an issue for applications that are not dependent on the synchronization of multiple objects. If your application is highly dependent on the synchronization of a related set of objects, you should consider the effects of a partial update in your application design.

In terms of the directory replication, partial update occurs when applications read the same set of objects from different replicas while replication is in progress. Applications at the remote replica see some, but not all, of the changes.

NOTE

The window in which partial update can affect an application is small: the application must start reading objects while inbound replication is in progress, after one or more of the related, changed objects have been received but before all have been received. The time between the updates at the source replica directly affects the size of this window—updates that occur close together in time are replicated close together in time. Partial update can be an issue when an application uses a related set of objects.

For example, a remote access service can use the directory to store policy and profile data. The policy data is stored in one set of objects, and the profile in another set. When a user connects to the remote access service, the remote access service reads the policy to determine whether the user is allowed to connect, and if so what profile to apply to the user session. Partial update can affect the remote access service in several ways:

- If the policy is complex and consists of multiple objects, the remote access service might read a partially

updated policy resulting in incorrect denial or granting of service to the user, inability to process the policy due to internal inconsistency, and so on.

- If both the policy and profiles have been updated, the service might correctly process the policy but apply a stale profile, because the policy objects have replicated but the profile objects have not.
- If the profile is complex and consists of multiple objects, the service might correctly process the policy but apply a partially updated profile because the policy objects have replicated, but only some of the profile objects have done so.

Collisions

Collisions occur when the same attributes of two or more replicas of a given object are changed during the same replication interval. The replication process reconciles the collision; because of reconciliation a user or application may "see" a value other than the one they wrote.

A simple example is user address information: A user changes a mailing address at replica R1 and an administrator changes the same mailing address at replica R2. A written value propagates until another value is selected over that value by the collision reconciliation mechanism. As long as a value continues to "win" against other values in the collision resolution process, the value continues to propagate. Ultimately, this "winning" value will be propagated to R1,R2, and all other replicas if no other changes are made.

Collision resolution is an issue for applications that make assumptions about the internal consistency of objects or sets of objects. For example, a network bandwidth allocation management service stores network bandwidth data for a given network segment in the directory in an object O1. The object contains the bandwidth available in bits-per-second and the maximum bandwidth any single user can reserve. The service expects that the user reservable bandwidth will always be less than or equal to the bandwidth available.

Consider the following sequence of events:

- The object in the initial fully replicated state gives the available bandwidth as 56 kilobits-per-second, and the user reservable bandwidth as 9,600 bits-per-second.
- An administrator at replica R1 changes the values to 64k and 19,200 respectively.
- An administrator at replica R1 changes the values to 10 million and 1 million respectively before the update from R1 arrives.

Assuming the attributes in question have equal version numbers when the updates occur, there is a small, but real, possibility that the object will end up with a maximum bandwidth of 64k and a user reservable maximum of 1 million—if the application performs the updates as separate write operations. The application should always update both properties in a single operation.

Detecting and Avoiding Replication Latency

6/3/2022 • 2 minutes to read • [Edit Online](#)

Replication latency is a fact of life in a loosely coupled distributed system. Applications must accommodate this. The best way to accommodate replication latency is to design applications to minimize the effects. The ideal directory-enabled application:

- Is insensitive to version skew.
- Does not depend on relationships among multiple objects.
- Has no intra- or inter-object consistency requirements.

Applications and services that fit this profile need not be concerned with replication latency. Other applications must be designed with replication latency in mind. The key to success in designing such an application is awareness of the replication process. Steps taken at design time to reduce inter-object dependencies and minimize partial update windows will pay large dividends at run time. Approaches to dealing with replication latency are divided into two classes—avoidance strategies that reduce the impact of latency and detection strategies that allow an application to detect latency-induced states.

What Can You Know, and When Can You Know It?

6/3/2022 • 2 minutes to read • [Edit Online](#)

Applications that are sensitive to latency-induced states must recognize problem states and take appropriate action. There are two problem states: version skew and partial update. Version skew is not detectable by examining the Directory Service (remember the axiom of distributed computing stated in [Why Active Directory Domain Services Use This Replication Model](#)). Partial update can be detected by adding metadata to the objects that compose the related set. Suggestions for such metadata appear in subsequent sections of this document. There are avoidance strategies applications can take to reduce the possibility of both version skew and partial update. These strategies are also discussed in subsequent sections of this document.

Temporal Locality

6/3/2022 • 2 minutes to read • [Edit Online](#)

Temporal locality is an avoidance strategy that reduces the window for partial updates. Replication of changes from a given source replica proceeds in ascending USN order. Writes that occur close together in time will have USNs that are "near" each other, and will be propagated close together in time. Applications that create or update multiple, related objects should write the objects as close together in time as possible. For example, the application could gather all necessary input from the user and apply it to the directory when the user clicks an "Apply" button.

Temporal locality also reduces the odds of collision resolution introducing intra-object inconsistencies.

Out-of-Band Signaling

6/3/2022 • 2 minutes to read • [Edit Online](#)

Cooperating applications that share common data using the directory can use out-of-band signaling to avoid both version skew and partial updates. Put simply, the cooperating applications use a mechanism separate from directory replication to inform each other of changes in the shared common data. Out-of-band signaling is most effective when used in combination with a versioning mechanism—this allows the partner detecting the change to notify remote partners what version of the shared data to wait for.

Returning to the RPC example given in the "Version Skew" section of [Replication Behavior in Active Directory Domain Services](#), the RPC server could call back into connected clients to inform them of the move to a new server: "I am going to move. Please stand by, replication will bring you the new address shortly" so that the client can handle the transition period.

Applications that require identical policies to be in force in order to communicate with each other can use out-of-band signaling to ensure that a new policy is not employed until all involved parties have received it. For example, if the Internet Protocol security (IPsec) policy between two machines is changed, an agent on the source machine could contact an agent on the destination machine to negotiate the application of the changed policy, with the source machine delaying application until the destination machine has received the new policy as well.

Effective Date and Time

6/3/2022 • 2 minutes to read • [Edit Online](#)

Effective date and time is an avoidance strategy that prevents version skew and partial updates by deferring the effective data of information to some point in the future when the change has a high probability of being fully replicated. To implement effective dates, the application objects must include an effective date timestamp, which is filled in when the object is created or changed. Consumers of the objects verify the timestamp and defer use of the objects until that date and time are reached.

Some important considerations:

- Applications that choose this approach must ensure that there is a set of applicable data to use until the updated objects become effective.
- Distributed time service in Windows NT 4.0 keeps the clocks of the connected Windows 2000 synchronized. No time service is perfect, however, so there is a small window for version skew.
- Setting a good effective date presumes knowledge of the overall replication latency for the distributed system in question. In a stable network a good rule of thumb for effective dates is the (time of the update) + (2*(average overall latency)). So, for a system whose overall latency averages 4 hours, a delay of 8 hours is reasonable.
- In an unstable network, determining a "good" value for effective dates is much more difficult, since the latency may be highly variable. Effective date is most "effective" in an unstable network when combined with other avoidance or detection strategies such as checksums or consistency GUIDs. For more information, see [Checksums and Object Counts](#).

Checksums and Object Counts

6/3/2022 • 2 minutes to read • [Edit Online](#)

Checksums and object counts are detection strategies that allow an application to detect a partial update state. Checksums can also be used to detect inconsistencies introduced by collision resolution. Both checksums and object counts require a place to store the value used for verifying a checksum or object count. This can be on a "master" object chosen from among those involved in the application-specific relationship or on a parent object under which the related objects are stored.

For checksums, applications reading the related objects verify the checksum by calculating a local result and comparing it with the stored value. If the values do not match, the replica is in a partial update state and the objects cannot be used.

For object counts, applications count the related objects (typically children of a single parent) and compare the count with the stored value. If the counts do not match, the replica is in a partial update state and the objects cannot be used.

Some important considerations:

- For the checksum approach to work, the one or more attributes used in computing the checksum must be updated. The algorithm used to compute the checksum must reliably reflect differences in input. If many different inputs product the same checksum, the algorithm will not reliably detect partial updates. "Salting" the input with values like the **objectGUID** of the source computer and the date and time of the update is also helpful.
- Object counts work best when used with new sets of objects, or in combination with consistency GUIDs (see the next section for more information). The application performing the update must either know, in advance, the number of objects that will be in the container when the update is completed or use some other means of marking the container invalid while the update proceeds (for example, setting the count to zero). After completing the update the source application marks the container with the count of objects contained.

Consistency GUIDs

6/3/2022 • 2 minutes to read • [Edit Online](#)

Consistency GUIDs are a detection strategy that allows an application to detect partial updates. A consistency GUID (Globally Unique IDentifier) is applied to each object in a related set. In implementation, a source application generates a new GUID and applies it to each object it updates in the set of related objects. It then applies the new GUID to the rest of the objects in the set, and finishes by applying the new GUID to the "master" object. Typically, the "master" object will be a container that is the parent of the other objects in the set.

Some important considerations:

- Consistency GUIDs combined with object counts or checksums are more effective than consistency GUIDs alone, because the application reading the objects may not know how many objects with the GUID should be present.
- Applications must generate their own GUIDs (a Microsoft Win32 API, UuidCreate, provides this function), and not use the system-generated GUIDs found in an object's **objectGUID** attribute. This is because a consistency GUID needs to change each time the set of objects is updated. Object identity GUIDs found in **objectGUID** never change after the object has been created.
- Consistency GUIDs assume that no object is shared among sets, so each set can have a unique consistency GUID.

Versioning and Fallback Strategies

6/3/2022 • 2 minutes to read • [Edit Online](#)

When an application detects a partial update using one of the preceding techniques or reads a set of objects whose effective date has not yet been reached, the application must deal with the situation gracefully. For some applications, the graceful response is to "fall back" to a previous version of the objects in question. Active Directory Domain Services do not provide a versioning facility—applications that want this capability must provide it themselves. Approaches to versioning include keeping the "last known good" values cached locally and storing multiple sets of objects in the directory, for example, in "old," "current," and "new" containers. Many other schemes are possible.

Implementations must take care to avoid unintended consequences. An earlier version of objects should be used only when a partial update is detected or the new objects are not yet "effective." Falling back because something in the application "does not work" might circumvent an administrator's intent. For example, two computers that formerly could communicate might find themselves unable to do so because of a change in Internet Protocol security (IPsec) policy. If this is intentional on the part of the administrator, the affected systems should not fall back to the policy that allowed them to communicate, as this would be a security breach.

Active Directory Domain Services Architecture

6/3/2022 • 2 minutes to read • [Edit Online](#)

This section introduces the primary architectural components of Active Directory Domain Services.

- [Directory System Agent](#)
- [Data Model](#)
- [Schema](#)
- [Administration Model](#)
- [Global Catalog](#)
- [Active Directory Security](#)

Directory System Agent

6/3/2022 • 2 minutes to read • [Edit Online](#)

The *directory system agent* (DSA) is a collection of services and processes that run on each domain controller and provides access to the data store. The data store is the physical store of directory data located on a hard disk. In Active Directory Domain Services, the DSA is part of the local system authority (LSA) subsystem. Clients access the directory using one of the following mechanisms supported by the DSA:

- LDAP clients connect to the DSA using the Lightweight Directory Access Protocol (LDAP). Active Directory Domain Services support LDAP 3.0, defined by RFC 3377, and LDAP 2.0, defined by RFC 1777. Windows clients use LDAP 3.0 to connect to the DSA.
- MAPI clients such as Microsoft Exchange Server connect to the DSA using the MAPI remote procedure call interface.
- The DSAs in Active Directory Domain Services connect to each other to perform replication using a proprietary remote procedure call interface.

Data Model

6/3/2022 • 2 minutes to read • [Edit Online](#)

The data model for Active Directory Domain Services is derived from the X.500 data model. The directory holds objects that represent things of various sorts, described by attributes. The universe of objects that can be stored in the directory is defined in the schema. For each object class, the schema defines what attributes an instance of the class must have, what additional attributes it may have, and what object class can be a parent of the current object class.

Schema (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory schema is implemented as a set of object class instances stored in the directory. This is very different than many directories that have a schema but store it as a text file read at startup. Storing the schema in the directory has many advantages. For example, user applications can read it to discover what objects and properties are available.

Active Directory schema can be updated dynamically. That is, an application can extend the schema with new attributes and classes and use the extensions immediately. Schema updates are accomplished by creating or modifying the schema objects stored in the directory. Like every object in Active Directory, access-control lists (ACLs) protect schema objects, so authorized users only may alter the schema.

For more information, see [Active Directory Schema](#).

Administration Model

6/3/2022 • 2 minutes to read • [Edit Online](#)

Authorized users perform administration in Active Directory Domain Services. A user is authorized by a higher authority to perform a specified set of actions on a specified set of objects and object classes in some identified subtree of the directory. This is called *delegated administration*. Delegated administration allows very fine-grained control over who can do what and enables delegation of authority without granting elevated privileges.

Global Catalog

6/3/2022 • 2 minutes to read • [Edit Online](#)

A Domain run by Active Directory Domain Services can consist of many partitions or naming contexts. The distinguished name (DN) of an object includes enough information to locate a replica of the partition that holds the object. Many times however, the user or application does not know the DN of the target object or which partition might contain the object. The [*global catalog \(GC\)*](#) allows users and applications to find objects in an Active Directory domain tree, given one or more attributes of the target object.

The global catalog contains a partial replica of every naming context in the directory. It contains the schema and configuration naming contexts as well. This means the GC holds a replica of every object in the directory but with only a small number of their attributes. The attributes in the GC are those most frequently used in search operations (such as a user's first and last names or login names) and those required to locate a full replica of the object. The GC allows users to quickly find objects of interest without knowing what domain holds them and without requiring a contiguous extended namespace in the enterprise.

The global catalog is built automatically by the Active Directory Domain Services replication system. The replication topology for the global catalog is generated automatically. The properties replicated into the global catalog include a base set defined by Microsoft. Administrators can specify additional properties to meet the needs of their installation.

Security in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services are part of the Windows 2000 trusted computing base and are a full participant in the Windows 2000 security infrastructure. Every object in an Active Directory Server is protected by its own security descriptor. The system validates any attempt to access an object or attribute in an Active Directory Server by verifying the access rights granted by the object security descriptor.

The following topics discuss important elements of security in Active Directory Domain Services:

- [Object and Attribute Protection](#)
- [Delegation](#)
- [Inheritance](#)

For more information and code examples, see [Controlling Access to Objects in Active Directory Domain Services](#).

Object and Attribute Protection

6/3/2022 • 2 minutes to read • [Edit Online](#)

An access-control list (ACL) protects all objects in Active Directory Domain Services. ACLs determine who can view the object, what attributes they can read, and what actions each user can perform on the object. The existence of an object or an attribute is never revealed to an unauthorized user.

An ACL is a list of access-control entries (ACEs) stored with the object that it protects. In Windows NT and Windows 2000, an ACL is stored as a binary value, called a security descriptor. Each ACE contains a security identifier (SID), which identifies the principal (user or group) to whom the ACE applies, and data about what type of access the ACE grants or denies.

ACLs for directory objects contain ACEs that apply to the object as a whole and ACEs that apply to the individual attributes of the object. This allows an administrator to control not only which users can see an object, but also what properties those users can view. For example, all users might be granted read access to the email and telephone number attributes for all other users, but security properties of users might be denied to all but members of a special security administrators group. Individual users might be granted write access to personal attributes such as the telephone and mailing addresses on their own user objects.

Delegation

6/3/2022 • 2 minutes to read • [Edit Online](#)

Delegation is one of the most important security features of Active Directory Domain Services. Delegation enables a higher administrative authority to grant specific administrative rights for containers and subtrees to individuals and groups. Domain administrators, with broad authority over large segments of users, are no longer required.

An ACE can grant specific administrative rights for the objects in a container to a user or group. Rights are granted for specific operations on specific object classes using ACEs in the container's ACL. For example, to enable a user named "user 1" to be an administrator of the "Corporate Accounting" organizational unit, add ACEs to the ACL on "Corporate Accounting" as follows:

```
"";user 1";Grant ;Create, Modify, Delete;Object-Class User"user 1";Grant ;Create, Modify, Delete;Object-Class Group"user 1";Grant ;Write;Object-Class User; Attribute Password"
```

Now, user 1 can create new users and groups in Corporate Accounting and set the passwords on existing users, but user 1 cannot create any other object classes and cannot affect users in any other containers, unless, of course, user 1 is granted that access by ACEs on the other containers.

Inheritance (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

Inheritance allows a given ACE to be propagated from the container where it was applied to all children of the container. Inheritance can be combined with delegation to grant administrative rights to a whole subtree of the directory in a single operation.

Active Directory Schema (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory schema contains formal definitions of every object class that can be created in an Active Directory forest. The schema also contains formal definitions of every attribute that can exist in an Active Directory object.

Topics that provide an overview of the Active Directory schema include:

- [Schema Implementation](#)
- [Characteristics of Object classes](#)
- [Characteristics of Attributes](#)
- [The Abstract Schema](#)

For more information about schema programming, including reading the schema and defining new classes and attributes in the schema, see [Extending the Schema](#).

For more information, see the [Active Directory Domain Services Reference](#). For the reference pages for the predefined schema classes, attributes, and attribute syntaxes, see the [Active Directory Schema Reference](#).

About the Active Directory Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

Every object in Active Directory Domain Services is an instance of an object class defined in the Active Directory schema. An object class represents a category of objects, such as users, printers, or application programs, that share a set of common characteristics. The definition for each object class contains a list of the attributes that can be used to describe instances of the class. For example, the User class has attributes such as `givenName`, `surname`, and `streetAddress`. The list of attributes for a class is divided into those that an object of that class must contain, and additional attributes that an object may contain. The directory is arranged in a hierarchy; the definition of each class also lists the classes whose objects can be parents of objects of a given class—this controls the shape of the directory hierarchy.

The schema also formally defines each attribute. The definition for each attribute includes unique identifiers for the attribute, the syntax for the attribute (that is, the data type for the attribute's values), optional range limits for the attribute values, whether the attribute can have only one value or multiple values, and whether the attribute is indexed. The directory schema defines each attribute exactly once—object class definitions contain references to the attributes defined in the schema. For example, the "description" attribute is defined only once and is referenced by many object classes. This is different than a relational database schema, where the "attributes" (columns) are separately defined for each table.

Read-Only DCs and the Active Directory Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

Windows Server 2008 introduces a new type of domain controller, the Read-only Domain Controller (RODC). This provides a domain controller for use at branch offices where a full domain controller cannot be placed. The intent is to allow users in the branch offices to logon and perform tasks like file/printer sharing even when there is no network connectivity to hub sites.

RODC does not change the way schema is used. However, it is worthwhile to mention that the schema supports a Read-Only Partial Attribute Set (RO-PAS), also referred to as an RODC filtered attribute set, which is a special attribute set that is NOT replicated to RODCs for security reasons. RO-PAS are defined in the schema via the [searchFlags](#) attribute.

RODC filtered attribute set

Some applications that use [Active Directory Domain Services](#) as a data store may have credential-like data (such as passwords, credentials, or encryption keys) that should not be stored on a Read-Only Domain Controller in case the RODC is stolen or compromised. For this type of application, you can add the attribute to the RODC filtered attribute set to prevent it from replicating to RODCs in the forest, and mark the attribute as confidential, which removes the ability to read the data for members of the Authenticated Users group (including any RODCs).

Adding attributes to the RODC filtered attribute set

The RODC filtered attribute set is a dynamic set of attributes that is not replicated to any RODCs in the forest. You can configure the RODC filtered attribute set on a schema master that runs Windows Server 2008. When the attributes are prevented from replicating to RODCs, that data cannot be exposed unnecessarily if an RODC is stolen or compromised.

You cannot add system-critical attributes to the RODC filtered attribute set. An attribute is system critical if it is required for AD DS, Local Security Authority (LSA), Security Accounts Manager (SAM), and any of Microsoft-specific Security Service Providers, such as the Kerberos authentication protocol, to function properly. In releases of Windows Server 2008 after Beta 3, a system-critical attribute has a schemaFlagsEx attribute value of (schemaFlagsEx attribute value & 0x1 = TRUE).

For step by step instructions to adding attributes to the RODC filtered attribute set, see [Appendix D of the step-by-step guide for RODCs](#).

Marking attributes as confidential

In addition, it is recommended that you also mark as confidential any attributes that you configure as part of the RODC filtered attribute set. To mark an attribute confidential, you have to remove the Read permission for the attribute for the Authenticated Users group. Marking the attribute as confidential provides an additional safeguard against an RODC that is compromised by removing the permissions that are necessary to read the credential-like data. For more information about marking attributes as confidential, see [article 922836 in the Microsoft Knowledge Base](#).

Related topics

[Step-by-Step Guide for Read-only Domain Controllers](#)

Schema Implementation

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Active Directory Domain Services, class and attribute definitions are stored in the directory as instances of the **classSchema** and **attributeSchema** classes, respectively. **classSchema** and **attributeSchema** are classes defined in the schema. To manipulate the Active Directory schema, use the same LDAP operations used to manipulate other object. Because the schema is a key part of the directory that affects the entire forest, special restrictions apply to schema extensions. For more information about restrictions, see [Restrictions on Schema Extensions](#).

To summarize the schema implementation:

- Instances of the **classSchema** class define every object class supported by Active Directory Domain Services. The attributes of a **classSchema** object, for example, its **mayContain** and **mustContain** attributes, describe an object class, the same way that the attributes of a user object, for example, its **userPrincipalName** and **telephoneNumber** attributes, describe that user. For more information, see [Characteristics of Object Classes](#).
- Instances of the **attributeSchema** class are used to define every attribute supported by Active Directory Domain Services. The attributes of an **attributeSchema** object, for example, its **attributeSyntax** and **isSingleValued** attributes, describe an attribute, the same way the attributes of a user object describe that user. For more information, see [Characteristics of Attributes](#).
- Instances of the **attributeSchema** and **classSchema** classes are stored in a well-known place in the directory, the schema container. The schema container always has a distinguished name of the form:

```
CN=Schema,CN=Configuration,<DC=forestroot>
```

where "<DC=forestroot>" is the distinguished name of the root of the forest, for example, "DC=Fabrikam,DC=Com".

To get the distinguished name of the schema container, read the **schemaNamingContext** attribute of rootDSE. For more information about rootDSE and its attributes, see [Serverless Binding and RootDSE](#).

When thinking about the schema, remember:

- Schema changes are global. There is a single schema for an entire forest. The schema is globally replicated: a copy of the schema exists on every domain controller in the forest. When you extend the schema, you do so for the entire forest.
- Schema additions are not reversible. When a new class or attribute is added to the schema, it cannot be removed. An existing attribute or class can be disabled, but not removed. For more information, see [Disabling Existing Classes and Attributes](#).
- Disabling a class or attribute does not affect existing instances of the class or attribute, but it prevents new instances from being created. You cannot disable an attribute if it is included in any class that is not disabled.

Object Identifiers (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

Object Identifiers (OIDs) are unique numeric values issued by various issuing authorities to uniquely identify data elements, syntaxes, and other parts of distributed applications. OIDs are found in OSI applications, X.500 Directories, SNMP, and other applications where uniqueness is important. OIDs are based on a tree structure, in which a superior issuing authority, such as the ISO, allocates a branch of the tree to a subauthority, who in turn can allocate subbranches.

The LDAP protocol (RFC 2251) requires a directory service to identify object classes, attributes, and syntaxes with OIDs. This is part of the LDAP X.500 legacy.

OIDs in Active Directory Domain Services include some issued by the ISO for X.500 classes and attributes, and some issued by Microsoft and other issuing authorities. OID notation is a dotted string of numbers, for example "1.2.840.113556.1.5.9", which is described in the following table.

VALUE	MEANING	DESCRIPTION
1	ISO	Identifies the root authority.
2	ANSI	Group designation assigned by ISO.
840	USA	Country/region designation assigned by the group.
113556	Microsoft	Organization designation assigned by the country/region.
1	Active Directory	Assigned by the organization.
5	Classes	Assigned by the organization.
9	user class	Assigned by the organization.

For more information, and a discussion of two procedures used to obtain valid OIDs for use in extending the Active Directory schema, see [Obtaining an Object Identifier](#).

Characteristics of Object Classes

6/3/2022 • 8 minutes to read • [Edit Online](#)

Each object class in Active Directory Domain Services is defined by a **classSchema** object in the schema container. The attributes of a **classSchema** object specify the characteristics of the class, such as:

- Class identifiers: Classes have several identifiers including **ldapDisplayName**, which are used by LDAP clients to identify the class in search filters, and **schemaIDGUID**, which are used in security descriptors to control access to the class.
- Possible attributes: An object class definition includes lists of the mandatory and optional attributes that can be set on an instance of the class.
- Possible parents: Every object instance, except the root of the directory hierarchy, has exactly one parent. An object class definition includes lists of possible parents, that is, of the object classes that can contain an instance of the class.
- Superclasses and auxiliary classes: Every object class (except *top*) is derived from another class. A class inherits possible attributes and possible parents from the classes above it in the class hierarchy. A class can also have any number of auxiliary classes from which it inherits lists of possible attributes. For more information, see [Class Inheritance in the Active Directory Schema](#).

The following table lists the **IDAPDisplayName** and description of the key attributes of a **classSchema** object. For more information, and a complete list of the mandatory and optional attributes of a **classSchema** object, see [classSchema](#).

LDAPDISPLAYNAME	DESCRIPTION
cn	Every object in Active Directory Domain Services has a naming attribute from which its Relative Distinguished Name (RDN) is formed. The naming attribute for classSchema objects is cn (Common-Name). The value assigned to cn is the value that the object class will have as its RDN. For example, the cn of the organizationalUnit object class is Organizational-Unit , which would appear in a distinguished name as CN=Organizational-Unit . The cn must be unique in the schema container.
IDAPDisplayName	The name used by LDAP clients, such as the ADSI LDAP provider, to refer to the class, for example to specify the class in a search filter. A class's IDAPDisplayName must be unique in the schema container, which means it must be unique across all classSchema and attributeSchema objects. For more information about composing a cn and an IDAPDisplayName for a new class, see Naming Attributes and Classes .
schemaIDGUID	A GUID stored as an octet string. This GUID uniquely identifies the class. This GUID can be used in access control entries to control access to objects of this class. For more information, see Setting Permissions on Child Object Operations . On creation of the classSchema object, the Active Directory server generates this value if it is not specified. If you create a new class, generate your own GUID for each class so that all installations of your extension use the same schemaIDGUID to refer to the class.

LDAP DisplayName	Description
adminDisplayName	A display name of the class for use in administrative tools. If adminDisplayName is not specified when a class is created, the system uses the Common-Name value as the display name. This display name is used only if a mapping does not exist in the classDisplayName property of the display specifier for the class. For more information, see Display Specifiers and Class and Attribute Display Names .
governsID	The OID of the class. This value must be unique among the governsIDs of all classSchema objects and the attributeIDs of all attributeSchema objects. For more information, see Object Identifiers .
rDnAttId	Identifies the naming attribute, which is the attribute that provides the RDN for this class if different than the default (cn). Use of a naming attribute other than cn is discouraged. Naming attributes should be drawn from the well-known set (OU, CN, O, L, and DC) that is understood by all LDAP version 3 clients. For more information, see Object Names and Identities and Syntaxes for Attributes in Active Directory Domain Services . A naming attribute must have the Directory String syntax. For more information, see Syntaxes for Attributes in Active Directory Domain Services .
mustContain, systemMustContain	A pair of multi-valued properties that specify the attributes that must be present on instances of this class. These are mandatory attributes that must be present during creation and cannot be cleared after creation. After creation of the class, these properties cannot be changed. The full set of mandatory attributes for a class is the union of the systemMustContain and mustContain values on this class and all inherited classes.
mayContain, systemMayContain	A pair of multi-valued properties that specify the attributes that MAY be present on instances of this class. These are optional attributes that are not mandatory and, therefore, may or may not be present on an instance of this class. You can add or remove mayContain values from an existing category 1 or category 2 classSchema object. Before removing a mayContain value from a classSchema object, you should search for instances of the object class and clear any values for the attribute that you are removing. After creation of the class, the systemMayContain property cannot be changed. The full set of optional attributes for a class is the union of the systemMayContain and mayContain values on this class and all inherited classes.
possSuperiors, systemPossSuperiors	A pair of multi-valued properties that specify the structural classes that can be legal parents of instances of this class. The full set of possible superiors is the union of the systemPossSuperiors and possSuperiors values on this class and any inherited structural or abstract classes. systemPossSuperiors and possSuperiors values are not inherited from auxiliary classes. You can add or remove possSuperiors values from an existing category 1 or category 2 classSchema object. After creation of the class, the systemPossSuperiors property cannot be changed.

LDAP DisplayName	Description
objectClassCategory	<p>An integer value that specifies the category of the class, which can be one of the following:</p> <ul style="list-style-type: none"> • Structural, meaning that it can be instantiated in the directory. • Abstract, meaning that the class provides a basic definition of a class that can be used to form structural classes. • Auxiliary, meaning that a class that can be used to extend the definition of a class that inherits from it but cannot be used to form a class by itself. <p>For more information, see Structural, Abstract, and Auxiliary Classes.</p>
subClassOf	<p>An OID for the immediate superclass of this class, that is, the class from which this class is derived. For structural classes, subClassOf can be a structural or abstract class. For abstract classes, subClassOf can be an abstract class only. For auxiliary classes, subClassOf can be an abstract or auxiliary class. If you define a new class, ensure that the subClassOf class exists or will exist when the new class is written to the directory. If class does not exist, the classSchema object is not added to the directory.</p>
auxiliaryClass, systemAuxiliaryClass	<p>A pair of multi-valued properties that specify the auxiliary classes that this class inherits from. The full set of auxiliary classes is the union of the systemAuxiliaryClass and auxiliaryClass values on this class and all inherited classes. For an existing classSchema object, values can be added to the auxiliaryClass property but not removed. After creation of the class, the systemAuxiliaryClass property cannot be changed.</p>
defaultObjectCategory	<p>The distinguished name of this object class or one of its superclasses. When an instance of this object class is created, the system sets the objectCategory property of the new instance to the value specified in the defaultObjectCategory property of its object class. The objectCategory property is an indexed property used to increase the efficiency of object class searches. If defaultObjectCategory is not specified when a class is created, the system sets it to the distinguished name (DN) of the classSchema object for this class. If this object will be frequently queried by the value of a superclass rather than the object's own class, you can set defaultObjectCategory to the DN of the superclass. For example, if you are subclassing a predefined (category 1) class, the best practice is to set defaultObjectCategory to the same value as the superclass. This enables the standard UI to "find" your subclass.</p> <p>For more information, see Object Class and Object Category.</p>

LDAPDISPLAYNAME	DESCRIPTION
defaultHidingValue	<p>A Boolean value that specifies the default setting of the showInAdvancedViewOnly property of new instances of this class. Many directory objects are not interesting to end users. To keep these objects from cluttering the UI, every object has a Boolean attribute called showInAdvancedViewOnly. If defaultHidingValue is set to TRUE, new object instances are hidden in the Administrative snap-ins and the Windows shell. A menu item for the object class will not appear in the New context menu of the Administrative snap-ins even if the appropriate creation wizard properties are set on the object class's displaySpecifier object.</p> <p>If defaultHidingValue is set to FALSE, new instances of the object are displayed in the Administrative snap-ins and the Windows shell. Set this property to FALSE to see instances of the class in the administrative snap-ins and the shell and enable a creation wizard and its menu item in the New menu of the administrative snap-ins.</p> <p>If the defaultHidingValue value is not set, the default is TRUE.</p>
systemFlags	<p>An integer value that contains flags that define additional properties of the class. The 0x10 bit identifies a category 1 class (a class that is part of the base schema that is included with the system). You cannot set this bit, which means that the bit is not set in category 2 classes (which are extensions to the schema).</p>
systemOnly	<p>A Boolean value that specifies whether only the Active Directory server can modify the class. System-only classes can be created or deleted only by the Directory System Agent (DSA). System-only classes are those that the system depends on for normal operations.</p>
defaultSecurityDescriptor	<p>Specifies the default security descriptor for new objects of this class. For more information, see Default Security Descriptor and How Security Descriptors are Set on New Directory Objects.</p>
isDefunct	<p>A Boolean value that indicates whether the class is defunct. For more information, see Disabling Existing Classes and Attributes.</p>
description	<p>A text description of the class for use by administrative applications.</p>
objectClass	<p>Identifies the object class of which this object is an instance, which is the classSchema object class for all class definitions and the attributeSchema object class for all attribute definitions.</p>

Structural, Abstract, and Auxiliary Classes

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **objectClassCategory** attribute of a **classSchema** object can have a value, as listed in the following table, that indicates whether the class is structural, abstract, or auxiliary.

VALUE	DESCRIPTION
1	A structural class, which is the only type of class that can have instances in Active Directory Domain Services. A structural class can be a subclass of an abstract or structural class. A structural class can include any number of auxiliary classes in its definition.
2	An abstract class, which is a template used to derive new abstract, auxiliary, and structural classes. An abstract class can only be a subclass of an abstract class. Abstract classes cannot be instantiated in Active Directory Domain Services. An abstract class can include any number of auxiliary classes in its definition.
3	An auxiliary class, which can be included in the definition of a structural, abstract, or auxiliary class, in which case, the mustContain , systemMustContain , mayContain , and systemMayContain values of the auxiliary class are added to those of the class. An auxiliary class can be a subclass of an abstract or auxiliary class. Auxiliary classes cannot be instantiated in Active Directory Domain Services. An auxiliary class can include any number of auxiliary classes in its definition.

Do not confuse the **objectClassCategory** with an [object category](#).

Class Inheritance in the Active Directory Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

All object classes in an Active Directory directory service schema are derived from the special class **top**. With the exception of **top**, all object classes are subclasses of another object class. For example, **contact** is a subclass of **organizationalPerson**; **organizationalPerson** is a subclass of **person**; and **person** is a subclass of **top**. The **subClassOf** attribute of a **classSchema** object is a single-valued property that indicates the immediate superclass of the class.

Some of the attribute values that define a class are inherited from its superclasses. So the **contact** class inherits values from its superclasses, which are the **organizationalPerson**, **person**, and **top** classes. A class inherits the following data from its superclasses:

- Possible attributes: The values of the **mustContain**, **mayContain**, **systemMustContain**, and **systemMayContain** attributes of a **classSchema** object define a complete list of the attributes that can be set on an instance of the object class. For each object class, the values of these attributes include all of the values that are inherited from its superclasses, as well as any values that are set explicitly for the object class itself. Thus, the **mustContain** attribute of the **organizationalPerson** class includes all the **mustContain** values that are inherited from the **person** and **top** classes as well as any **mustContain** values that are set explicitly on the **organizationalPerson** class.
- Possible parents in the directory hierarchy: The values of the **possSuperiors** and **systemPossSuperiors** attributes of a **classSchema** object define a complete list of the object classes that can contain an instance of the object class. For each object class, the values include those inherited from its superclasses, as well as those set explicitly for the object class itself.

Be aware that object class can also have many auxiliary classes, which are specified in the **auxiliaryClass** and **systemAuxiliaryClass** attributes of a **classSchema** object. An object class inherits **mustContain**, **mayContain**, **systemMustContain**, and **systemMayContain** values from its auxiliary classes.

Object Class and Object Category

6/3/2022 • 2 minutes to read • [Edit Online](#)

Each instance of an object class has a multi-valued **objectClass** property that identifies the class of which the object is an instance, as well as all structural or abstract superclasses from which that class is derived. Thus, the **objectClass** property of a user object would identify the **top**, **person**, **organizationalPerson**, and **user** classes. The **objectClass** property does not include auxiliary classes in the list. The system sets the **objectClass** value when the object instance is created and it cannot be changed.

Each instance of an object class also has an **objectCategory** property, which is a single-valued property that contains the distinguished name of either the class of which the object is an instance or one of its superclasses. When an object is created, the system sets its **objectCategory** property to the value specified by the **defaultObjectCategory** property of its object class. An object's **objectCategory** property cannot be changed.

For more information, and a code example that retrieves an object's **objectClass** property, see [Retrieving the objectClass Attribute](#).

IMPORTANT

Prior to Windows Server 2008, the **objectClass** attribute is not indexed. This is because it has multiple values and is highly non-unique; that is, every instance of the **objectClass** attribute includes the **top** class. This means an index would be very large and ineffective. To locate objects of a given class, use the **objectCategory** attribute, which is single-valued and indexed. For more information about using these properties in search filters, see [Deciding What to Find](#).

For most classes, the **defaultObjectCategory** is the distinguished name of the class's **classSchema** object. For example, the **defaultObjectCategory** for the **organizationalUnit** class is "CN=Organizational-Unit,CN=Schema,CN=Configuration,<DC=forestroot>". However, some classes refer to another class as their **defaultObjectCategory**. This allows a query to readily find groups of related objects, even if they are of differing classes. For example, the **user**, **person**, **organizationalPerson**, and **contact** classes all identify the **person** class in their **defaultObjectCategory** properties. This allows search filters like (objectCategory=person) to locate instances of all these classes with a single query. Queries for people are very common, so this is a simple optimization.

If you create a subclass from a structural class, the best practice is to set the **defaultObjectCategory** value of the new class to the same distinguished name of the superclass. This allows the standard UI to "find" the subclass.

Characteristics of Attributes

6/3/2022 • 5 minutes to read • [Edit Online](#)

Each attribute in an Active Directory Domain Services object is defined by an **attributeSchema** object in the schema container. The properties of an **attributeSchema** object specify the characteristics of the attribute, such as:

- Attribute identifiers. Attributes have several identifiers, of which the most interesting from a programming perspective are the **IDAPDisplayName**, which is used by LDAP clients to read and write the attribute, and the **schemaIDGUID**, which is used in security descriptors to control access to the attribute.
- The type of data contained by instances of the attribute. An attribute's syntax properties determine the type of data, such as integer, string, or binary. Additional properties can specify the range of values that are allowed for the attribute and whether an instance of the attribute can have multiple values.
- Including the attribute in groups. Other properties tag an attribute to be included in a property set, which is a set of related properties, or to be included in the set of attributes that are replicated in the global catalog or indexed to optimize search performance.

The following table lists the **IDAPDisplayName** and description of the key properties of an **attributeSchema** object. For more information and a complete list of the mandatory and optional properties of an **attributeSchema** object, see [Attribute-Schema Class](#).

LDAPDISPLAYNAME	DESCRIPTION
cn	Every object in Active Directory Domain Services has a naming attribute from which its RDN is formed. The naming attribute for attributeSchema objects is cn (common name). The value assigned to cn is the value that the attributeSchema object will have as its RDN. For example, the cn of the isSingleValued object in the schema container is set as Is-Single-Valued, which would appear in a distinguished name as CN=Is-Single-Valued. The cn must be unique in the schema container.
IDAPDisplayName	The name used by LDAP clients, such as the ADSI LDAP provider, to read and write the attribute using the LDAP protocol. An attribute's IDAPDisplayName must be unique in the schema container, which means it must be unique across all classSchema and attributeSchema objects. For more information about composing a cn and an IDAPDisplayName for a new attribute, see Naming Attributes and Classes .
schemaIDGUID	A GUID stored as an octet string. This GUID uniquely identifies the attribute. This GUID can be used in access control entries to control access to instances of this attribute. For more information, see Setting Permissions to a Specific Property . On creation of the attributeSchema object, the Active Directory server generates this value if it is not specified. If you are creating a new attribute, it is recommended that you generate your own GUID for each attribute so that all installations of your extension will use the same schemaIDGUID to refer to the attribute.

LDAP DisplayName	Description
adminDisplayName	A display name of the attribute for use in administrative tools. If adminDisplayName is not specified when a class is created, the system uses the Common-Name value as the display name. This display name is used only if a mapping does not exist in the attributeDisplayNames property of the display specifier for the class. For more information, see Display Specifiers and Class and Attribute Display Names .
attributeID	The OID of this attribute. This value must be unique among the attributeID values of all attributeSchema objects and governIDs of all classSchema objects. For more information, see Object Identifiers .
attributeSecurityGUID	A GUID stored as an octet string. This is an optional GUID that identifies the attribute as a member of an attribute grouping; this is also called a property set. You can use this GUID in access control entries to control access to all attributes in the property set, that is, to all attributes that have the specified GUID set in their attributeSecurityGUID property. For more information, see Setting Permissions on a Group of Properties .
attributeSyntax	The object identifier of the syntax for this attribute. The combination of the attributeSyntax and oMSyntax properties determines the syntax of the attribute, that is, the type of data stored by instances of the attribute. For more information about the attributeSyntax , oMSyntax , and oMObjectClass syntax attributes, see Syntaxes for Attributes in Active Directory Domain Services .
oMSyntax	An integer that is the XDS representation of the syntax.
oMObjectClass	An octet string that must be specified for attributes of oMSyntax 127. For attributes with any other oMSyntax value, this property is not used. If no oMObjectClass is specified for an attribute with an oMSyntax of 127, the default oMObjectClass is set. Usually, there is a one-to-one mapping between the attributeSyntax and the oMObjectClass .
rangeLower , rangeUpper	A pair of integers that specify the lower and upper bounds of the range of values for this attribute. All values set for the attribute must be within or equal to the specified bounds. For attributes with numeric syntax the range specifies the minimum and maximum value. For attributes with string syntax the range specifies the minimum and maximum size, in characters. For attributes with binary syntax, the range specifies the number of bytes. If both rangeLower and rangeUpper are set, rangeLower must be less than rangeUpper . If one constraint is present without the other, the missing constraint is unbounded. For example, if the rangeLower for an integer is 3, and rangeUpper is absent, it means there is no upper constraint on the attribute. Likewise, if rangeUpper for a string is 2000, and rangeLower is absent, this indicates that there is no lower constraint on the length.

LDAP DisplayName	Description
isSingleValue	A Boolean value that is TRUE if the attribute can have only one value or FALSE if the attribute can have multiple values. If this property is not set, the attribute has a single value. Multi-valued attributes are unordered; there is no guarantee they will be stored or returned in any specific order. In the event of a replication collision, conflict resolution is for each attribute, not for each value within an attribute. The entire multi-value succeeds or fails. For more information about replication collision, see Consistency GUIDs .
searchFlags	Contains a set of flags that specify search and indexing information for an attribute. For more information, see Indexed Attributes .
isMemberOfPartialAttributeSet	A Boolean value that is TRUE if the attribute is replicated to the global catalog or FALSE if the attribute is not included in the global catalog. For more information, see Attributes Included in the Global Catalog .
linkID	An integer that indicates that the attribute is a linked attribute. An even integer is a forward link and an odd integer is a back link.
systemFlags	An integer value that contains flags that define additional properties of the attribute. For more information, see System-Flags .
systemOnly	A Boolean value that specifies whether only the Active Directory server can modify the attribute.
mAPIID	An integer by which MAPI clients identify this attribute.
isDefunct	A Boolean value that indicates whether the attribute is defunct. For more information, see Disabling Existing Classes and Attributes .
description	A text description of the attribute.
objectClass	Identifies the object class of which this object is an instance, which is the classSchema object class for all class definitions and the attributeSchema object class for all attribute definitions.

For more information about attributes, see:

- [Syntaxes for Attributes in Active Directory Domain Services](#)
- [Indexed Attributes](#)
- [Attributes Included in the Global Catalog](#)
- [Linked Attributes](#)

Syntaxes for Attributes in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services define a set of attribute syntaxes for specifying the type of data contained by an attribute. The predefined syntaxes do not actually appear in the directory, and you cannot add new syntaxes. Several methods can be used to identify the syntax of an attribute class:

- The [IADs.Get](#), [IADs.GetEx](#), [IADs.Put](#), and [IADs.PutEx](#) methods use the **VARIANT** structure to get and set the values of an object's attributes. The **vt** member of this structure is a **VARTYPE** value that identifies the data type.
- The methods of the [IDirectoryObject](#) and [IDirectorySearch](#) interfaces use a value from the **ADSTYPEENUM** enumeration to specify the data type.
- To specify the syntax of a new attribute class, set the **attributeSyntax** and **oMSyntax** attributes of an [attributeSchema](#) object. If the value of **oMSyntax** is 127, you must also set the **oMObjectClass** attribute. For more information, see [Choosing a Syntax](#).

For a complete list of the syntaxes provided by Active Directory Domain Services, including each syntax's corresponding **VARTYPE** and **ADSTYPEENUM** value, see [Syntaxes](#).

Indexed Attributes (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

Attributes may be indexed. Indexing an attribute can improve the performance of queries for that attribute.

An attributes is indexed when the **searchFlags** attribute in the attribute's schema definition has the least significant bit set to 1. Setting the least significant bit of the **searchFlags** attribute schema definition to 1 will dynamically build an index. Setting the least significant bit of the **searchFlags** attribute schema definition to 0 will cause the index for the attribute to be removed. The index will be built automatically by a background thread on the domain controller.

Ideally, indexed attributes should be single valued with highly unique values evenly distributed across the set of instances. The less unique the values of an attribute are, the less effective the index will be.

Multi-valued attributes can also be indexed, but the cost to build the index for a multi-valued attribute is larger in terms of storage, update, and search time. The uniqueness requirement for a multi-valued property is the same as that for a single-valued property—the more unique the values are the more effective the index.

The more indexed attributes a class has, the more time is required to create new instances of the class.

Indexes apply to attributes, not to classes. That is, when an attribute is marked as indexed, all instances of the attribute are added to the index, not just the instances that are members of a particular class.

To verify that a server is using an index to process a query, set the following registry value on a domain controller to 4. Then perform a query on that domain controller and look in the directory event log for data about the indexes, if any, used to process the query.

```
HKEY_LOCAL_MACHINE  
  SYSTEM  
    Current Control Set  
      Services  
        NTDS  
          Diagnostics  
            9 Internal Processing
```

For more information about other bits in the **searchFlags** property, see [Characteristics of Attributes](#).

Including Attributes in the Global Catalog

6/3/2022 • 2 minutes to read • [Edit Online](#)

The global catalog of a forest includes a partial replica of every object in the forest. For each object, the global catalog includes only a subset of each object's attributes. The **isMemberOfPartialAttributeSet** attribute of an **attributeSchema** object is set to **TRUE** if the attribute is replicated to the global catalog.

Attributes with the following characteristics are appropriate for storage in the global catalog:

- The attribute is globally interesting, either because the attribute is required for locating objects that can occur anywhere in the forest, or because read access to the attribute is valuable even when the full object is not accessible. An example of the first type is the **location** attribute, which can be used to find a **printQueue** object. An example of the second type is **telephoneNumber**, because you can call someone even if you cannot access a full replica of their **user** object.
- The volatility of the attribute is very low. This is important, because if an attribute class is included in the global catalog, changes to every value of that attribute class throughout the enterprise forest are replicated to all global catalog servers in the enterprise.
- The size of the attribute value is small. "Small" is highly subjective: when placing an attribute in the global catalog, consider the impact of replicating the attribute to all global catalog servers in the enterprise. The smaller the attribute, the lower the impact. Because replication occurs only when the attribute changes, the impact of replication is also smaller as volatility decreases, so a large attribute with very low volatility may have a smaller overall impact than a small attribute with high volatility.

When deciding whether or not to place an attribute in the global catalog remember that you are trading increased replication and increased disk storage on global catalog servers for, potentially, faster query performance. Typically, you would use the global catalog to search for an object of interest so you can read selected attributes of the object. If the attributes you are interested in are replicated to the global catalog, you can read them directly from the global catalog. Alternately, to read attributes that are not replicated to the global catalog, you must perform additional steps to retrieve them. In this case, after searching the global catalog to find the object of interest, you must read the object's distinguished name from the global catalog, use the DN to bind directly to a full replica of the object, which may be on a different server, and finally read the non-global-catalog attributes from the full replica of the object.

Frequently queried and referenced attributes, such as employee name and phone number, are good to include in the global catalog. An infrequently referenced attribute such as "driverVersion" for printers is best left out of the global catalog.

Linked Attributes (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

Linked attributes are pairs of attributes in which the system calculates the values of one attribute (the back link) based on the values set on the other attribute (the forward link) throughout the forest. A back-link value on any object instance consists of the DNs of all the objects that have the object's DN set in the corresponding forward link. For example, "Manager" and "Reports" are a pair of linked attributes, where Manager is the forward link and Reports is the back link. Now suppose Bill is Joe's manager. If you store the DN of Bill's user object in the "Manager" attribute of Joe's user object, then the DN of Joe's user object will show up in the "Reports" attribute of Bill's user object.

A forward link/back link pair is identified by the **linkID** values of two **attributeSchema** definitions. The **linkID** of the forward link is an even, positive, nonzero value, and the **linkID** of the associated back link is the forward **linkID** plus one. For example, the **linkID** for "Manager" is 42 and the **linkID** for "Reports" is 43.

The following is a list of guidelines for defining a new pair of linked attributes:

- The **linkID** values must be unique among all **attributeSchema** objects. To avoid conflicts, you should auto-generate the **linkID** by following the instructions in the topic [Obtaining a Link ID](#).
- A back link must have a corresponding forward link, that is, the forward link must exist before a corresponding back link attribute can be created.
- A back link is always a multi-valued attribute. A forward link can be single-valued or multi-valued. Use a multi-valued forward link when there is a many-to-many relationship.
- The **attributeSchema** value of a forward link must be 2.5.5.1, 2.5.5.7, or 2.5.5.14. These values correspond to syntaxes that contain a distinguished name, such as the [Object\(DS-DN\)](#) syntax.
- The **attributeSchema** value of a back link must be 2.5.5.1, which is the [Object\(DS-DN\)](#) syntax.
- By convention, back link attributes are added to the **mayContain** value of the **top** abstract class. This enables the back link attribute to be read from objects of any class because they are not actually stored with the object, but are calculated based on the forward link values.

The Abstract Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

The schema container contains all of the **classSchema** and **attributeSchema** objects that define the classes and attributes that can exist in a directory forest. The schema container also contains an object named **Aggregate of class subSchema**. This **subSchema** object is known as the abstract schema.

The abstract schema contains a subset of the data stored in the **classSchema** and **attributeSchema** objects. Its purpose is to provide a simple and efficient mechanism for retrieving the frequently used elements of the class and attribute definitions. For example, to retrieve the optional and mandatory attributes of an object class, bind to multiple objects to collect the **mayContain**, **mustContain**, **systemMayContain**, and **systemMustContain** values from the class and all its superclasses, as well as from any auxiliary classes of the class and its superclasses. The abstract schema conveniently collects all this data in a single object.

As with any object in Active Directory Domain Services, you can bind to the **subSchema** object and read its attributes, parsing the string values to retrieve the desired data. However, ADSI provides a set of interfaces that make it much easier to read the abstract schema. For more information, see [Reading the Abstract Schema](#).

The following table lists key attributes of a **subSchema** object.

ATTRIBUTE	DESCRIPTION
attributeTypes	A multi-valued attribute that contains strings that represent each attribute in the schema. Each value contains the attributeID , IDAPDisplayName , attributeSyntax , rangeLower , rangeUpper , and an item that indicates whether the attribute can have multiple values.
extendedAttributeInfo	A multi-valued attribute that contains strings that represent additional data for each attribute. Each value contains the attributeID , IDAPDisplayName , schemaIDGUID , and attributeSecurityGUID .
extendedClassInfo	A multi-valued attribute that contains strings that represent additional data for each class. Each value contains the governsID , IDAPDisplayName , and schemaIDGUID of the class.
objectClasses	A multi-valued attribute that contains strings that represent each class in the schema. Each value contains the governsID , IDAPDisplayName , mustContain , mayContain , and so on.

Using Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

This section provides guidelines for writing applications that use or publish data in an Active Directory directory service.

NOTE

The following documentation is for computer programmers. If you are trying to resolve an Active Directory home printing error, see the [following suggestion](#) from the Microsoft community pages; if that doesn't help, try these recommendations from [TechNet](#).

Active Directory Domain Services are compliant with Lightweight Directory Access Protocol 3.0, which is defined by RFC 2251 and other RFCs. Any of the following API sets can be used to access Active Directory Domain Services. Each API set has advantages and disadvantages that depend on the programming language, programming environment, and intended method of execution. The majority of the examples in this guide use ADSI, which is supported by languages such as C and C++, as well as automation-compliant languages such as Microsoft Visual Basic and Visual Basic Scripting Edition.

For more information about specific Active Directory Domain Services technologies, see:

- [Lightweight Directory Access Protocol](#)
- [Active Directory Service Interfaces](#)
- [System.DirectoryServices](#)

This section discusses the following topics:

- [Binding to Active Directory Domain Services](#)
- [Searching in Active Directory Domain Services](#)
- [Creating and Deleting Objects in Active Directory Domain Services](#)
- [Moving Objects](#)
- [Reading and Writing Attributes of Objects in Active Directory Domain Services](#)
- [Controlling Access to Objects in Active Directory Domain Services](#)
- [Extending the Schema](#)
- [Extending the User Interface for Directory Objects](#)
- [Managing Users](#)
- [Managing Groups](#)
- [Tracking Changes](#)
- [Service Publication](#)
- [Service Logon Accounts](#)
- [Mutual Authentication Using Kerberos](#)
- [Backing Up and Restoring Active Directory](#)
- [Storing Dynamic Data](#)
- [Application Directory Partitions](#)
- [Detecting the Operation Mode of a Domain](#)

Binding to Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Active Directory Domain Services, the act of associating a programmatic object with a specific Active Directory Domain Services object is known as *binding*. When a programmatic object, such as an [IADs](#) or [DirectoryEntry](#) object, is associated with a specific directory object, the programmatic object is considered to be *bound to* the directory object.

Binding Functions and Methods

The method for programmatically binding to an Active Directory object will depend on the programming technology that is used. For more information about binding to objects in Active Directory Domain Services with a specific programming technology, see the topics listed in the following table.

PROGRAMMING TECHNOLOGY	FOR MORE INFORMATION
Active Directory Service Interfaces	Binding to an ADSI Object
Lightweight Directory Access Protocol	Establishing an LDAP Session
System.DirectoryServices	Binding to Directory Objects

Binding Strings

All bind functions and methods require a binding string. The form of the binding string depends on the provider. Active Directory Domain Services are supported by two providers, LDAP and WinNT.

Beginning with Windows 2000, the LDAP provider is used to access Active Directory Domain Services. The LDAP binding string can take one of the following forms:

- "LDAP://<host name>/<object name>"
- "GC://<host name>/<object name>"

In the examples above, "LDAP:" specifies the LDAP provider. "GC:" uses the LDAP provider to bind to the Global Catalog service to execute fast queries.

"<host name>" specifies the server to bind to and is optional. If possible, do not specify a server. It is also possible to bind to an object in a different domain. To do this pass the domain naming system (DNS) name of the target domain for "<host name>". For example, to bind to the Users container in the domain2 domain of fabrikam.com, the binding string would be

"LDAP://domain2.fabrikam.com/CN=Users,DC=domain2,DC=fabrikam,DC=com".

"<object name>" represents a specific object in Active Directory Domain Services. The object name can be a distinguished name or an object GUID.

For more information about LDAP binding strings, see [LDAP ADsPath](#).

For Windows NT 4.0, the WinNT provider is used for access to directory data such as users, user groups, computers, services, and other network objects in the Windows 2000. The WinNT provider on Windows 2000 and later systems has limited functionality compared to the LDAP provider. For more information about WinNT

binding strings, see [WinNT ADsPath](#).

An ADsPath of "LDAP://" or "GC://" can be used to bind to the root of the namespace. When bound to the root of the namespace, the supplied namespace object contains no properties and contains the domain object for LDAP and a container object containing a partial replica of all domains in the forest for GC.

For more information about binding in Active Directory Domain Services, see:

- [Serverless Binding and RootDSE](#)
- [Binding to the Global Catalog](#)
- [Using objectGUID to Bind to an Object](#)
- [Binding to Well-Known Objects Using WKGUID](#)
- [Binding to an Object Using a SID](#)
- [Enabling Rename-Safe Binding with the otherWellKnownObjects Property](#)
- [Authentication](#)

Serverless Binding and RootDSE

6/3/2022 • 2 minutes to read • [Edit Online](#)

If possible, do not hard-code a server name. Furthermore, under most circumstances, binding should not be unnecessarily tied to a single server. Active Directory Domain Services support serverless binding, which means that Active Directory can be bound to on the default domain without specifying the name of a domain controller. For ordinary applications, this is typically the domain of the logged-on user. For service applications, this is either the domain of the service logon account or that of the client that the service impersonates.

In LDAP 3.0, rootDSE is defined as the root of the directory data tree on a directory server. The rootDSE is not part of any namespace. The purpose of the rootDSE is to provide data about the directory server. The following is the binding string that is used to bind to rootDSE.

```
LDAP://<servername>/rootDSE
```

The <servername> is the DNS name of a server. The <servername> is optional, as shown in the following format.

```
LDAP://rootDSE
```

In this case, a default domain controller from the domain that the security context of the calling thread is in will be used. If a domain controller cannot be accessed within the site, the first domain controller that can be found will be used.

The rootDSE is a well-known and reliable location on every directory server to get distinguished names of the domain, schema, and configuration containers, and other data about the server and the contents of its directory data tree. These properties rarely change on a particular server. An application can read these properties at startup and use them throughout the session.

In summary, an application should use serverless binding to bind to the directory on the current domain, use rootDSE to get the distinguished name for a namespace, and use that distinguished name to bind to objects in the namespace.

For more information about attributes supported by rootDSE, see [RootDSE](#) in the Active Directory Schema documentation.

For more information and a code example that shows how to use serverless binding and rootDSE, see [Example Code for Getting the Distinguished Name of the Domain](#).

Example Code for Getting the Distinguished Name of the Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes a code example that gets the distinguished name of the domain that the local computer is a member of by using serverless binding.

The following Visual Basic code example gets the distinguished name of the domain that the local computer is a member of by using serverless binding.

```
Dim rootDSE As IADs
Dim DistinguishedName As String

Set rootDSE = GetObject("LDAP://rootDSE")
DistinguishedName = "LDAP://" & rootDSE.Get("defaultNamingContext")
```

The following C# code example gets the distinguished name of the domain that the local computer is a member of by using serverless binding.

```
DirectoryEntry RootDirEntry = new DirectoryEntry("LDAP://RootDSE");
Object distinguishedName = RootDirEntry.Properties["defaultNamingContext"].Value;
```

The following C/C++ code example gets the distinguished name of the domain that the local computer is a member of by using serverless binding.

```
IADs *pads;
hr = ADsGetObject( L"LDAP://rootDSE",
                    IID_IADs,
                    (void**)&pads);

if(SUCCEEDED(hr))
{
    VARIANT var;

    VariantInit(&var);

    hr = pads->Get(CComBSTR("defaultNamingContext"), &var);
    if(SUCCEEDED(hr))
    {
        if(VT_BSTR == var.vt)
        {
            wprintf(var.bstrVal);
        }

        VariantClear(&var);
    }

    pads->Release();
}
```

Binding to the Global Catalog

6/3/2022 • 4 minutes to read • [Edit Online](#)

The Global Catalog is a namespace that contains directory data for all domains in a forest. The Global Catalog contains a partial replica of every domain directory. It contains an entry for every object in the enterprise forest, but does not contain all the properties of each object. Instead, it contains only the properties specified for inclusion in the Global Catalog.

The Global Catalog is stored on specific servers throughout the enterprise. Only domain controllers can serve as Global Catalog servers. Administrators indicate whether a given domain controller holds a Global Catalog by using the Active Directory Sites and Services Manager.

To bind to the Global Catalog with ADSI, use the "GC:" moniker.

There are two ways to bind to the Global Catalog to perform a search in a forest:

- Bind to the enterprise root object to search across all domains in the forest.
- Bind to a specific object to search that object and its children. For example, if you bind to a domain that has two domains beneath it in a domain tree in the forest, you can search across those three domains. Be aware that the distinguished name for the object to bind to is exactly the same as the distinguished name used to bind to the "LDAP:" namespace. Recall that "LDAP:" is a full replica of a single domain and that "GC:" is a partial replica of all domains in the forest.

As with the "LDAP:" moniker, you can use serverless binding or bind to a specific Global Catalog server. If searching in the current forest, use serverless binding. However, if searching in another forest, specify either a domain name or a Global Catalog server to bind to, such as shown in the following examples.

Bind using the domain name:

```
GC://fabrikam.com
```

Bind using the server name:

```
GC://servername
```

You can also bind to a specific object within the Global Catalog. To bind to the sales object in the fabrikam domain, use the following format.

```
GC://fabrikam.com/DC=sales,DC=fabrikam,DC=com
```

Or, to bind to the sales object on the server, use the following format.

```
GC://servername.fabrikam.com/DC=sales,DC=fabrikam,DC=com
```

To search the entire forest

- Bind to the root of the Global Catalog namespace.
- Enumerate the Global Catalog container. The Global Catalog container contains a single object that you can use to search the entire forest.

3. Use the object in the container to perform the search. In C/C++, call [QueryInterface](#) to get an [IDirectorySearch](#) pointer on the object so that you can use the [IDirectorySearch](#) interface to perform the search. In Visual Basic, use the object returned from the enumeration in your ADO query.

To enumerate the Global Catalog servers in a site, perform an LDAP subtree search of "cn=<yoursite>,cn=sites,<DN of the configurationNamingContext>", using the following filter string.

```
(&(objectCategory=nTDSSDA)(options:1.2.840.113556.1.4.803:=1))
```

This filter uses the [LDAP_MATCHING_RULE_BIT_AND](#) matching rule operator (1.2.840.113556.1.4.803) to find [nTDSSDA](#) objects that have the low-order bit set in the bitmask of the [options](#) attribute. The low-order bit, which corresponds to the [NTDSSDA_OPT_IS_GC](#) constant defined in [Ntdsapi.h](#), identifies the [nTDSSDA](#) object of a Global Catalog server. For more information about matching rules, see [Search Filter Syntax](#).

The parent of the [nTDSSDA](#) object is the server object, and the [dNSHostName](#) property of the server object is the DNS name of the Global Catalog server.

You cannot use #define constants such as [NTDSSDA_OPT_IS_GC](#) and [LDAP_MATCHING_RULE_BIT_AND](#) directly in a search filter string. However, you could use these constants as arguments to a function such as [swprintf_s](#) to insert the constant values into a filter string.

The global catalog does not represent the entire forest tree structure. For example, you might expect that the following code example would enumerate all of the domains in the forest and all child objects of each domain. In reality, what it actually does is enumerate all of the domains in the forest, but none of the enumerated domain objects contain any children. This is a limitation of the global catalog.

```
Dim oGC As IADs
Dim oDomain As IADs
Dim oChild As IADs

Set oGC = GetObject("GC:")
For Each oDomain In oGC
    ' Print the name of the domain.
    Debug.Print oDomain.Name

    ' Enumerate the child objects of the domain.
    For Each oChild In oDomain
        Debug.Print oChild.Name
    Next
Next
```

To correct this, it is necessary to bind to each domain object and then enumerate the child objects of each domain. The proper technique is shown in the following code example.

```
Dim oGC As IADs
Dim oDomainEnum As IADs
Dim oDomainBind As IADs
Dim oChild As IADs

Set oGC = GetObject("GC:")
For Each oDomainEnum In oGC
    ' Print the name of the domain.
    Debug.Print oDomainEnum.Name

    ' Bind to the domain.
    Set oDomainBind = GetObject("LDAP://" + oDomainEnum.Name)

    ' Enumerate the child objects of the domain.
    For Each oChild In oDomainBind
        Debug.Print oChild.Name
    Next
Next
```

For more information and code examples that show how to search an entire forest, see [Example Code for Searching a Forest](#).

Example Code for Searching a Forest

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic contains example code that searches a forest.

The following C/C++ code example binds to the root of the Global Catalog and enumerates the single object, which is the root of the forest, so that it can be used to search the entire forest.

```
Set gc = GetObject("GC:")
For each child in gc
    Set entpr = child
Next
' Now entpr is an object that can be used
' to search the entire forest.
```

The following C/C++ code example contains a function that returns an [IDirectorySearch](#) pointer used to search the entire forest.

The function performs a serverless bind to the root of the Global Catalog, enumerates the single item, which is the root of the forest and can be used to search the entire forest, calls [QueryInterface](#) to get an [IDirectorySearch](#) pointer to the object, and returns that pointer for use by the caller to search the forest.

```

HRESULT GetGC(IDirectorySearch **ppDS)
{
    HRESULT hr;
    IEnumVARIANT *pEnum = NULL;
    IADsContainer *pCont = NULL;
    VARIANT var;
    IDispatch *pDisp = NULL;
    ULONG lFetch;

    // Set IDirectorySearch pointer to NULL.
    *ppDS = NULL;

    // First, bind to the GC: namespace container object.
    hr = ADsOpenObject(TEXT("GC:"),  

                       NULL,  

                       NULL,  

                       ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.  

                       IID_IADsContainer,  

                       (void**)&pCont);
    if (FAILED(hr)) {
        _tprintf(TEXT("ADsOpenObject failed: 0x%x\n"), hr);
        goto cleanup;
    }

    // Get an enumeration interface for the GC container to enumerate the  

    // contents. The actual GC is the only child of the GC container.
    hr = ADsBuildEnumerator(pCont, &pEnum);
    if (FAILED(hr)) {
        _tprintf(TEXT("ADsBuildEnumerator failed: 0x%x\n"), hr);
        goto cleanup;
    }

    // Now enumerate. There is only one child of the GC: object.
    hr = pEnum->Next(1, &var, &lFetch);
    if (FAILED(hr)) {
        _tprintf(TEXT("ADsEnumerateNext failed: 0x%x\n"), hr);
        goto cleanup;
    }

    // Get the IDirectorySearch pointer.
    if ((hr == S_OK) && (lFetch == 1))
    {
        pDisp = V_DISPATCH(&var);
        hr = pDisp->QueryInterface( IID_IDirectorySearch, (void**)ppDS);
    }

cleanup:

    if (pEnum)
        ADsFreeEnumerator(pEnum);
    if (pCont)
        pCont->Release();
    if (pDisp)
        (pDisp)->Release();
    return hr;
}

```

Using objectGUID to Bind to an Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

An object distinguished name changes if the object is renamed or moved, therefore the distinguished name is not a reliable object identifier. In Active Directory Domain Services, an object's **objectGUID** property never changes, even if the object is renamed or moved. For more information about **objectGUID** and identifiers, see [Object Names and Identities](#).

The Active Directory LDAP provider provides a method to bind to an object using the object GUID. The binding string format is as follows:

```
LDAP://servername/<GUID=XXXXX>
```

In this example, "servername" is the name of the directory server and "XXXXX" is the string representation of the hexadecimal value of the GUID. The "servername" is optional. The GUID string is specified in the "XXXXXXXXXXXXXXXXXXXXXXXXXXXX" form. The GUID string can also take the form "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", which is the same form as the string produced by the [StringFromGUID2](#) function, without the surrounding braces "{}". For more information and a code example that shows how to create a bindable string from a GUID, see [Example Code for Creating a Bindable String Representation of a GUID](#). The [IADs.GUID](#) property can be used to retrieve the proper string form of the GUID.

When binding using the object GUID, some [IADs](#) and [IADsContainer](#) methods and properties are not supported. The following [IADs](#) properties are not supported by objects obtained by binding using the object GUID:

- [ADsPath](#)
- [Name](#)
- [Parent](#)

The following [IADsContainer](#) methods are not supported by objects obtained by binding using the object GUID:

- [GetObject](#)
- [Create](#)
- [Delete](#)
- [CopyHere](#)
- [MoveHere](#)

To use these methods and properties after binding to an object using the object GUID, use the [IADs.Get](#) method to retrieve the object distinguished name and then use the distinguished name to bind to the object again.

If an application stores or caches identifiers or references to objects stored in Active Directory Domain Services, the object GUID is the best identifier to use for several reasons:

- The **objectGUID** property of an object never changes even if the object is renamed or moved.
- It is easy to bind to the object using the object GUID.
- If the object is renamed or moved, the **objectGUID** property provides a single identifier that can be used to quickly find and identify the object rather than having to compose a query that has conditions for all properties that would identify that object.

Example Code for Creating a Bindable String Representation of a GUID

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example can be used to return a string representation of a GUID that can be used to bind to the object.

```

//*****
// GUIDtoBindableString()
//
// Converts a GUID into a string that can be used for binding with
// the <GUID= or <WKGUID= syntax. The caller must free the allocated
// string with the FreeADsStr when it is no longer required.
//
//*****

HRESULT GUIDtoBindableString(LPGUID pGUID, LPWSTR *ppGUIDString)
{
    if((!pGUID) || (ppGUIDString==NULL))
    {
        return E_INVALIDARG;
    }

    // Build bindable GUID string.
    DWORD dwBytes = sizeof(GUID);
    WCHAR szByte[3];
    LPWSTR pwszGUID = new WCHAR[(dwBytes * 2) + 1];
    if(NULL == pwszGUID)
    {
        return E_OUTOFMEMORY;
    }

    *pwszGUID = NULL;

    HRESULT hr = S_OK;
    LPBYTE lpByte;
    DWORD dwItem;

    // Loop through to add each byte to the string.
    for(dwItem = 0,
        lpByte = (LPBYTE)pGUID; dwItem < dwBytes;
        dwItem++)
    {
        // Append to pwszGUID, double-byte, byte at dwItem index.
        swprintf_s(szByte, L"%02x", lpByte[dwItem]);
        wcscat_s(pwszGUID, szByte);
    }

    // Allocate memory for the string.
    *ppGUIDString = AllocADsStr(pwszGUID);

    delete [] pwszGUID;

    if(NULL != *ppGUIDString)
    {
        hr = S_OK;
    }
    else
    {
        hr = E_OUTOFMEMORY;
    }

    // Caller must free ppGUIDString using FreeADsStr.
    return hr;
}

```

Reading an Object's objectGUID and Creating a String Representation of the GUID

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **objectGUID** property of each object in Active Directory Domain Services is stored in the directory as an octet string. An octet string is an array of one-byte characters. Use the [IADs::get_GUID](#) method to retrieve the bindable string form of a directory object's **objectGUID**.

The following code examples show a function that reads the **objectGUID** attribute and returns a string representation of the GUID used to bind to the object.

- [Visual Basic Example](#)
- [C++ Example](#)

Visual Basic Example

```

Dim sADsPathObject As String
Dim sObjectGUID As String
Dim sBindByGuidStr As String

Dim IADsObject As IADs
Dim IADsObject2 As IADs
On Error Resume Next

' Query the user for an ADsPath to start.
sADsPathObject = InputBox("This code binds to a directory object by ADsPath, retrieves the GUID, then rebinds by GUID." & vbCrLf & vbCrLf & "Specify the ADsPath of the object to bind to :")

If sADsPathObject = "" Then
    GoTo CleanUp
End If

MsgBox "Binding to " & sADsPathObject

' Bind to initial object.
Set IADsObject = GetObject(sADsPathObject)

If (Err.Number <> 0) Then
    MsgBox Err.Number & " on GetObject method"
    GoTo CleanUp
End If

' Save the GUID of the object.
sObjectGUID = IADsObject.Guid

MsgBox "The GUID for " & vbCrLf & vbCrLf & sADsPathObject & vbCrLf & vbCrLf & " is " & vbCrLf & vbCrLf & sObjectGUID

' Release the initial object.
Set IADsObject = Nothing

' Build a string for Binding to the object by GUID.
sBindByGuidStr = "LDAP://<GUID=" & sObjectGUID & ">"

MsgBox sBindByGuidStr

' Bind BACK to the Same object using the GUID.
Set IADsObject = GetObject(sBindByGuidStr)

If (Err.Number <> 0) Then
    MsgBox Err.Number & " on GetObject method "
    GoTo CleanUp
End If

MsgBox "Successfully RE bound to " & sADsPathObject & vbCrLf & vbCrLf & " using the path:" & vbCrLf & vbCrLf & sBindByGuidStr

' Release bind by GUID Object.
CleanUp:
    Set IADsObject = Nothing

```

C++ Example

```

#define UNICODE
#include <ole2.h>
#include <stdio.h>
#include <activeds.h>

#define PATH_SIZE 1024

```

```

void wmain(int argc, wchar_t *argv[])
{
    // Initialize COM.
    CoInitialize(0);

    WCHAR wszADsPathObject[PATH_SIZE + 1];
    HRESULT hr;
    IADs *pIADsObject = NULL;
    IADs *pIADsObjectByGuid = NULL;

    // If no ADsPath was specified on the command line, query the user for a ADsPath to start.
    if(!argv[1])
    {
        _putws(L"This code binds to a directory object by ADsPath, retrieves the GUID,\n"
              L" then rebinds by GUID. \n\nEnter the ADsPath of the object to bind to :\n");
        fgetws(wszADsPathObject, PATH_SIZE, stdin);
    }
    else
    {
        wcsncpy_s(wszADsPathObject, argv[1], PATH_SIZE - 1);
        wszADsPathObject[PATH_SIZE] = 0;
    }

    if (!wszADsPathObject[0])
    {
        return;
    }

    wprintf(L"\nBinding to %s\n", wszADsPathObject);

    // Bind to initial object.
    hr = ADsGetObject(wszADsPathObject, IID_IADs, (void**)&pIADsObject);
    if (SUCCEEDED(hr))
    {
        BSTR bstrGuid = NULL;
        hr = pIADsObject->get_GUID(&bstrGuid);

        if (SUCCEEDED(hr))
        {
            LPWSTR wszFormat = L"LDAP://<GUID=%s>";
            LPWSTR pwszBindByGuidStr;

            wprintf(L"\n The GUID for\n%s\nnis\n%s\n", wszADsPathObject, bstrGuid);

            // Build a string for Binding to the object by GUID.
            pwszBindByGuidStr = new WCHAR[wcslen(wszFormat) + wcslen(bstrGuid) + 1];
            if(pwszBindByGuidStr)
            {
                swprintf_s(pwszBindByGuidStr, wszFormat, bstrGuid);

                // Bind BACK to the Same object using the GUID.
                hr = ADsGetObject(pwszBindByGuidStr, IID_IADs, (void**)&pIADsObjectByGuid);

                if (SUCCEEDED(hr))
                {
                    wprintf(L"\nSuccessfully re-bound to\n%s\nUsing the path:\n%s\n",
                           wszADsPathObject, pwszBindByGuidStr);

                    // Release bind by GUID Object.
                    pIADsObjectByGuid->Release();
                    pIADsObjectByGuid = NULL;
                }
            }

            delete pwszBindByGuidStr;
        }
        else
        {
            hr = E_OUTOFMEMORY;
        }
    }
}

```

```
    SysFreeString(bstrGuid);
}

pIADsObject->Release();
pIADsObject = NULL;
}

if (FAILED(hr))
{
    wprintf(L"Failed");
}
}
```

Binding to Well-Known Objects Using WKGUID

6/3/2022 • 2 minutes to read • [Edit Online](#)

Containers can have one, or more, important subobjects that can be renamed or moved. It can be difficult to track these subobjects and build correct binding strings for them, especially if they are renamed or moved.

To support rename-safe binding to these objects within these containers, Active Directory Domain Services have two attributes: **wellKnownObjects** and **otherWellKnownObjects**. These attributes have an attribute syntax of **ADSTYPE_DN_WITH_BINARY**. They allow multiple values and contain the GUID/DN tuples of well-known objects within the containers on which they are set. The Active Directory server maintains the distinguished name portion of each **wellKnownObjects** and **otherWellKnownObjects** entry so that it contains the current distinguished name of the object specified when the entry was created.

The **otherWellKnownObjects** property can be set and used on any object.

The **wellKnownObjects** property is used on the domainDNS and configuration containers. The **wellKnownObjects** property is a system-only property and can only be modified by the operating system.

The **domainDNS** container has the following well-known objects:

- Users
- Computers
- System
- Domain Controllers
- Infrastructure
- Deleted Objects
- Lost and Found

The configuration container has the well-known object: Deleted Objects.

These objects are in every **domainDNS** and configuration container in an Active Directory Domain Service.

You can bind to a well-known object using the WKGUID binding format. Binding with WKGUID is only supported in Active Directory Domain Services, that is, the LDAP provider.

IMPORTANT

Always use the WKGUID binding format to bind to the well-known objects, as listed above, in the domain and configuration containers. This ensures that these containers can be bound to even if they are moved or renamed.

The WKGUID binding string format is as follows:

```
LDAP://<servername>/<WKGUID=<XXXXXX>,<container DN>>
```

"<server name>" is the directory server name. The "<server name>" is optional.

"<XXXXXX>" is the string representation of the hexadecimal value of the GUID that represents the well-known object. The GUID string is specified in the "XXXXXXXXXXXXXXXXXXXXXX" form and is not the same form as the GUID string produced by the **StringFromGUID2** function, which takes the form "{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX}". For more information and a code example that shows how to create a bindable

string from a GUID, see [Example Code for Creating a Bindable String Representation of a GUID](#).

To bind to an object specified in **otherWellKnownObjects** in an object, specify "<XXXXX>" as the bind string form of the well-known object GUID of the object. For more information, see [Reading an Object's objectGUID and Creating a String Representation of the GUID](#). For more information about setting the **otherWellKnownObjects** property on objects, see [Enabling Rename-Safe Binding with the otherWellKnownObjects Property](#).

To bind to an object specified in **wellKnownObjects** in domainDNS or configuration containers, specify "<XXXXX>" as one of the following constants, listed in the following table, as defined in Ntdsapi.h.

CONTAINER	GUID IDENTIFIER
Users	GUID_USERS_CONTAINER_W
Computers	GUID_COMPUTRS_CONTAINER_W
System	GUID_SYSTEMS_CONTAINER_W
Domain Controllers	GUID_DOMAIN_CONTROLLERS_CONTAINER_W
Infrastructure	GUID_INFRASTRUCTURE_CONTAINER_W
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
Lost and Found	GUID_LOSTANDFOUND_CONTAINER_W

For example, to bind to the user container in a domain, specify **GUID_USERS_CONTAINER_W** as "<XXXXX>".

"<container DN>" is the distinguished name of the container object that has this object represented as a value in its **wellKnownObjects** property. For example, to bind to the users container in a domain, specify the domain distinguished name as the "<container DN>".

For example, to bind to the users container of the domain Fabrikam.com, use the following binding string.

```
LDAP://<WKGUID=a9d1ca15768811d1aded00c04fd8d5cd,dc=Fabrikam,dc=com>
```

For more information and a code example that shows how to bind to a well-known GUID, see [Example Code for Binding to Well Known Objects](#).

Example Code for Binding to Well Known Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ code example shows how to bind to a well-known object using the WKGUID binding string.

```
*****  
//  
//  BindToWellKnownObject()  
//  
//  Binds to one of the well-known objects in the current domain  
//  with the current user credentials. pwszGUID must be one of  
//  the GUID strings defined in NTDSAPI.H, such as  
//  GUID_USERS_CONTAINER_W.  
//  
*****  
  
HRESULT BindToWellKnownObject(LPCWSTR pwszGUID, IADs **ppObject)  
{  
    if(NULL == ppObject)  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr;  
    IADs *pRoot;  
  
    *ppObject = NULL;  
  
    // Bind to the rootDSE object.  
    hr = ADsOpenObject(L"LDAP://rootDSE",  
                      NULL,  
                      NULL,  
                      ADS_SECURE_AUTHENTICATION,  
                      IID_IADs,  
                      (LPVOID*)&pRoot);  
    if(SUCCEEDED(hr))  
    {  
        VARIANT var;  
  
        VariantInit(&var);  
  
        // Get the current domain DN.  
        hr = pRoot->Get(CComBSTR("defaultNamingContext"), &var);  
        if(SUCCEEDED(hr))  
        {  
            // Build the binding string.  
            LPWSTR pwszFormat = L"LDAP://<WKGUID=%s,%s>";  
            LPWSTR pwszPath;  
  
            pwszPath = new WCHAR[lstrlenW(pwszFormat) +  
                            lstrlenW(pwszGUID) +  
                            lstrlenW(var.bstrVal)];  
            if(NULL != pwszPath)  
            {  
                swprintf_s(pwszPath, pwszFormat, pwszGUID, var.bstrVal);  
  
                // Bind to the object.  
                hr = ADsOpenObject(pwszPath,  
                                  NULL,  
                                  NULL,  
                                  ADS_SECURE_AUTHENTICATION,  
                                  IID_IADs,
```

```
        (LPVOID*)ppObject);  
  
    delete pwszPath;  
}  
else  
{  
    hr = E_OUTOFMEMORY;  
}  
  
VariantClear(&var);  
}  
  
pRoot->Release();  
}  
  
return hr;  
}
```

Binding to an Object Using a SID

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Windows Server 2003, it is possible to bind to an object using the object Security Identifier (SID) as well as a GUID. The object SID is stored in the **objectSID** attribute. Binding to an SID does not work on Windows 2000.

The LDAP provider for Active Directory Domain Services provides a method to bind to an object using the object SID. The binding string format is:

```
LDAP://servername/<SID=XXXXXX>
```

In this example, "servername" is the name of the directory server and "XXXXXX" is the string representation of the hexadecimal value of the SID. The "servername" is optional. The SID string is specified in a form where each character in the string is the hexadecimal representation of each byte of the SID. For example, if the array is:

```
0xAB 0x14 0xE2
```

the SID binding string would be "<SID=AB14E2>". The [ADsEncodeBinaryData](#) function should not be used to convert the SID array into a string because it precedes each byte character with a backslash, which is not a valid bind string format.

The SID string can also take the form "<SID=S-X-X-XX-XXXXXXXXXX-XXXXXXXXXX-XXXXXXXXXX-XXX>", where the "S-X-X-XX-XXXXXXXXXX-XXXXXXXXXX-XXXXXXXXXX-XXX" portion is the same as the string returned by the [ConvertSidToStringSid](#) function.

When binding using the object SID, some [IADs](#) and [IADsContainer](#) methods and properties are not supported. The following [IADs](#) properties are not supported by objects obtained by binding using the object SID:

- [ADsPath](#)
- [Name](#)
- [Parent](#)

The following [IADsContainer](#) methods are not supported by objects obtained by binding using the object SID:

- [GetObject](#)
- [Create](#)
- [Delete](#)
- [CopyHere](#)
- [MoveHere](#)

To use these methods and properties after binding to an object using the object SID, use the [IADs.Get](#) method to retrieve the object distinguished name and then use the distinguished name to bind to the object again.

The following code example shows how to convert an **objectSid** into a bindable string.

```
HRESULT VariantArrayToBytes(VARIANT Variant,
    LPBYTE *ppBytes,
    DWORD *pdwBytes);

*****  
GetSIDBindStringFromVariant()
```

Converts a SID in VARIANT form, such as an objectSid value, and converts it into a bindable string in the form:

```
LDAP://<SID=xxxxxxxx...>
```

The returned string is allocated with AllocADsMem and must be freed by the caller with FreeADsMem.

```
******/
```

```
LPWSTR GetSIDBindStringFromVariant(VARIANT vSID)
{
    LPWSTR pwszReturn = NULL;

    if(VT_ARRAY & vSID.vt)
    {
        HRESULT hr;
        LPBYTE pByte;
        DWORD dwBytes = 0;

        hr = VariantArrayToBytes(vSID, &pByte, &dwBytes);
        if(S_OK == hr)
        {
            // Convert the BYTE array into a string of hex
            // characters.
            CComBSTR sbstrTemp = "LDAP://<SID=";

            for(DWORD i = 0; i < dwBytes; i++)
            {
                WCHAR wszByte[3];

                swprintf_s(wszByte, L"%02x", pByte[i]);
                sbstrTemp += wszByte;
            }

            sbstrTemp += ">";
            pwszReturn =
                (LPWSTR)AllocADsMem((sbstrTemp.Length() + 1) *
                    sizeof(WCHAR));
            if(pwszReturn)
            {
                wcscpy_s(pwszReturn, sbstrTemp.m_str);
            }
        }

        FreeADsMem(pByte);
    }
}

return pwszReturn;
}
```

```
*****
```

```
VariantArrayToBytes()
```

This function converts a VARIANT array into an array of BYTES. This function allocates the buffer using AllocADsMem. The caller must free this memory with FreeADsMem when it is no longer required.

```
******/
```

```
HRESULT VariantArrayToBytes(VARIANT Variant,
    LPBYTE *ppBytes,
    DWORD *pdwBytes)
{
    if(!(Variant.vt & VT_ARRAY) ||
        !Variant.parray ||
```

```
    !ppBytes ||  
    !pdwBytes)  
{  
    return E_INVALIDARG;  
}  
  
*ppBytes = NULL;  
*pdwBytes = 0;  
  
HRESULT hr = E_FAIL;  
SAFEARRAY *pArrayVal = NULL;  
CHAR HUGEPEP *pArray = NULL;  
  
// Retrieve the safe array.  
pArrayVal = Variant.parray;  
DWORD dwBytes = pArrayVal->rgsabound[0].cElements;  
*ppBytes = (LPBYTE)AllocADsMem(dwBytes);  
if(NULL == *ppBytes)  
{  
    return E_OUTOFMEMORY;  
}  
  
hr = SafeArrayAccessData(pArrayVal, (void HUGEPEP * FAR *) &pArray);  
if(SUCCEEDED(hr))  
{  
    // Copy the bytes to the safe array.  
    CopyMemory(*ppBytes, pArray, dwBytes);  
    SafeArrayUnaccessData( pArrayVal );  
    *pdwBytes = dwBytes;  
}  
  
return hr;  
}
```

Enabling Rename-Safe Binding with the otherWellKnownObjects Property

6/3/2022 • 2 minutes to read • [Edit Online](#)

Objects of the Container class have an **otherWellKnownObjects** attribute that you can use to associate a GUID with the distinguished name (DN) of a child object in the container. If the child object is moved or renamed, the Active Directory server updates the DN in the **otherWellKnownObjects** value for that child object. This enables use of the WKGUID binding feature to bind to the child object using the GUID and the DN of the container rather than the child object's DN.

The **otherWellKnownObjects** attribute is equivalent to the **wellKnownObjects** attribute except that applications and services can write an **otherWellKnownObjects** value, but only the system can write **wellKnownObjects**.

Using **otherWellKnownObjects** attribute and WKGUID binding is beneficial in the following situations where rename-safe binding is required in relation to a specific container object.

If a container object contains other important objects, or if important objects can be renamed or moved.

If the important objects exist for each instance of the container object. For example, the system uses the **wellKnownObjects** attribute of each **domainDNS** object to store a value for the Users container, which exists in every instance of a **domainDNS** object. This enables applications to bind to the Users container in a rename-safe way by specifying the well-known GUID and the DN of the **domainDNS** container. Applications can use a container's **otherWellKnownObjects** attribute similarly.

If you require rename-safe binding and/or search capability on the important objects.

To add rename-safe binding and search capabilities

1. Add a value to the **otherWellKnownObjects** property of the container object when the important object is created within that container. The value contains the GUID that represents the well-known object. Be aware that this is not the **objectGUID** and the **distinguishedName** for that object.
2. Use the WKGUID binding feature to bind to or search the important object.

The **otherWellKnownObjects** attribute can have multiple values and contains the GUID/DN tuples of well-known objects within the containers on which they are set. The **otherWellKnownObjects** attribute has the **DNWithBinary** syntax in which values have the following form:

```
B:<char count>:<well known GUID>:<object DN>
```

In this example, "<char count>" is the count of hexadecimal digits in "<well known GUID>", which is 32 (number of hex digits in a GUID) for both **otherWellKnownObjects** and **wellKnownObjects**. "<well known GUID>" is the hexadecimal digit representation of the well-known GUID. "<object DN>" is the distinguished name of the object represented by this WKO value. The server maintains the ObjectDN portion of each **wellKnownObjects** and **otherWellKnownObjects** value so that it contains the current distinguished name of the object originally specified when the value was created.

For example, if {df447b5e-aa5b-11d2-8d53-00c04f79ab81} is the well-known GUID of the MyObject object in the MyContainer container in the Fabrikam.com domain, the **otherWellKnownObjects** value would specify the well-known GUID and the DN of MyObject:

```
B:32:df447b5eaa5b11d28d5300c04f79ab81:cn=MyObject,cn=MyContainer,dc=Fabrikam,dc=com
```

To bind to this object, use the following WKGUID binding string that specifies the well-known GUID of the object and the DN of the container:

```
LDAP://<WKGUID=df447b5eaa5b11d28d5300c04f79ab81,cn=MyContainer,dc=Fabrikam,dc=com>
```

After binding to this object, you can use the ADSI COM interfaces to search, read, modify, or delete the object.

For more information and a code example that shows how to add an object to the **otherWellKnownObjects** attribute, see [Example Code for Creating a Container Object](#).

Example Code for Creating a Container Object

6/3/2022 • 4 minutes to read • [Edit Online](#)

The following code example creates two objects. First, a container object and, second, a sub-container object. A value for the sub-container object is added to the `otherWellKnownObjects` property of the container object. The code example binds to the sub-container object using the WKGUID binding and displays its ADsPath. It then renames the sub-container object and binds again using the same WKGUID binding.

```
#define _WIN32_WINNT 0x0501

#include "resource.h"
#include <atlbase.h>
#include <atlcom.h>
#pragma comment(lib, "Activeds.lib")
#pragma comment(lib, "adsiid.Lib")
#include <atlenc.h>
#include <activeds.h>
#include <stdio.h>

void wmain( int argc, wchar_t *argv[ ] )
{
    HRESULT hr = S_OK;
    CComPtr<IADs> pObject ;
    CComPtr<IADsContainer> pDomain ;
    IDispatch *pDisp = NULL;

    CComPtr<IADsContainer> pNewContainer ;
    CComPtr<IADs> pIADsObject, pNewObject, pTestWKO1, pTestWKO2;

    CComVariant vartest, var;
    CComBSTR bstr, szNewContainerDN (MAX_PATH),
               szPath(MAX_PATH),
               szRelPath(MAX_PATH);
    // Names of the container and child object.
    CComBSTR szContainer ( L"MyWKOTestContainer"),
               szNewObject ( L"MyWKOTestObject"),
               szNewObjectRenameRDN ( L"cn=ObjectwithNEWNAME");

    // Get rootDSE and the domain container DN.
    hr = ADsGetObject(L"LDAP://rootDSE", IID_IADs, (void**)&pObject);
    if (FAILED(hr))
    {
        wprintf(L"Not Found. Cannot bind to the domain.\n");
        return hr;
    }

    hr = pObject->Get(CComBSTR(L"defaultNamingContext"),&var);
    if (SUCCEEDED(hr))
    {
        // Build the ADsPath to the domain.
        szPath = L"LDAP://";
        szPath.AppendBSTR(var.bstrVal);
        var.Clear();
        // Bind to the current domain.
        hr = ADsGetObject(szPath, IID_IADsContainer, (void**)&pDomain);
        if (SUCCEEDED(hr))
        {
            // Create the container.
            szRelPath = L"cn=";
            szRelPath += szContainer;
            hr = pDomain->Create(CComBSTR(L"container"),
                                  szRelPath,
                                  L"objectCategory",
                                  L"cn=ObjectwithNEWNAME");
            if (SUCCEEDED(hr))
            {
                // Set the otherWellKnownObjects property.
                hr = pObject->Set(CComBSTR(L"otherWellKnownObjects"),
                                    CComVariant(pTestWKO1));
                if (SUCCEEDED(hr))
                {
                    // Bind to the sub-container object using the WKGUID binding.
                    hr = ADsGetObject(L"LDAP://" + szRelPath + L"/" + szContainer,
                                      IID_IADs,
                                      (void**)&pIADsObject);
                    if (SUCCEEDED(hr))
                    {
                        // Display the ADsPath of the sub-container object.
                        wprintf(L"ADsPath: %s\n", pIADsObject->GetAdspath());
                        // Rename the sub-container object.
                        hr = pIADsObject->Rename(CComBSTR(L"cn=ObjectwithNEWNAME"),
                                                   L"cn=ObjectwithNEWNAME");
                        if (SUCCEEDED(hr))
                        {
                            // Bind to the sub-container object again using the WKGUID binding.
                            hr = ADsGetObject(L"LDAP://" + szRelPath + L"/" + szContainer,
                                              IID_IADs,
                                              (void**)&pIADsObject);
                            if (SUCCEEDED(hr))
                            {
                                // Display the ADsPath of the sub-container object.
                                wprintf(L"ADsPath: %s\n", pIADsObject->GetAdspath());
                            }
                        }
                    }
                }
            }
        }
    }
}
```



```

        hr = ADsGetObject(szPath, IID_IADs, (void**)&pTestWKO1);
        if (SUCCEEDED(hr))
        {
            hr = pTestWKO1->Get(CComBSTR(L"distinguishedName"),
                                   &vartest);
            if (SUCCEEDED(hr))
            {
                wprintf(L"Successfully bound to object. DN: %s\n",
                       vartest.bstrVal);
                vartest.Clear();
            }
        }
        else
            wprintf(L"Binding failed with hr: %x\n",hr);

        // Bind again using the DN to get a regular ADsPath.
        szPath = L"LDAP://";
        szPath += var.bstrVal;
        hr = ADsGetObject(szPath, IID_IADs, (void**)&pTestWKO1);
        hr = pTestWKO1->get_ADsPath(&bstr);
        // Rename the WKO object.
        hr = pNewContainer->MoveHere(bstr, szNewObjectRenameRDN,NULL);
        pTestWKO1.Release();
        // Bind again using WKGUID binding.
        // Build the ADsPath to the well-known object.
        szPath = L"LDAP://<WKGUID=";
        szPath += MyWKOTestObjectGUID;
        szPath += L",";
        szPath += szNewContainerDN;
        szPath += L">";
        wprintf(
            L"Bind AGAIN with the following WKGUID binding string: %s\n",
            szPath
        );
        hr = ADsGetObject(szPath, IID_IADs, (void**)&pTestWKO2);
        if (SUCCEEDED(hr))
        {
            hr = pTestWKO2->Get(CComBSTR(L"distinguishedName"),
                                   &vartest);
            if (SUCCEEDED(hr))
            {
                wprintf(L"Successfully bound to object. ");
                wprintf(L"Be aware that the DN reflects the rename.");
                wprintf(L" DN: %s\n",vartest.bstrVal);
                vartest.Clear();
            }
        }
        else
            wprintf(L"Binding failed with hr: %x\n",hr);
    }

}
}

}

}

// Offer user the option to delete test containers.
wprintf(L"Delete the test container and object (Y/N):");
    CComBSTR pszBuffer(MAX_PATH * 2);
    _getws_s(pszBuffer, MAX_PATH * 2);
if (pszBuffer == L"Y" )
{
    // Delete the object.
    // Delete the container.
    hr = pNewContainer->Delete(CComBSTR(L"container"),
                                 szNewObjectRenameRDN);
    if (SUCCEEDED(hr))
    {

```

```

        wprintf(L"Test object successfully deleted.\n");
        szRelPath = L"cn=";
        szRelPath += szContainer;
        // Delete the container.
        hr = pDomain->Delete(CComBSTR(L"container"), szRelPath);
        if (SUCCEEDED(hr))
            wprintf(L"Test container and its contents successfully deleted.\n");
        else
            wprintf(L"Failed to delete test container. hr: %x\n",hr);
    }
    else
        wprintf(L"Failed to delete test container. hr: %x\n",hr);
}
hr = S_FALSE;

}

}

}

// converts hexadecimal string to a octet encoded byte
array STDMETHODIMP ConvertHexGuidToArray(PCWSTR hexGuid, LPSAFEARRAY *sa) throw() {

    if (hexGuid == NULL) return E_INVALIDARG;

    int hexLen = (int)wcslen(hexGuid);
    HRESULT hr = S_OK;
    CTempBuffer<CHAR> sHex(hexLen);
    int loopLen = hexLen;
    // make ansi
    while(loopLen-- != 0)
        sHex[loopLen] = (CHAR) hexGuid[loopLen];
    SAFEARRAY *temp= SafeArrayCreateVector(VT_UI1, 0, hexLen / 2);
    if (temp == NULL)
        hr = E_OUTOFMEMORY;
    PBYTE dst ;
    if (hr == S_OK) hr = SafeArrayAccessData(temp, (void**)&dst);
    int dstLen = hexLen / 2;
    if (hr == S_OK)
    {
        AtlHexDecode(sHex, hexLen, dst, &dstLen );
        hr = SafeArrayUnaccessData(temp);
        *sa = temp;
    }
    return hr;
}

HRESULT AddValueToOtherWKOProperty(IADs *pads,
                                    LPOLESTR szContainerDN,
                                    // DN for container
                                    // whose otherWellKnownObjects
                                    // property to modify.
                                    PCWSTR pwKOGUID, // WKO GUID for the WKO.
                                    BSTR szWKOObjectDN // DN of the WKO.
                                    )
{
    HRESULT hr = E_FAIL;

    CComPtr<IADsDNWithBinary> pdnWithBin;
    CComBSTR retGuid;
    pdnWithBin.CoCreateInstance(CLSID_DNWithBinary);

    CComVariant v;
    v.vt = VT_UI1 | VT_ARRAY;
    ConvertHexGuidToArray(pwKOGUID, &v.parray);
}

```

```
    hr = pdnWithBin->put_DNString(szWK0ObjectDN);
    hr = pdnWithBin->put_BinaryValue(v);
    v.Clear();
    v.vt = VT_DISPATCH;
    pdnWithBin.QueryInterface(&v.pdispVal);

    SAFEARRAY *psa = SafeArrayCreateVector(VT_VARIANT, 0, 1);
    VARIANT *varray = (VARIANT*)psa->pvData;
    VariantCopy(&varray[0] ,&v);
    CComVariant v2;
    v2.vt = VT_VARIANT | VT_ARRAY;
    v2.parray = psa;
    hr = pads->PutEx(ADS_PROPERTY_APPEND,
                      CComBSTR(L"otherWellKnownObjects"),
                      v2);

    return hr;
}
```

Authentication (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

Every object in Active Directory Domain Services has a unique security descriptor that defines the access permissions that are required to read or update the object or its individual properties. Access privileges are determined by the rights granted to a user's account or group memberships.

When an application binds to an object in the directory, the access privileges that the application has to that object are based on the user context specified during the bind operation. For the binding functions and methods **ADsGetObject**, **ADsOpenObject**, **GetObject**, **IADsOpenDSObject::OpenDSObject**, an application can implicitly use the credentials of the caller, explicitly specify the credentials of a user account, or use an unauthenticated user context (Guest).

Searching in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Search is a key feature of Active Directory Domain Services. Search enables you to find objects in the directory based on selection criteria (query) and to retrieve specified properties for the objects found.

Searching within Active Directory Domain Services is a matter of finding a Domain Controller (DC), binding to the object where the search should begin in the directory, submitting a query, and processing the results.

For more information about the search feature in Active Directory, see:

- [Deciding What to Find](#)
- [Deciding Where to Search](#)
- [Choosing the Search Technology](#)
- [Creating a Query Filter](#)
- [Deciding Which Attributes to Retrieve for Each Object Found](#)
- [Binding to the Search Start Point](#)
- [Specifying the Search Scope](#)
- [Specifying Other Search Options](#)
- [Processing the Search Results](#)
- [Effects of Security on Searching](#)
- [Creating Efficient Queries](#)
- [Referrals](#)
- [Enumerating Domain Controllers](#)

Deciding What to Find

6/3/2022 • 2 minutes to read • [Edit Online](#)

Before you search a directory, consider how your search will perform based on your approach. The data and properties to be returned affect where you bind to start a search, the depth of your search, your query filter, and search performance.

For example, if you want search for all user objects with surname Smith:

AREA	DESCRIPTION
Where to search	A specific container or organizational unit (OU) within a domain, a specific domain, a specific domain tree, or the entire forest. If you search for objects within a specific container or domain, the search query will perform better by binding directly to that container or domain instead of performing a subtree search on a domain tree.
Type of search	If you verify the existence of, or retrieve the properties of a particular object that has a distinguished name (DN) you already know, you should perform a base search, which searches only the object you have bound to. If you know an object is a direct descendant of a particular container, bind to that container and do a one-level search (attributeSchema and classSchema objects in the schema container and extended-right objects in the extended-rights container are good examples). If you do not know exactly where the object is, or if you want to search the object you have bound to and all the child objects below it in the directory hierarchy, perform a subtree search.
Use indexes where possible	Finally, if you look for a specific class of object, the query filter should have expressions that evaluate properties that are defined for that class. To search for group objects, include the expression (objectCategory=group) in the filter. To search for user objects, specify (&(objectClass=user)(objectCategory=person)) because the computer class derives from the user class, so (objectClass=user) would return both users and computers and also because both contact and user objects have an objectCategory of person, so (objectCategory=person) would return both users and contacts. For more information, see Object Class and Object Category and Indexed Attributes .

Example Code for Searching for Users

6/3/2022 • 9 minutes to read • [Edit Online](#)

The following code examples search for users in the domain of the user account under which the calling process is running.

- [C++ Example](#)
- [Visual Basic Example](#)

C++ Example

The following C++ code example searches the current domain for all users, or a specific user based on a specified filter.

```
// Add msrvct.dll to the project.  
// Add activeds.lib to the project.  
  
#include "stdafx.h"  
#include <objbase.h>  
#include <wchar.h>  
#include <activeds.h>  
// Define UNICODE.  
// Define version 5 for Windows 2000.  
#define _WIN32_WINNT 0x0500  
#include <sddl.h>  
  
HRESULT FindUsers(IDirectorySearch *pContainerToSearch, // IDirectorySearch pointer to the container to  
search.  
    LPOLESTR szFilter, // Filter to find specific users.  
    // NULL returns all user objects.  
    LPOLESTR *pszPropertiesToReturn, // Properties to return for user objects found.  
    // NULL returns all set properties.  
    BOOL bIsVerbose // TRUE indicates that display all properties for the found objects.  
    // FALSE indicates that only the RDN.  
);  
  
// Entry point for the application.  
void wmain(int argc, wchar_t *argv[ ])  
{  
#ifdef _MBCS  
    // Handle the command line arguments.  
    DWORD dwLength = MAX_PATH*2;  
    LPOLESTR pszBuffer = new OLECHAR[dwLength];  
    wcsncpy_s(pszBuffer, L"", dwLength);  
    BOOL bReturnVerbose = FALSE;  
  
    for (int i = 1;i<argc;i++)  
    {  
        if (_wcsicmp(argv[i],L"/V") == 0)  
        {  
            bReturnVerbose = TRUE;  
        }  
        else if ((-_wcsicmp(argv[i],L"/?") == 0)||  
                 (-_wcsicmp(argv[i],L"-?") == 0))  
        {  
            wprintf(L"This application queries for users in the current user domain.\n");  
            wprintf(L"Syntax: queryusers [/V][querystring]\n");  
            wprintf(L"where /V specifies that all properties for the found users should be returned.\n");  
            wprintf(L"querystring is the query criteria in ldap query format.\n");  
        }  
    }  
}
```

```

wprintf(L"Defaults: If no /V is specified, the query returns only the RDN and DN of the items
found.\n");
    wprintf(L"If no querystring is specified, the query returns all users.\n");
    wprintf(L"Example: queryusers (sn=Smith)\n");
    wprintf(L>Returns all users with surname Smith.\n");
    return;
}
else
{
    wcsncpy_s(pszBuffer,argv[i],dwLength-wcslen(pszBuffer));
}
}

if (_wcsicmp(pszBuffer,L "") == 0)
    wprintf(L"\nFinding all user objects...\n\n");
else
    wprintf(L"\nFinding user objects based on query: %s...\n\n", pszBuffer);

// Initialize COM.
CoInitialize(NULL);
HRESULT hr = S_OK;
// Get rootDSE and the current user domain container distinguished name.
IADs *pObject = NULL;
IDirectorySearch *pContainerToSearch = NULL;
LPOLESTR szPath = new OLECHAR[MAX_PATH];
VARIANT var;
hr = ADsOpenObject(L"LDAP://rootDSE",
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                    IID_IADs,
                    (void**)&pObject);
if (FAILED(hr))
{
    wprintf(L"Cannot execute query. Cannot bind to LDAP://rootDSE.\n");
    if (pObject)
        pObject->Release();
    return;
}
if (SUCCEEDED(hr))
{
    hr = pObject->Get(_bstr_t("defaultNamingContext"),&var);
    if (SUCCEEDED(hr))
    {
        // Build path to the domain container.
        wcsncpy_s(szPath,L"LDAP://",MAX_PATH);
        wcsncat_s(szPath,var.bstrVal,MAX_PATH-wcslen(szPath));
        hr = ADsOpenObject(szPath,
                           NULL,
                           NULL,
                           ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                           IID_IDirectorySearch,
                           (void**)&pContainerToSearch);

        if (SUCCEEDED(hr))
        {
            hr = FindUsers(pContainerToSearch, // IDirectorySearch pointer to domainDNS container.
                           pszBuffer,
                           NULL, // Return all properties.
                           bReturnVerbose
                           );
            if (SUCCEEDED(hr))
            {
                if (S_FALSE==hr)
                    wprintf(L"User object cannot be found.\n");
            }
            else if (E_ADS_INVALID_FILTER==hr)
                wprintf(L"Cannot execute query. Invalid filter was specified.\n");
            else
                wprintf(L"Query failed to run. HRESULT: %x\n",hr);
        }
    }
}

```

```

    }
    else
    {
        wprintf(L"Cannot execute query. Cannot bind to the container.\n");
    }
    if (pContainerToSearch)
        pContainerToSearch->Release();
    }
    VariantClear(&var);
}
if (pObject)
    pObject->Release();

// Uninitialize COM.
CoUninitialize();
delete [] szPath;
delete [] pszBuffer;
return;
#endif _MBCS
}

HRESULT FindUsers(IDirectorySearch *pContainerToSearch, // IDirectorySearch pointer to the container to search.
LPOLESTR szFilter, // Filter for finding specific users.
// NULL returns all user objects.
LPOLESTR *pszPropertiesToReturn, // Properties to return for user objects found.
// NULL returns all set properties.
BOOL bIsVerbose // TRUE indicates that all properties for the found objects are displayed.
// FALSE indicates only the RDN.
)
{
    if (!pContainerToSearch)
        return E_POINTER;
    DWORD dwLength = MAX_PATH*2;
    // Create search filter.
    LPOLESTR pszSearchFilter = new OLECHAR[dwLength];

    // Add the filter.
    swprintf_s(pszSearchFilter, dwLength, L"(&(objectClass=user)(objectCategory=person)%s",szFilter);

    // Specify subtree search.
    ADS_SEARCHPREF_INFO SearchPrefs;
    SearchPrefs.dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    SearchPrefs.vValue.dwType = ADSTYPE_INTEGER;
    SearchPrefs.vValue.Integer = ADS_SCOPE_SUBTREE;
    DWORD dwNumPrefs = 1;

    // COL for iterations.
    LPOLESTR pszColumn = NULL;
    ADS_SEARCH_COLUMN col;
    HRESULT hr = S_OK;

    // Interface Pointers
    IADs *pObj = NULL;
    IADs * pIADs = NULL;

    // Search handle.
    ADS_SEARCH_HANDLE hSearch = NULL;

    // Set search preference.
    hr = pContainerToSearch->SetSearchPreference(&SearchPrefs, dwNumPrefs);
    if (FAILED(hr))
        return hr;

    LPOLESTR pszBool = NULL;
    DWORD dwBool = 0;
    PSID pObjectSID = NULL;
    LPOLESTR szSID = NULL;
    LPOLESTR szDSGUID = new WCHAR [39];

```

```

LPGUID pObjectGUID = NULL;
FILETIME filetime;
SYSTEMTIME systemtime;
DATE date;
VARIANT varDate;
LARGE_INTEGER liValue;
LPOLESTR *pszPropertyList = NULL;
LPOLESTR pszNonVerboseList[] = {L"name",L"distinguishedName"};

LPOLESTR szName = new OLECHAR[MAX_PATH];
LPOLESTR szDN = new OLECHAR[MAX_PATH];

VariantInit(&varDate);

int iCount = 0;
DWORD x = 0L;

if (!bIsVerbose)
{
    // Return non-verbose list properties only.
    hr = pContainerToSearch->ExecuteSearch(pszSearchFilter,
                                              pszNonVerboseList,
                                              sizeof(pszNonVerboseList)/sizeof(LPOLESTR),
                                              &hSearch
                                             );
}
else
{
    if (!pszPropertiesToReturn)
    {
        // Return all properties.
        hr = pContainerToSearch->ExecuteSearch(pszSearchFilter,
                                                 NULL,
                                                 (DWORD)-1,
                                                 &hSearch
                                                );
    }
    else
    {
        // Specified subset.
        pszPropertyList = pszPropertiesToReturn;
        // Return specified properties.
        hr = pContainerToSearch->ExecuteSearch(pszSearchFilter,
                                                 pszPropertyList,
                                                 sizeof(pszPropertyList)/sizeof(LPOLESTR),
                                                 &hSearch
                                                );
    }
}
if (SUCCEEDED(hr))
{
    // Call IDirectorySearch::GetNextRow() to retrieve the next data row.
    hr = pContainerToSearch->GetFirstRow(hSearch);
    if (SUCCEEDED(hr))
    {
        while(hr != S_ADS_NOMORE_ROWS)
        {
            // Keep track of count.
            iCount++;
            if (bIsVerbose)
                wprintf(L"-----\n");
            // Loop through the array of passed column names,
            // print the data for each column.

            while(pContainerToSearch->GetNextColumnName(hSearch, &pszColumn) != S_ADS_NOMORE_COLUMNS)
            {
                hr = pContainerToSearch->GetColumn(hSearch, pszColumn, &col);
                if (SUCCEEDED(hr))
                    wprintf(L"%s\t", col);
            }
        }
    }
}

```

```

    ...
    if (SUCCEEDED(hr))
    {
        // Print the data for the column and free the column.
        if(bIsVerbose)
        {
            // Get the data for this column.
            wprintf(L"%s\n",col.pszAttrName);
            switch (col.dwADSType)
            {
                case ADSTYPE_DN_STRING:
                    for (x = 0; x < col.dwNumValues; x++)
                    {
                        wprintf(L" %s\r\n",col.pADSValues[x].DNString);
                    }
                    break;
                case ADSTYPE_CASE_EXACT_STRING:
                case ADSTYPE_CASE_IGNORE_STRING:
                case ADSTYPE_PRINTABLE_STRING:
                case ADSTYPE_NUMERIC_STRING:
                case ADSTYPE_TYPEDNAME:
                case ADSTYPE_FAXNUMBER:
                case ADSTYPE_PATH:
                    for (x = 0; x < col.dwNumValues; x++)
                    {
                        wprintf(L" %s\r\n",col.pADSValues[x].CaseIgnoreString);
                    }
                    break;
                case ADSTYPE_BOOLEAN:
                    for (x = 0; x < col.dwNumValues; x++)
                    {
                        dwBool = col.pADSValues[x].Boolean;
                        pszBool = dwBool ? L"TRUE" : L"FALSE";
                        wprintf(L" %s\r\n",pszBool);
                    }
                    break;
                case ADSTYPE_INTEGER:
                    for (x = 0; x < col.dwNumValues; x++)
                    {
                        wprintf(L" %d\r\n",col.pADSValues[x].Integer);
                    }
                    break;
                case ADSTYPE_OCTET_STRING:
                    if (_wcsicmp(col.pszAttrName,L"objectSID") == 0)
                    {
                        for (x = 0; x < col.dwNumValues; x++)
                        {
                            pObjectSID = (PSID)(col.pADSValues[x].OctetString.lpValue);
                            // Convert SID to string.
                            ConvertSidToStringSid(pObjectSID, &szSID);
                            wprintf(L" %s\r\n",szSID);
                            LocalFree(szSID);
                        }
                    }
                    else if (_wcsicmp(col.pszAttrName,L"objectGUID") == 0)
                    {
                        for (x = 0; x < col.dwNumValues; x++)
                        {
                            // Cast to LPGUID.
                            pObjectGUID = (LPGUID)(col.pADSValues[x].OctetString.lpValue);
                            // Convert GUID to string.
                            ::StringFromGUID2(*pObjectGUID, szDSGUID, 39);
                            // Print the GUID.
                            wprintf(L" %s\r\n",szDSGUID);
                        }
                    }
                    else
                        wprintf(L" Value of type Octet String. No Conversion.");
                    break;
                case ADSTYPE_UTCTIME:
                    ...

```

```

case ADSTYPE_UTCTIME:
    for (x = 0; x < col.dwNumValues; x++)
    {
        systemtime = col.pADsValues[x].UTCTime;
        if (SystemTimeToVariantTime(&systemtime,
            &date) != 0)
        {
            // Pack in variant.vt.
            varDate.vt = VT_DATE;
            varDate.date = date;
            VariantChangeType(&varDate,&varDate,VARIANT_NOVALUEPROP,VT_BSTR);
            wprintf(L" %s\r\n",varDate.bstrVal);
            VariantClear(&varDate);
        }
        else
            wprintf(L" Could not convert UTC-Time.\n",pszColumn);
    }
    break;
case ADSTYPE_LARGE_INTEGER:
    for (x = 0; x < col.dwNumValues; x++)
    {
        liValue = col.pADsValues[x].LargeInteger;
        filetime.dwLowDateTime = liValue.LowPart;
        filetime.dwHighDateTime = liValue.HighPart;
        if((filetime.dwHighDateTime==0) && (filetime.dwLowDateTime==0))
        {
            wprintf(L" No value set.\n");
        }
        else
        {
            // Verify properties of type LargeInteger that represent time.
            // If TRUE, then convert to variant time.
            if ((0==wcscmp(L"accountExpires", col.pszAttrName)) ||
                (0==wcscmp(L"badPasswordTime", col.pszAttrName)) ||
                (0==wcscmp(L"lastLogon", col.pszAttrName)) ||
                (0==wcscmp(L"lastLogoff", col.pszAttrName)) ||
                (0==wcscmp(L"lockoutTime", col.pszAttrName)) ||
                (0==wcscmp(L"pwdLastSet", col.pszAttrName)))
            )
            {
                // Handle special case for Never Expires where low part is -1.
                if (filetime.dwLowDateTime==-1)
                {
                    wprintf(L" Never Expires.\n");
                }
                else
                {
                    if (FileTimeToLocalFileTime(&filetime, &filetime) != 0)
                    {
                        if (FileTimeToSystemTime(&filetime,
                            &systemtime) != 0)
                        {
                            if (SystemTimeToVariantTime(&systemtime,
                                &date) != 0)
                            {
                                // Pack in variant.vt.
                                varDate.vt = VT_DATE;
                                varDate.date = date;

VariantChangeType(&varDate,&varDate,VARIANT_NOVALUEPROP,VT_BSTR);
                                wprintf(L" %s\r\n",varDate.bstrVal);
                                VariantClear(&varDate);
                            }
                            else
                            {
                                wprintf(L" FileTimeToVariantTime failed\n");
                            }
                        }
                    }
                }
            }
        }
    }

```

```

        {
            wprintf(L"  FileTimeToSystemTime failed\n");
        }

    }
else
{
    wprintf(L"  FileTimeToLocalFileTime failed\n");
}
}

}

else
{
    // Print the LargeInteger.
    wprintf(L"  high: %d low: %d\r\n",filetime.dwHighDateTime,
filetime.dwLowDateTime);
}

}

}

break;
case ADSTYPE_NT_SECURITY_DESCRIPTOR:
    for (x = 0; x< col.dwNumValues; x++)
    {
        wprintf(L"    Security descriptor.\n");
    }
    break;
default:
    wprintf(L"Unknown type %d.\n",col.dwADsType);
}
}
else
{
#endif _MBCS
    // Verbose handles only the two single-valued attributes: cn and
ldapdisplayname,
    // so this is a special case.
    if (0==wcscmp(L"name", pszColumn))
    {
        wcscopy_s(szName,col.pADsValues->CaseIgnoreString);
    }
    if (0==wcscmp(L"dn", pszColumn))
    {
        wcscopy_s(szDN,col.pADsValues->CaseIgnoreString);
    }
#endif _MBCS
}
pContainerToSearch->FreeColumn(&col);
}
FreeADsMem(pszColumn);
}
if (!bIsVerbose)
    wprintf(L"%s\n  DN: %s\n\n",szName,szDN);
// Get the next row.
hr = pContainerToSearch->GetNextRow(hSearch);
}

}

// Close the search handle to cleanup.
pContainerToSearch->CloseSearchHandle(hSearch);
}
if (SUCCEEDED(hr) && 0==iCount)
    hr = S_FALSE;

delete [] szName;
delete [] szDN;
delete [] szDSGUID;
delete [] pszSearchFilter;
return hr;
}

```

Visual Basic Example

The following Visual Basic code example searches the current domain for users with the specified surname and returns the name and **distinguishedName** attributes for the objects that are found. The following code example uses ADO to perform the search.

```
Dim Con As ADODB.Connection
Dim ocommand As ADODB.Command
Dim gc As IADs

On Error Resume Next
' Maximum number of items to list on a msgbox.
MAX_DISPLAY = 5

' Prompt for surname to search for.
strName = InputBox("This routine searches in the current domain for users with the specified surname." &
vbCrLf & vbCrLf &"Specify the surname:")

If strName = "" Then
    msgbox "No surname was specified. The routine will search for all users."
End If

' Create ADO connection object for Active Directory
Set Con = CreateObject("ADODB.Connection")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on CreateObject"
End If
Con.Provider = "ADsDSOObject"
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Provider"
End If
Con.Open "Active Directory Provider"
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Open"
End If

' Create ADO command object for the connection.
Set ocommand = CreateObject("ADODB.Command")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on CreateObject"
End If
ocommand.ActiveConnection = Con
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Active Connection"
End If

' Get the ADsPath for the domain to search.
Set root = GetObject("LDAP://rootDSE")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject for rootDSE"
End If
sDomain = root.Get("defaultNamingContext")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get on defaultNamingContext"
End If
Set domain = GetObject("LDAP://" & sDomain)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject for domain"
End If

' Build the ADsPath element of the commandtext
sADsPath = "<" & domain.ADsPath & ">"
```

```

' Build the filter element of the commandtext
If (strName = "") Then
    sFilter = "(&(objectCategory=person)(objectClass=user))"
Else
    sFilter = "(&(objectCategory=person)(objectClass=user)(sn=" & strName & "))"
End If

' Build the returned attributes element of the commandtext.
sAttribsToReturn = "name,distinguishedName"

' Build the depth element of the commandtext.
sDepth = "subTree"

' Assemble the commandtext.
oCommand.CommandText = sADsPath & ";" & sFilter & ";" & sAttribsToReturn & ";" & sDepth
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on CommandText"
End If
' Display.
show_items "CommandText: " & oCommand.CommandText, ""

' Execute the query.
Set rs = oCommand.Execute
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Execute"
End If

strText = "Found " & rs.RecordCount & " Users in the domain:"
intNumDisplay = 0
intCount = 0

' Navigate the record set.
rs.MoveFirst
While Not rs.EOF
    intCount = intCount + 1
    strText = strText & vbCrLf & intCount & ") "
    For i = 0 To rs.Fields.Count - 1
        If rs.Fields(i).Type = adVariant And Not (IsNull(rs.Fields(i).Value)) Then
            strText = strText & rs.Fields(i).Name & " = "
            For j = LBound(rs.Fields(i).Value) To UBound(rs.Fields(i).Value)
                strText = strText & rs.Fields(i).Value(j) & " "
            Next
        Else
            strText = strText & rs.Fields(i).Name & " = " & rs.Fields(i).Value & vbCrLf
        End If
    Next
    intNumDisplay = intNumDisplay + 1
    ' Display in msgbox if there are MAX_DISPLAY items to display.
    If intNumDisplay = MAX_DISPLAY Then
        Call show_items(strText, "Users in domain")
        strText = ""
        intNumDisplay = 0
    End If
    rs.MoveNext
Wend

show_items strText, "Users in domain"
.....
' Display subroutines
.....
Sub show_items(strText, strName)
    MsgBox strText, vbInformation, "Search domain for users with Surname " & strName
End Sub

Sub BailOnFailure(ErrNum, ErrText)    strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub

```


Deciding Where to Search

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic explains the different searches that can be performed in Active Directory Domain Services.

All searches are performed in one of the following naming contexts:

- Domain, including application directory partitions
- Schema container
- Configuration container
- Global catalog

Searching a Domain

Domains contain most of the highly used objects such as users, contacts, groups, organizational units, computers, and so on. Most queries will search a domain. For more information about searching a domain, see [Searching Domain Contents](#).

Searching the Schema Container

The schema container contains the **classSchema** and **attributeSchema** objects that provide the formal definition of every class and attribute that can exist in Active Directory Domain Services. To search for objects in the schema container, bind to the schema container on any domain controller. The schema container is available on all domain controllers. The distinguished name of the schema container is obtained from the **schemaNamingContext** attribute from rootDSE. For more information about rootDSE and the **schemaNamingContext** attribute, see [Serverless Binding and RootDSE](#).

For more information about reading from the schema or abstract schema container, see [Guidelines for Binding to the Schema](#).

Searching the Configuration Container

The configuration container contains configuration information, such as sites, services and display specifiers, for the entire forest. To search for objects in the configuration container, bind to the configuration container on any domain controller. The configuration container is available on all domain controllers. The distinguished name of the configuration container is obtained from the **configurationNamingContext** attribute from rootDSE. For more information about rootDSE and the **configurationNamingContext** attribute, see [Serverless Binding and RootDSE](#).

Searching the Global Catalog

The global catalog holds a replica of every object in Active Directory Domain Services, but with only a small number of their attributes. The attributes in the global catalog are those most frequently used in search operations, such as a user's first and last names, login names, and so on. For more information about searching the global catalog, see [Searching the Global Catalog](#).

Searching Domain Contents

6/3/2022 • 2 minutes to read • [Edit Online](#)

Before discussing where to bind to begin a search for objects in a domain, it is helpful to understand how data is stored in Active Directory Domain Services.

If you have a forest with more than one domain, Active Directory Domain Services does not store all object data on a single domain controller — for performance, scalability, and reliability reasons. A domain controller holds all information about only the domain that it is a member of (it has a full replica of the domain). But a domain controller does not hold complete information about any other domain.

If you bind to the domain object with referral chasing turned off, you can search for any object in that domain and only that domain. For more information about referral chasing, see [Referrals](#). A search can retrieve any property and can use a query filter containing any property.

In a forest, domains are arranged hierarchically as domain trees. A domain tree can be just a single domain or a domain with one or more child domains. These child domains, in turn, can have child domains beneath them. A domain tree is also a contiguous namespace. A contiguous namespace means that the child domains are a continuation of the naming hierarchy. For example, a domain fabrikam.com (or DC=Fabrikam, DC=COM) can have a child domain named mydivision (mydivision.fabrikam.com or DC=mydivision, DC=Fabrikam, DC=COM), which in turn could have a child domain named mydev (mydev.mydivision.fabrikam.com or DC=mydev, DC=mydivision, DC=Fabrikam, DC=COM).

If you bind to a domain object (with referral chasing turned on) for a domain within a domain tree, you will search that domain and the entire hierarchy within it. The search can retrieve any property and can use a query filter containing any property.

If a domain controller contains a full replica of only its own domain, you can perform a subtree search on a domain tree. A domain holds references to its child domains. When a domain controller processes a subtree search request against its own domain, the domain controller searches that domain and then returns referrals to each of its child domains to the client. A referral is the way that a directory server communicates that it does not contain the information required to complete a request (such as a query) but has a reference to a server that may contain the required information. In the case of a subtree search of a domain tree, a referral is returned for each direct child domain so that the search can be continued at a domain controller in each child domain. If referral chasing is turned on, the LDAP client library (Wldap32.dll) uses those referrals to bind to a domain controller in each child domain and continue the search. If referral chasing is turned off, the LDAP client does not resolve the referrals and the search is complete.

A subtree search on a domain tree with referral chasing turned on can be time-consuming if there is a slow connection to the domain controllers for the child domains. If you want to search only a single domain, you should turn referral chasing off to avoid having to search the child domains unnecessarily.

Searching the Global Catalog

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services also have a global catalog (GC), which contains a partial replica of all objects in the directory. It also contains partial replicas of the schema and configuration containers. One or more domain controllers in a domain can hold a copy of the global catalog. For more information about binding to a global catalog, see [Binding to the Global Catalog](#).

The global catalog holds a replica of every object in Active Directory Domain Services, but with only a small number of their attributes. The attributes in the global catalog are those most frequently used in search operations, such as a user's first and last names, login names, and so on. The global catalog attributes also include those required to locate a full replica of the object. The global catalog allows users to quickly find objects of interest without knowing what domain holds them and without requiring a contiguous extended namespace in the enterprise; that is, you can search the entire forest.

So, if you bind to an object in the global catalog, you will search that object and the entire hierarchy below it — without having to go to any other server. However, the search can only use a query filter containing properties in the global catalog and can retrieve only properties in the global catalog.

Searching the global catalog has the following benefits:

- Global catalog enables you to search the entire forest or any part of the forest as well as the schema and configuration containers.
- Global catalog enables you to perform a complete search on a single server. No referrals or referral chasing is required.

Searching the global catalog has the following disadvantages:

- Global catalog contains a small subset of the properties on each object. If your query filter includes properties that are not in the global catalog, the query will evaluate the expressions containing those properties as false. If you specify non-global catalog properties in the list of properties to return, those properties are not retrieved.
- To search the global catalog, a domain controller that contains a global catalog must be available. If one is not available, you cannot perform a global catalog search.
- The global catalog is read-only. This means you cannot bind to an object in the global catalog to create, modify, or delete objects.

Choosing the Search Technology

6/3/2022 • 2 minutes to read • [Edit Online](#)

The technologies, listed in the following table, can be used to search in Active Directory Domain Services.

TECHNOLOGY	DESCRIPTION
DirectorySearcher	The DirectorySearcher class is provided by the System.DirectoryServices namespace to allow searching within Active Directory Domain Services with .NET Framework. For more information, see Searching the Directory .
IDirectorySearch	ADSI provides the IDirectorySearch interface to query an Active Directory server, as well as other directory services such as NDS, using LDAP. IDirectorySearch is a COM interface that returns richly typed data, such as Integer, Octet String, String, Security Descriptor, UTC-Time, Large Integer, or Boolean. For more information about how to use IDirectorySearch , see Searching With the IDirectorySearch Interface .
OLE DB	OLE DB is a set of COM interfaces that provide applications with uniform access to data stored in diverse data sources, regardless of location or type. ADSI also provides an OLE DB provider for ADSI that enables applications to use OLE DB to access Active Directory Domain Services. The ADSI OLE DB provider uses the IDirectorySearch interfaces to submit queries to Active Directory Domain Services and to collect the results.
ADO and other OLE DB-based data access technologies	The ADSI OLE DB provider enables any data-access technology based on OLE DB, such as ADO, to search within Active Directory Domain Services.
LDAP API	Windows 2000 domain controllers are directory servers that are compliant with LDAP version 3. The LDAP API is a C-style function library. Applications can use the LDAP API to search within Active Directory Domain Services.

Consider the following when choosing a technology:

- For Microsoft Visual Basic and Visual Basic Scripting Edition (VBScript), ADO is recommended.
- For C/C++, you can choose any of the technologies.
- If your application extensively uses ADSI, it may be simpler to use [IDirectorySearch](#). If you use [IDirectoryObject](#) to manage objects in Active Directory Domain Services, use [IDirectorySearch](#) to make handling the properties returned from the search easier. [IDirectorySearch](#) uses the same [ADSVALUE](#) structures as [IDirectoryObject](#) to represent properties. In addition, [IDirectorySearch](#) is exposed on almost all ADSI COM objects. If you have a pointer to an ADSI COM object, you can call [QueryInterface](#) to get an [IDirectorySearch](#) pointer that you can use to perform a search starting at the directory object represented by the ADSI COM object.
- If your application already uses OLE DB, ADO, or LDAP API, you can continue to use those technologies to search within Active Directory Domain Services.

- If your application must join data from an Active Directory Domain Service and a SQL Server 7 database, use OLE DB. By using OLE DB, your application can perform distributed queries that reference Active Directory Domain Services and tables and rowsets from one or more Microsoft SQL Server 7 databases.

Creating a Query Filter

6/3/2022 • 2 minutes to read • [Edit Online](#)

A query filter instructs Active Directory Domain Services to find data in an LDAP query syntax. All the specified data access technologies listed in the [Choosing the Search Technology](#) topic support LDAP query syntax.

The LDAP query syntax is as follows:

```
<expression><expression>...
```

A filter can contain one, or more, expressions. An expression has the following form:

```
(<logicaloperator><comparison><comparison...>)
```

where "<logicaloperator>" is one of the following.

OPERATOR	DESCRIPTION
" "	Logical OR
"&"	Logical AND
"!"	Logical NOT

and "<comparison>" is the following:

```
(<attribute><operator><value>)
```

where "<attribute>" is the **IDAPDisplayName** of the attribute to evaluate, "<value>" is the value to compare against, and "<operator>" is one of the following comparison operators.

OPERATOR	DESCRIPTION
"= "	Equals
"~ = "	Approximately equals
"<= "	Less than or equal to
">= "	Greater than or equal to

In addition, depending on the attribute syntax, the "<value>" may contain the wildcard symbol ("*"). A "<value>" that contains only a wildcard will check for the existence of any value in "<attribute>". If no value is set for "<attribute>", the test will fail.

If any of the following special characters must appear in the query filter as literals, they must be replaced by the listed escape sequence.

ASCII CHARACTER	ESCAPE SEQUENCE SUBSTITUTE
*	"\2a"
("\28"
)	"\29"
\	"\5c"
NUL	"\00"

In addition, arbitrary binary data may be represented using the escape sequence syntax by encoding each byte of binary data with the backslash followed by two hexadecimal digits. For example, the four-byte value 0x00000004 is encoded as "\00\00\00\04" in a filter string.

Examples

The following query string will search for all objects of type "computer".

```
(objectCategory=computer)
```

The following query string will search for all objects of type "computer" with a name that begins with "desktop".

```
(&(objectCategory=computer)(name=desktop*))
```

The following query string will search for all objects of type "computer" with a name that begins with "desktop" or a name that begins with "notebook".

```
(&(objectCategory=computer)(|(name=desktop*)(name=notebook*)))
```

The following query string will search for all objects of type "user" that have a home phone number.

```
(&(objectCategory=user)(homePhone=*))
```

For more information about query filter strings, and usage examples, see:

- [Finding Objects by Class](#)
- [Finding Objects by Name](#)
- [Finding a List of Attributes To Query](#)
- [Checking the Query Filter Syntax](#)
- [How to Specify Comparison Values](#)

Finding Objects by Class

6/3/2022 • 2 minutes to read • [Edit Online](#)

A typical search queries for a specific object class. The following code example searches for computers with location in Building 7N.

```
(&(objectCategory=computer)(location=Building 7N))
```

Consider why **objectClass** is not used. Do not use **objectClass** without another comparison that contains an indexed attribute. Index attributes can increase the efficiency of a query. The **objectClass** attribute is multi-valued and not indexed. To specify the type or class of an object, use **objectCategory**.

Less efficient:

```
(objectClass=computer)
```

More Efficient:

```
(objectCategory=computer)
```

Be aware that there are some cases where a combination of **objectClass** and **objectCategory** must be used. The user class and contact class should be specified as follows.

```
(&(objectClass=user)(objectCategory=person))  
(&(objectClass=contact)(objectCategory=person))
```

Be aware that you could search for both users and contacts with the following.

```
(objectCategory=person)
```

Finding Objects by Name

6/3/2022 • 2 minutes to read • [Edit Online](#)

Most objects in Active Directory Domain Services use the **cn** property as their naming attribute. Some objects, however, use a naming attribute other than **cn**. For example, a domain controller uses the **domainDNS** property for the naming attribute and an organizational unit uses the **organizationalUnit** property for the naming attribute. To avoid having to use a different naming attribute for different object types, the **name** property, which contains the relative distinguished name of the object, should be used to search for objects by name.

Examples

The following code examples show different query strings that can be used to find objects by name.

The following query string finds all objects with a name that begins with "Jeff".

```
(name=Jeff*)
```

The following query string finds all computer objects with a name that begins with "leased" or "corp".

```
(&(objectCategory=computer)(|(name=leased*)(name=corp*)))
```

The following query string finds all users and with a name that begins with "Karen" or "Jeff".

```
(&(&(objectClass=user)(objectCategory=person))(|(name=Karen*)(name=Jeff*)))
```

Finding a List of Attributes To Query

6/3/2022 • 2 minutes to read • [Edit Online](#)

When searching for objects of a particular class, comparisons in your search filter should specify attributes that actually exist on the objects of that class. To get the list attributes on an object of a particular class, bind to that class in the abstract schema and retrieve the **IADsClass.MandatoryProperties** and **IADsClass.OptionalProperties** properties. For more information, see [Reading the Abstract Schema](#).

In addition, all objects inherit from the top abstract class. Therefore, any attribute in **top** can exist, although it may not be set, on any object.

If searching the global catalog, ensure that you specify attributes in the global catalog. Attributes included in the global catalog have the **isMemberOfPartialAttributeSet** set to **TRUE** on their **attributeSchema** objects. Be aware that this data is not available in the abstract schema; read it from the **attributeSchema** object in the schema container.

In the global catalog, a back link attribute can be queried only if both of the following conditions are met: First, the attribute is marked for inclusion in the global catalog. Second, the corresponding forward link is also marked for inclusion in the global catalog. This applies to query filters as well as query results. For more information, see [Linked Attributes](#).

In addition, some attributes, mostly on the user object, are constructed. Query filters cannot contain constructed attributes. Constructed attributes cannot be evaluated in query filters; however, they can be returned in query results. This applies to all the naming contexts and the global catalog. Attributes that are constructed have **ADS_SYSTEMFLAG_ATTR_IS_CONSTRUCTED** (0x00000004) in the **systemFlags** property on their **attributeSchema** objects.

NOTE

For more information about predefined classes and attributes included with the system, see [Active Directory Domain Services Reference](#). These pages list mandatory and optional attributes of each object class. For attributes, the reference page indicates whether the attribute is indexed, constructed, linked, or in the global catalog.

Checking the Query Filter Syntax

6/3/2022 • 2 minutes to read • [Edit Online](#)

The LDAP API provides a simple syntax-verification function. Be aware that it only verifies the syntax and not the existence of the properties specified in the filter.

The following function verifies the syntax of the query filter and returns S_OK if the filter is valid or S_FALSE if it is not.

```
HRESULT CheckFilterSyntax(
    LPOLESTR szServer, // NULL binds to a DC in the current domain.
    LPOLESTR szFilter) // Filter to check.
{
    HRESULT hr = S_OK;
    DWORD dwReturn;
    LDAP *hConnect = NULL; // Connection handle

    if (!szFilter)
        return E_POINTER;

    // LDAP_PORT is the default port, 389

    hConnect = ldap_open(szServer, LDAP_PORT);

    // Bind using the preferred authentication method on Windows 2000
    // and the calling thread's security context.

    dwReturn = ldap_bind_s( hConnect, NULL, NULL, LDAP_AUTH_NEGOTIATE );
    if (dwReturn==LDAP_SUCCESS) {
        dwReturn = ldap_check_filter(hConnect, szFilter);
        if (dwReturn==LDAP_SUCCESS)
            hr = S_OK;
        else
            hr = S_FALSE;
    }

    // Unbind to free the connection.

    ldap_unbind( hConnect );

    return hr;
}
```

How to Specify Comparison Values

6/3/2022 • 7 minutes to read • [Edit Online](#)

Each attribute type has a syntax that determines the type of comparison values that you can specify in a search filter for that attribute.

The following sections describe requirements for each attribute syntax. For more information about attribute syntaxes, see [Syntaxes for Attributes in Active Directory Domain Services](#).

Boolean

The value specified in a filter must be a string value that is either "TRUE" or "FALSE". The following examples show how to specify a Boolean comparison string.

The following example will search for objects that have a **showInAdvancedViewOnly** set to TRUE:

```
(showInAdvancedViewOnly=TRUE)
```

The following example will search for objects that have a **showInAdvancedViewOnly** set to FALSE:

```
(showInAdvancedViewOnly=FALSE)
```

Integer and Enumeration

The value specified in a filter must be a decimal Integer. Hexadecimal values must be converted to decimal. A value comparison string takes the following form:

```
<attribute name>:<value>
```

"<attribute name>" is the **IDAPDisplayName** of the attribute and "<value>" is the value to use for comparison.

The following code example shows a filter that will search for objects that have a **groupType** value that is equal to the **ADS_GROUP_TYPE_UNIVERSAL_GROUP** (8) flag and the **ADS_GROUP_TYPE_SECURITY_ENABLED** (0x80000000) flag. The two flags combined equal 0x80000008, which converted to decimal is 2147483656.

```
(groupType=2147483656)
```

The LDAP matching rule operators can also be used to perform bitwise comparisons. For more information about matching rules, see [Search Filter Syntax](#). The following code example shows a filter that will search for objects that have a **groupType** with the **ADS_GROUP_TYPE_SECURITY_ENABLED** (0x80000000 = 2147483648) bit set.

```
(groupType:1.2.840.113556.1.4.803:=2147483648))
```

OctetString

The value specified in a filter is the data to be found. The data must be represented as a two character encoded

byte string where each byte is preceded by a backslash (\). For example, the value 0x05 will appear in the string as "\05".

The **ADsEncodeBinaryData** function can be used to create an encoded string representation of binary data. The **ADsEncodeBinaryData** function does not encode byte values that represent alpha-numeric characters. It will, instead, place the character into the string without encoding it. This results in the string containing a mixture of encoded and unencoded characters. For example, if the binary data is 0x05|0x1A|0x1B|0x43|0x32, the encoded string will contain "\05\1A\1BC2". This has no effect on the filter and the search filters will work correctly with these types of strings.

Wildcards are accepted.

The following code example shows a filter that contains encoded string for **schemaIDGUID** with GUID value of "{BF967ABA-0DE6-11D0-A285-00AA003049E2}":

(schemaidguid=\BA\7A\96\BF\E6\0D\0D\11\A2\85\00\AA\00\30\49\E2)

Sid

The value specified in a filter is the encoded byte string representation of the SID. For more information about encoded byte strings, see the previous section in this topic which discusses `OctetString` syntax.

The following code example shows a filter that contains an encoded string for `objectSid` with SID string value of "S-1-5-21-1935655697-308236825-1417001333":

(ObjectSid=\01\04\00\00\00\00\05\15\00\00\00\11\c3\5Fs\19R\5F\12u\B9ut)

DN

The entire distinguished name, to be matched, must be supplied.

Wildcards are not accepted.

Be aware that the **objectCategory** attribute also enables you to specify the **IDAPDisplayName** of the class set on the attribute.

The following example shows a filter that specifies a **member** that contains "CN=TestUser,DC=Fabrikam,DC=COM":

(member=CN=TestUser,DC=Fabrikam,DC=COM)

INTEGER8

The value specified in a filter must be a decimal integer. Convert hexadecimal values to decimal.

The following code example shows a filter that specifies a `creationTime` set to a [FILETIME](#) of "1999-12-31 23:59:59 (UTC/GMT)":

(creationTime=125911583990000000)

The following functions create an exact match (=) filter for a large integer attribute and verify the attribute in the schema and its syntax:

```
*****  
//  
//  
//  CheckAttribute()
```



```

LPOLESTR *pszFilter)

{
    HRESULT hr = E_FAIL;

    if ((!szAttribute) || (!pszFilter))
    {
        return E_POINTER;
    }

    // Verify that the attribute exists and has
    // Integer8 (Large Integer) syntax.

    hr = CheckAttribute(szAttribute, L"Integer8");
    if (S_OK == hr)
    {
        LPWSTR szFormat = L"%s=%I64d";
        LPWSTR szTempFilter = new WCHAR[lstrlenW(szFormat) + lstrlenW(szAttribute) + 20 + 1];

        if(NULL == szTempFilter)
        {
            return E_OUTOFMEMORY;
        }

        swprintf_s(szTempFilter, L"%s=%I64d", szAttribute, liValue);

        // Allocate buffer for the filter string.
        // Caller must free the buffer using CoTaskMemFree.
        *pszFilter = (OLECHAR *)CoTaskMemAlloc(sizeof(OLECHAR) * (lstrlenW(szTempFilter) + 1));
        if (*pszFilter)
        {
            wcscpy_s(*pszFilter, szTempFilter);
            hr = S_OK;
        }
        else
        {
            hr = E_OUTOFMEMORY;
        }

        delete szTempFilter;
    }

    return hr;
}

```

PrintableString

Attributes with these syntaxes should adhere to specific character sets. For more information, see [Syntaces for Attributes in Active Directory Domain Services](#).

Currently, Active Directory Domain Services do not enforce those character sets.

The value specified in a filter is a string. The comparison is case-sensitive.

GeneralizedTime

The value specified in a filter is a string that represents the date in the following form:

```
YYYYMMDDHHMMSS.0Z
```

"0Z" indicates no time differential. Be aware that the Active Directory server stores date/time as Greenwich Mean Time (GMT). If a time differential is not specified, the default is GMT.

If the local time zone is not GMT, use a differential value to specify your local time zone and apply the differential to GMT. The differential is based on: $\text{GMT} = \text{Local} + \text{differential}$.

To specify a differential, use:

```
YYYYMMDDHHMMSS.0[+/-]HHMM
```

The following example shows a filter that specifies a **whenCreated** time set to 3/23/99 8:52:58 PM GMT:

```
(whenCreated=19990323205258.0Z)
```

The following example shows a filter that specifies a **whenCreated** time set to 3/23/99 8:52:58 PM New Zealand Standard Time (differential is +12 hours):

```
(whenCreated=19990323205258.0+1200)
```

The following code example shows how to calculate time zone differential. The function returns the differential between the current local time zone and GMT. The value returned is a string in the following format:

```
[+/-]HHMM
```

For example, Pacific Standard Time is -0800.

```
//*****
//  GetLocalTimeZoneDifferential()
//*****  
  
HRESULT GetLocalTimeZoneDifferential(LPOLESTR *pszDifferential)
{
    if(NULL == pszDifferential)
    {
        return E_INVALIDARG;
    }

    HRESULT hr = E_FAIL;
    DWORD dwReturn;
    TIME_ZONE_INFORMATION timezoneinfo;
    LONG lTimeDifferential;
    LONG lHours;
    LONG lMinutes;

    dwReturn = GetTimeZoneInformation(&timezoneinfo);

    switch (dwReturn)
    {
        case TIME_ZONE_ID_STANDARD:
            lTimeDifferential = timezoneinfo.Bias + timezoneinfo.StandardBias;

            // Bias is in minutes. Calculate the hours for HHMM format.
            lHours = -(lTimeDifferential/60);

            // Bias is in minutes. Calculate the minutes for HHMM format.
            lMinutes = lTimeDifferential%60L;

            hr = S_OK;
            break;

        case TIME_ZONE_ID_DAYLIGHT:
            lTimeDifferential = timezoneinfo.Bias + timezoneinfo.DaylightBias;

            // Bias is in minutes. Calculate the hours for HHMM format.
```

```

// Apply the additive inverse.
// Bias is based on GMT=Local+Bias.
// A differential, based on GMT=Local-Bias, is required.
lHours = -(lTimeDifferential/60);

// Bias is in minutes. Calculate the minutes for HHMM format.
lMinutes = lTimeDifferential%60L;

hr = S_OK;
break;

case TIME_ZONE_ID_INVALID:
default:
    hr = E_FAIL;
    break;
}

if (SUCCEEDED(hr))
{
    // The caller must free the memory using CoTaskMemFree.
    *pszDifferential = (OLECHAR *)CoTaskMemAlloc(sizeof(OLECHAR) * (3 + 2 + 1));
    if (*pszDifferential)
    {
        swprintf_s(*pszDifferential, L"%+03d%02d", lHours, lMinutes);

        hr = S_OK;
    }
    else
    {
        hr = E_OUTOFMEMORY;
    }
}

return hr;
}

```

UTCTime

The value specified in a filter is a string that represents the date in the following form:

YYMMDDHHMMSSZ

Z indicates no time differential. Be aware that the Active Directory server stores date and time as GMT time. If a time differential is not specified, GMT is the default.

The seconds value ("SS") is optional.

If GMT is not the local time zone, apply a local differential value to specify your local time zone. The differential is: GMT=Local+differential.

To specify a differential, use the following form:

YYMMDDHHMMSS[+/-]HHMM

The following example shows a filter that specifies a **myTimeAttrib** time set to 3/23/99 8:52:58 PM GMT:

(myTimeAttrib=990323205258Z)

The following example shows a filter that specifies a **myTimeAttrib** time set to 3/23/99 8:52:58 PM without seconds specified:

```
(myTimeAttrib=9903232052Z)
```

The following example shows a filter that specifies a **myTimeAttrib** time set to 3/23/99 8:52:58 PM New Zealand Standard Time (differential is 12 hours). This is equivalent to 3/23/99 8:52:58 AM GMT.

```
(myTimeAttrib=990323205258+1200)
```

DirectoryString

The value specified in a filter is a string. DirectoryString can contain Unicode characters. The comparison is case-insensitive.

OID

The entire OID to be matched must be supplied.

Wildcards are not accepted.

The **objectCategory** attribute enables you to specify the **IDAPDisplayName** of the class set for the attribute.

The following example shows a filter that specifies **governsID** for volume class:

```
(governsID=1.2.840.113556.1.5.36)
```

Two equivalent filters that specifies **systemMustContain** attribute containing **uNCName**, which has an OID of 1.2.840.113556.1.4.137:

```
(SystemMustContain=uNCName)
```

```
(SystemMustContain=1.2.840.113556.1.4.137)
```

Other Syntaxes

The following syntaxes are evaluated in a filter similar to an octet string:

- ObjectSecurityDescriptor
- AccessPointDN
- PresentationAddresses
- ReplicaLink
- DNWithString
- DNWithOctetString
- ORName

Deciding Which Attributes to Retrieve for Each Object Found

6/3/2022 • 2 minutes to read • [Edit Online](#)

If you search for objects of a particular class, it seems logical that you retrieve attributes that actually exist on the objects of that class. Follow the same rules discussed in [Finding a List of Attributes To Query](#). Be aware that constructed attributes can be retrieved in query, but constructed attributes cannot be used in filters.

Retrieving the objectClass Attribute

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **objectClass** attribute contains the class of which the object is an instance, as well as all classes from which that class is derived. For example, the **user** class inherits from **top**, **person**, and **organizationalPerson**; therefore, the **objectClass** attribute contains the names of those classes, as well as **user**. So, how do you find out what class the object is an instance of? The **objectClass** attribute is the only attribute with multiple values that has ordered values. The first value is the top of the class hierarchy, which is the top class, and the last value is the most derived class, which is the class that the object is an instance of.

The following function takes a pointer to a column containing an **objectClass** attribute and returns the instantiated **objectClass** of the object.

```
HRESULT GetClass(ADS_SEARCH_COLUMN *pcol, LPOLESTR *ppClass)
{
    if (!pcol)
        return E_POINTER;

    HRESULT hr = E_FAIL;
    if (ppClass)
    {
        LPOLESTR szClass = new OLECHAR[MAX_PATH];
        wcscpy_s(szClass, L"");
        if (_wcscmp(pcol->pszAttrName,L"objectClass") == 0 )
        {
            for (DWORD x = 0; x< pcol->dwNumValues; x++)
            {
                wcscpy_s(szClass, pcol->pADsValues[x].CaseIgnoreString);
            }
        }
        if (0==wcscmp(L"", szClass))
        {
            hr = E_FAIL;
        }
        else
        {
            //Allocate memory for string.
            //Caller must free using CoTaskMemFree.
            *ppClass = (OLECHAR *)CoTaskMemAlloc (
                sizeof(OLECHAR)*(wcslen(szClass)+1));
            if (*ppClass)
            {
                wcscpy_s(*ppClass, szClass);
                hr = S_OK;
            }
            else
                hr=E_FAIL;
        }
    }
    return hr;
}
```

Binding to the Search Start Point

6/3/2022 • 2 minutes to read • [Edit Online](#)

The start point for a search is a container that directly or indirectly contains the objects searched for. The search will not be performed in containers above the search start point. For example, take the following hypothetical directory structure:

```
Domain A
  Container 1
    Object 1
    Object 2
  Container 2
    Object 3
    Object 4
```

If a search is performed for all objects and the search start point is "Container 2", only "Object 3" and "Object 4" would be retrieved. If the same search was performed with the search start point at "Container 1", "Object 1" and "Object 2" would be retrieved.

To bind to the search start point, bind to the ADsPath of the container that will be the search start point.

Specifying the Search Scope

6/3/2022 • 2 minutes to read • [Edit Online](#)

You can specify the scope of a search as either a base, one-level, or subtree search. Use the **ADS_SEARCHPREF_SEARCH_SCOPE** flag with the values of the [ADS_SCOPEENUM](#) enumeration to specify the search scope. The following list includes descriptions of the search types:

- **Base.** A base search limits the search to the base object. The maximum number of objects returned is always one. This search is useful to verify the existence of an object for retrieving group membership. For example, if you have an object distinguished name, and you must verify the object's existence based on the path, you can use a one-level search. If the search fails, you can assume that the object may have been renamed or moved to a different location, or you were given the wrong information about the object. Be aware that you should store the globally unique identifier (GUID) of the object instead of the distinguished name, if you wish to revisit an object. The GUID will always reference the same object, regardless of where the object is located within the directory hierarchy.
- **One-level.** A one-level search is restricted to the immediate children of a base object, but excludes the base object itself. This setting can perform a targeted search for immediate child objects of a parent object. For example, consider a parent object P1 and its immediate children: C1, C2, and C3. A one-level search evaluates C1, C2, and C3 against the search criteria, but does not evaluate P1. Use a one-level search to enumerate all children of an object. An [IADsContainer](#) enumeration translates to a one-level search.
- **Subtree.** A subtree search (or a deep search) includes all child objects as well as the base object. You can request the LDAP provider to chase referrals to other LDAP directory services, including other directory domains or forests.

Specifying Other Search Options

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services support most search options provided by the search technology. For more information about additional search options, see [Specifying Other Search Options with IDirectorySearch](#).

Processing the Search Results

6/3/2022 • 2 minutes to read • [Edit Online](#)

After the first call to [IDirectorySearch::GetFirstRow](#) or [IDirectorySearch::GetNextRow](#), either **S_OK**, **S_ADS_NOMORE_ROWS**, or an error result is returned.

If the return value is **S_ADS_NOMORE_ROWS**, no more objects matching the filter were found. If an error result is returned, the query failed. In both cases, you are not required to process the rows in the result because nothing was returned.

If **S_OK** is returned, a row has been retrieved. You can parse the columns by name using [IDirectorySearch::GetColumn](#). The name is the **IDAPDisplayName** of the attribute in the column. The set of all columns was defined by the **pAttributeNames** parameter of the [IDirectorySearch::ExecuteSearch](#) method. If **NULL** was specified, the set of all columns is the union of all properties found for all the objects returned. To read the entire set of columns returned for an object, use the [IDirectorySearch::GetNextColumnName](#) to iterate each column, and use the column name returned to call [IDirectorySearch::GetColumn](#).

The [IDirectorySearch::GetColumn](#) method returns an **ADS_SEARCH_COLUMN** structure that contains the attribute name, the type of the attribute, count of values, and a pointer to an array of **ADSVALUE** structures that contain the values. You can loop through the **ADSVALUE** structures to read the values for the property returned by the column. You must read the appropriate member of the **ADSVALUE** structure based on the **ADSTYPE** specified by the **dwADsType** member of the **ADS_SEARCH_COLUMN** structure (or the **dwType** member of the **ADSVALUE** structure). For example, if **dwADsType** was **ADSTYPE_INTEGER**, you would read the **Integer** member of each **ADSVALUE** structure.

For more information and a code example, see [Example Code for Searching for Users](#).

Effects of Security on Searching

6/3/2022 • 2 minutes to read • [Edit Online](#)

Security is an implicit filter when performing searches, enumerating containers, or reading properties.

ADSI can return NO_SUCH_PROPERTY or NO_SUCH_OBJECT errors even when the object exists if you do not have access to read attributes on the object.

For example, a caller may be able to enumerate the child objects in a container because the caller has LIST_CONTENTS rights on the container. But the same caller may not be able to access the enumerated objects if the caller does not have read access to the child objects. In this case, a query for a child object may return NO_SUCH_OBJECT even though the caller successfully enumerated the object.

If the caller does not have sufficient rights, the following return codes may be returned:

E_ADS_INVALID_DOMAIN_OBJECT

E_ADS_PROPERTY_NOT_SUPPORTED

E_ADS_PROPERTY_NOT_FOUND

Creating Efficient Queries

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following table identifies important concepts to consider when creating an efficient query.

Area	Description
Indexing	<p>Ensure that the query filter contains at least one indexed attribute.</p> <p>For more information, see Indexed Attributes.</p>
Class vs. Category	<p>The statement "objectClass=xyz" refers to directory objects in which "xyz" represents any class in the object class hierarchy, whereas "objectCategory=xyz", refers to those directory objects in which "xyz" identifies a specific class in the object class hierarchy. The objectClass property can take multiple values, whereas objectCategory takes a single value and is, thus, better suited for type matching of objects in a directory search.</p>
Text searching	<p>Avoid searching for text in the middle and on the end of a string.</p> <p>For example, "cn=*hille*" or "cn=*larouse".</p> <p>Using more specific matching criteria tends to increase search performance. This is because Active Directory Domain Services will evaluate all predicates, identifies the indices, and then chooses one index most likely to yield the smallest set of returned values. This technique does not work well with mid- and end-string searches. If you have no other option other than using these searches, you can define a tuple index for the attribute.</p> <p>For more details about tuple indexes see How Tuple Indexing Works.</p>
Subtree searching	<p>Use the global catalog if you are considering subtree searches. Chasing referrals requires extensive resources.</p> <p>For more information, see Specifying Other Search Options.</p>
Use of paging	<p>Assume that a subtree search will return a large result set. Use paging when performing subtree searches. The server will stream a large result set in chunks reducing the server side memory resources. This minimizes network usage and reduces the need for sending large chunks of data over a network.</p> <p>For more information, see Specifying Other Search Options.</p>
Combine searches	<p>Use multiple attributes for a search. One search of an object that reads two attributes is more efficient than two searches of the same object, each returning one attribute.</p>
Efficient use of bind	<p>Bind to an object one time and hold the binding handle for the rest of the session. Do not bind and unbind for each call. If you use ADO or OLE DB, do not create many connection objects.</p>

AREA	DESCRIPTION
RoodDSE caching	<p>Read the rootDSE one time and remember its contents for the rest of your session.</p> <p>For more information, see Serverless Binding and RootDSE.</p>
Reference persistance	<p>Persist references to objects as GUIDs, not distinguished names, in order to be rename and delete safe.</p> <p>For more information, see Using objectGUID to Bind to an Object.</p>

How Tuple Indexing Works

6/3/2022 • 2 minutes to read • [Edit Online](#)

Tuple indices are used to optimize searches that have 0 or more [medial-search strings](#) and 0 or 1 final-search strings. They can also be used to optimize searches for an initial search string if no ordinary index is available over that attribute.

You can turn on tuple indexing for an attribute by setting bit 5, which corresponds to the value 32, in the [searchFlags](#) attribute. This attribute is set in the schema object that represents the attribute that needs the tuple index. The performance impact of turning on tuple indexing is that any string value that is set for that attribute will be expanded into a large number of fragments in the tuple index. When an attribute expands, it consumes a larger amount of disk space in the Directory Information Tree file, and also gets updated more slowly.

Tuple indices are designed to accelerate searches of the form `*string*`. The acceleration can be considerable because this form of search cannot be optimized in any other way, and in its un-optimized form, it forces the Active Directory server to walk every object in the scope of the search to perform the query. Thus, a base search would just search one object, which would use fewer resources, an immediate children search would search just the children of an object (which could use fewer resources or more resources depending on the container size), and a subtree search will walk the entire subtree under the base object, which would usually require a lot of resources and be very slow because of the subtree size.

Tuple indices work by breaking a string into *tuples*. For example, the string "Active Directory" would be broken into the following tuples:

- "Active Dir"
- "ctive Dire"
- "tive Direc"
- "ive Direct"
- "ve Directo"
- "e Director"
- " Directory"
- "Directory"
- "irectory"
- "rectory"
- "ectomy"
- "ctory"
- "tory"
- "ory"

NOTE

The directory will stop at 32767 characters when expanding a string for tuple indexing.

A tuple index would contain an entry for each one of these tuples. So, if a user searches for `*cto*`, then the Active Directory server will look up all the matches for "cto" in the index and, in this case, find a pointer back to

the record that had a (tuple indexed) attribute with a value of "Directory".

If the medial search string (`*cto*` in the previous example) is specific enough, then the search will be quite efficient because it greatly reduces the number of objects that the Active Directory server must inspect to perform the query.

Initial, Medial, and Final Search Strings

6/3/2022 • 2 minutes to read • [Edit Online](#)

There are three types of wildcard searches that are referenced throughout the documentation about construction search string queries. These wildcard searches are: initial-, medial-, and final-search strings. This topics describes these searches.

Initial Search Strings

Initial-search strings match a given set of characters at the beginning of a string, followed by a wildcard. For example, the initial-search string `Act*` would match "Active Directory".

Medial-Search Strings

Medial-search strings match a given set of characters in the middle of a string, preceded by and/or followed by a wildcard. The medial-search strings `*Dir*`, `*ive*Dir*`, and `*ve*Dir*tor*` would all match "Active Directory".

Final-search Strings

Final-search strings match a given set of characters at the end of a string, preceded by a wildcard. For example, the final-search string `*ory` would match "Active Directory".

Referrals (AD DS)

6/3/2022 • 3 minutes to read • [Edit Online](#)

Active Directory Domain Services maintain referral data in **crossRef** objects stored in the partitions container (**crossRefContainer**) in the configuration container. In the [Deciding Where to Search](#) topic, referrals are discussed in the context of a domain within a domain tree and the generation of referrals to subordinate domains on a subtree search.

Active Directory Domain Services create and maintains **crossRef** objects for all domains in the forest. In addition, there are **crossRef** objects for the configuration and schema containers. These **crossRef** objects are used to generate referrals in response to queries that request data about objects that exist in the forest, but not contained on the directory server handling the request. These are called *internal cross references*, because they refer to domains, schema, and configuration containers within the forest.

If referral chasing is not enabled and a subtree search is performed, the search will return all objects within the specified domain that meet the search criteria. The search will also return referrals to any subordinate domains that are direct descendants of the directory server domain. The client must resolve the referrals by binding to the path specified by the referral and submitting another query.

If referral chasing is turned on and a subtree search is performed, the underlying LDAP API will automatically attempt to bind to any referrals and add the search results to the result set. In most cases, the referral chasing will be transparent to the caller. If the referral is to an object in a different domain or forest, the underlying LDAP API will attempt to use the current credentials to bind to the target of the referral. If this is successful, the referral chasing will be transparent. If this is not successful, the referral and a referral error code will be returned.

For more information about setting the referral chasing search preference, see [Specifying Other Search Options](#).

In addition to the **dnsRoot** (DNS name of the domain) and **nCName** (distinguished name for the domain) properties, the **crossRef** object also contains the **nETBIOSName** (NetBIOS name of the domain) and **trustParent** (distinguished name for the **crossRef** object that represents the domain's direct parent domain) properties.

Active Directory Domain Services can also have *external cross references* that refer to objects outside of the forest. External cross references must be added explicitly by an administrator. Be aware that the target server of the external cross reference must have a DNS root; that is, it must adhere to RFC 2247.

An external cross reference refers to an object on any other forest, as long as the name is unique in the target forest. For example, LDAP client applications that are used by your company can submit operations to your directory servers for Fabrikam.com. You create a **crossRef** object for Fabrikam.com. Now, instead of returning an Object Not Found error, your directory servers can return a referral to an object in the Fabrikam.com domain and the clients can chase the referral. For example, if you have an object with the following distinguished name:

```
CN=SomeObject,OU=SomeOU,DC=Fabrikam,DC=Com
```

You can add an external cross reference for an object with the name "ChildOfSomeObject".

```
CN=ChildOfSomeObject,CN=SomeObject,OU=SomeOU,DC=Fabrikam,DC=Com
```

A subtree search that contains "SomeObject" will also return a referral to "ChildOfSomeObject". Be aware that there must be an LDAP server at the address that is specified by the referral (one of the properties on the **crossRef** object) and that this LDAP server must serve the namespace that is identified by

"ChildOfSomeObject".

Because **crossRef** objects are stored in the configuration container, each domain controller (DC) has a copy of all **crossRef** objects. Therefore, every DC contains data about every domain in the forest as well as their superior/subordinate relationships. This enables each DC to generate referrals to any domain in the forest and referrals for unexplored subordinate domain, schema, or configuration containers on a subtree search.

For more information about referrals, see:

- [When Referrals Are Generated](#)
- [Creating an External Referral](#)

For more information and a code example that shows how to obtain the distinguished name of the partitions container, see [Example Code for Locating the Partitions Container](#).

When Referrals Are Generated

6/3/2022 • 2 minutes to read • [Edit Online](#)

A referral is the way that a directory server communicates that it does not contain the data required to complete a query, but has a reference to a server that may contain the required data. Be aware that referrals are not just generated by query requests.

The following operations can result in one or more referrals:

- Binding to a server that does not contain the object specified by the requested distinguished name but has data about a server or domain that may contain that object. For ADSI, this can occur if the application calls [ADsGetObject](#) or [ADsOpenObject](#) to bind to an object that exists in another domain in the forest (internal referral) or a naming context that is completely separate from the forest (external referral). For the LDAP API, this can occur when performing add, modify, delete, or search operations that specify an object that exists in another domain in the forest (internal referral) or a naming context that is completely separate from the forest (external referral).

If name resolution fails to find an object locally and there are no **crossRef** objects for that portion of the namespace, the domain controller will attempt to construct an external referral based on the domain components of the distinguished name. For example, if a search was based at "CN=a,CN=b,DC=c,DC=d,DC=e", the domain controller will construct a referral to the LDAP server at DNS address "c.d.e".

All Windows 2000 domain controllers (which support only DC= naming for the upper components) recognize each other, and no external cross references are required for a client to bind from one forest to another. If other non-Windows 2000 directory servers, such as a Netscape server, is using DC= naming and has an appropriate SRV RR registered in DNS, it will get the advantage of the automatic referrals as well. If not, an external **crossRef** object must be added manually.

- Performing a subtree search on a domain that contains subordinate domains in the forest or subordinate external domains, schema, or configuration containers. A referral to the subordinate domain, external domain, schema or configuration container will be created. If referral chasing is enabled, the referral will be transparent to the caller.

Creating an External Referral

6/3/2022 • 2 minutes to read • [Edit Online](#)

If an external **crossRef** object is created and a domain controller uses it to generate a referral, the **crossRef** object provides two important data elements in the following properties. For more information about situations where this can occur, see the previous section.

PROPERTY	DESCRIPTION
dnsRoot	Specifies the server or domain that can serve data from the naming context specified in nCName .
nCName	Specifies the distinguished name for the domain, schema, or configuration container rooted at the server or domain specified by dnsRoot .

For example, if the server with DNS address of serv1.northwest.Fabrikam.com serves the naming context rooted at CN=MyContainer,OU=MyDOM,O=Fabrikam, set the **dnsRoot** to that server's DNS address and the **nCName** to the distinguished name of the domain, schema, or configuration container.

For more information and a code example that shows how to create an external referral, see [Example Code for Creating an External **crossRef** Object](#).

Example Code for Creating an External crossRef Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following Visual Basic code example shows how to create an external **crossRef** object.

```
' CreateCrossRef()
'
' Description: Creates a crossRef object in the partitions container.
'
' Parameters:
'
' CrossRefName - Contains the name of the crossRef object.
'
' CrossRefDNSRoot - Contains the value to be set for the dNSRoot
' attribute of the crossRef object.
'
' CrossRefNCName - Contains the value to be set for the nCName
' attribute of the crossRef object.
'

Sub CreateCrossRef(CrossRefName, _
                   CrossRefDNSRoot, _
                   CrossRefNCName)
    Dim oRootDSE As IADs
    Dim oPartitions As IADsContainer
    Dim ADsPath As String
    Dim oCrossRef As IADs

    ' Get the configurationNamingContext property from the
    ' RootDSE object.
    Set oRootDSE = GetObject("LDAP://RootDSE")
    ADsPath = "LDAP://CN=Partitions," +
              oRootDSE.Get("configurationNamingContext")

    ' Bind to the Partitions container.
    Set oPartitions = GetObject(ADsPath)

    ' Create the crossRef object.
    Set oCrossRef = oPartitions.Create("crossRef",
                                       "CN=" & CrossRefName)

    ' Set the dNSRoot attribute.
    oCrossRef.Put "dNSRoot", CrossRefDNSRoot

    ' Set the nCName attribute.
    oCrossRef.Put "nCName", CrossRefNCName

    ' Commit the crossRef object to the directory.
    oCrossRef.SetInfo
End Sub
```

Enumerating Domain Controllers

6/3/2022 • 3 minutes to read • [Edit Online](#)

In earlier versions of Windows, an application could only obtain a single domain controller in a domain by calling [DsGetDcName](#). There was no way to predict which domain controller would be retrieved or to obtain a list of the domain controllers. Windows enables an application to enumerate the domain controllers in a domain by using the [DsGetDcOpen](#), [DsGetDcNext](#), and [DsGetDcClose](#) functions.

To enumerate a domain controller, call [DsGetDcOpen](#). This function takes parameters that define the domain to enumerate and other enumeration options. [DsGetDcOpen](#) provides a domain enumeration context handle that is used to identify the enumeration operation when [DsGetDcNext](#) and [DsGetDcClose](#) are called.

The [DsGetDcNext](#) function is called with the domain enumeration context handle to retrieve the next domain controller in the enumeration. The first time this function is called, the first domain controller in the enumeration is retrieved. The second time this function is called, the second domain controller in the enumeration is retrieved. This process is repeated until [DsGetDcNext](#) returns `ERROR_NO_MORE_ITEMS`, that indicates the end of the enumeration.

The [DsGetDcNext](#) function will enumerate the domain controllers in two groups. The first group contains the domain controllers that cover the site of the computer where the function is executed and the second group contains the domain controllers that do not cover the site of the computer where the function is executed. If the `DS_NOTIFY_AFTER_SITE_RECORDS` flag is specified in the *OptionFlags* parameter in [DsGetDcOpen](#), the [DsGetDcNext](#) function will return `ERROR_FILEMARK_DETECTED` after all of the site-specific domain controllers have been retrieved. [DsGetDcNext](#) will then begin enumerating the second group, which contains all domain controllers in the domain, including the site-specific domain controllers contained in the first group.

Domain controllers that handle the site of the computer where the function is executed are enumerated first followed by the domain controllers that do not cover the site of the computer where the function is executed. A domain controller is said to cover a site if the domain controller is configured to reside in that site or if the domain controller resides in a site that is nearest to the site in question in terms of the configured inter-site link cost. If there are any domain controllers in both the group of domain controllers that cover and the group of domain controllers that do not cover the computer site, domain controllers are returned within the group in order of their configured priorities and weights that are specified in DNS. Domain controllers that have lower numeric priority are returned within a group first. If within a site-related group there is a subgroup of several domain controllers with the same priority, domain controllers are returned in a weighted random order where domain controllers with higher weight have more probability to be returned first. The sites, priorities, and weights are configured by the domain administrator to achieve effective performance and load balancing among multiple domain controllers available in the domain. Because of this, applications that use the [DsGetDcOpen](#)/[DsGetDcNext](#)/[DsGetDcClose](#) functions automatically take advantage of these optimizations.

When the enumeration is complete or is no longer required, the enumeration must be closed by calling [DsGetDcClose](#) with the domain enumeration context handle.

To reset the enumeration, it is necessary to close the current enumeration by calling [DsGetDcClose](#) and then reopen the enumeration by calling [DsGetDcOpen](#) again.

Example

The following code example shows how to use these functions to enumerate the domain controllers in the local domain.

```
DWORD dwRet;
PDOMAIN_CONTROLLER_INFO pdcInfo;

// Get a domain controller for the domain this computer is on.
dwRet = DsGetDcName(NULL, NULL, NULL, NULL, 0, &pdcInfo);
if(ERROR_SUCCESS == dwRet)
{
    HANDLE hGetDc;

    // Open the enumeration.
    dwRet = DsGetDcOpen(    pdcInfo->DomainName,
                          DS_NOTIFY_AFTER_SITE_RECORDS,
                          NULL,
                          NULL,
                          NULL,
                          0,
                          &hGetDc);
    if(ERROR_SUCCESS == dwRet)
    {
        LPTSTR pszDnsHostName;

        /*
        Enumerate each domain controller and print its name to the
        debug window.
        */
        while(TRUE)
        {
            ULONG ulSocketCount;
            LPSOCKET_ADDRESS rgSocketAddresses;

            dwRet = DsGetDcNext(
                hGetDc,
                &ulSocketCount,
                &rgSocketAddresses,
                &pszDnsHostName);

            if(ERROR_SUCCESS == dwRet)
            {
                OutputDebugString(pszDnsHostName);
                OutputDebugString(TEXT("\n"));

                // Free the allocated string.
                NetApiBufferFree(pszDnsHostName);

                // Free the socket address array.
                LocalFree(rgSocketAddresses);
            }
            else if(ERROR_NO_MORE_ITEMS == dwRet)
            {
                // The end of the list has been reached.
                break;
            }
            else if(ERROR_FILEMARK_DETECTED == dwRet)
            {
                /*
                DS_NOTIFY_AFTER_SITE_RECORDS was specified in
                DsGetDcOpen and the end of the site-specific
                records was reached.
                */
                OutputDebugString(
                    TEXT("End of site-specific domain controllers\n"));
                continue;
            }
            else
            {
                // Some other error occurred.
                break;
            }
        }
    }
}
```

```
// Close the enumeration.  
DsGetDcClose(hGetDc);  
}  
  
// Free the DOMAIN_CONTROLLER_INFO structure.  
NetApiBufferFree(pdcInfo);  
}
```

Creating and Deleting Objects in Active Directory Domain Services

6/3/2022 • 3 minutes to read • [Edit Online](#)

The procedure used to programmatically create and delete objects in Active Directory Domain Services is dependent upon the programming technology used. For more information about creating and deleting objects in Active Directory Domain Services with a specific programming technology, see the topics listed in the following table.

PROGRAMMING TECHNOLOGY	FOR MORE INFORMATION
Active Directory Service Interfaces	Creating and Deleting Objects
Lightweight Directory Access Protocol	Modifying a Directory Entry
System.DirectoryServices	Create, Delete, Rename and Move Objects

Creating an Object

In general, the only attributes required for an object to be created are the **cn** and **objectClass** attributes. Just creating an object does not necessarily make it a functional object however. Certain types of objects, such as users and groups, have additional required attributes to make them functional. For more information about creating specific types of objects, see [Creating a User](#) and [Creating Groups in a Domain](#).

Windows Server 2003: When an object of the **user**, **group**, or **computer** class is created on a domain controller that is running on Windows Server 2003 or later, the domain controller automatically sets the **sAMAccountName** attribute for the object to a unique string, if one is not specified.

Deleting an Object

The Active Directory server performs the following actions when an object is deleted:

- The **isDeleted** attribute of the deleted object is set to **TRUE**. Objects with an **isDeleted** attribute value set to **TRUE** are called *tombstones*.
- The deleted object is moved to the Deleted Objects container for its naming context. If the object **systemFlags** property contains the **0x02000000** flag, the object is not moved to the Deleted Objects container. For more information about binding to and enumerating the contents of the Deleted Objects container, see [Retrieving Deleted Objects](#).
- The Deleted Objects container is flat, so all objects reside at the same level within the Deleted Objects container. Thus, the relative distinguished name of the deleted object is changed to ensure that the name is unique within the Deleted Objects container. If the original name is longer than 75 characters, it is truncated to 75 characters. The following are then appended to the new name:
 1. A **0xA** character
 2. The string "**DEL:**"
 3. The string form of a unique GUID, such as "947e3228-70c9-4311-8b7a-e5c9b5bd4432"

An example of a deleted object name is:

```
Jeff Smith\0ADEL:947e3228-70c9-4311-8b7a-e5c9b5bd4432
```

- Most attribute values for the deleted object are removed. The following attributes are automatically retained:

- **attributeID**
- **attributeSyntax**
- **distinguishedName**
- **dNReferenceUpdate**
- **flatName**
- **governsID**
- **groupType**
- **instanceType**
- **IDAPDisplayName**
- **legacyExchangeDN**
- **mS-DS-CreatorSID**
- **mSMQOwnerID**
- **name**
- **nCName**
- **objectClass**
- **objectGUID**
- **objectSid**
- **oMSyntax**
- **proxiedObjectName**
- **replPropertyMetaData**
- **sAMAccountName**
- **securityIdentifier**
- **subClassOf**
- **systemFlags**
- **trustAttributes**
- **trustDirection**
- **trustPartner**
- **trustType**
- **userAccountControl**
- **uSNChanged**
- **uSNCreated**
- **whenCreated**

Other attributes that have a **searchFlags** attribute value that contains 0x00000008 are also retained.

The following attribute values are always removed from a deleted object:

- **objectCategory**
- **samAccountType**
- The security descriptor of the deleted object is retained and inheritable access control entries are not propagated. The security descriptor is retained as-is at the time the object is deleted.
- Links to and from the deleted object are cleared. This is performed in the background after the object is

deleted. If the deleted object is restored before all of the links are cleared, an error will be received.

- If the object is deleted on a Windows Server 2003 domain controller, the **lastKnownParent** attribute of the deleted object is set to the distinguished name of the container where the object was contained when it was deleted.

The deleted object remains in the Deleted Objects container for a period of time known as the *tombstone lifetime*. By default, the tombstone lifetime is 60 days, but this value can be changed by the system administrator. After the tombstone lifetime expires, the object is permanently removed from the Directory Service. To avoid missing a delete operation, an application must perform incremental synchronizations more frequently than the tombstone lifetime.

Windows Server 2003 adds the ability to restore deleted objects. For more information about deleted object restoration, see [Restoring Deleted Objects](#).

When an item is deleted, none of the attributes of the object can be modified. In Windows Server 2003, it is possible to modify the security descriptor (the **ntSecurityDescriptor** attribute) on a deleted object. This is to allow restoration of objects when the person restoring the object does not have write permissions to mandatory attributes. To update the security descriptor on a deleted object, the caller must have the "Reanimate Tombstone" control access right on the naming context, in addition to regular **WRITE_DAC** and **WRITE_OWNER** access. Even if the security descriptor is restrictive, the administrator can first take ownership of the object, assuming the administrator has the **SE_TAKE_OWNERSHIP_NAME** privilege, and then modify the security descriptor. To do this, use the **ldap_modify_ext_s** function with the **LDAP_SERVER_SHOW_DELETED_OID** control. The modification list must contain a single attribute replacement for the **ntSecurityDescriptor** attribute.

Retrieving Deleted Objects

6/3/2022 • 6 minutes to read • [Edit Online](#)

Deleted objects are stored in the Deleted Objects container. The Deleted Objects container is not normally visible, but the Deleted Objects container can be bound to by a member of the administrators group. The contents of the Deleted Objects container can be enumerated and individual deleted object attributes can be obtained using the [IDirectorySearch](#) interface with the `ADS_SEARCHPREF_TOMBSTONE` search preference.

The Deleted Objects container can be obtained by binding to the well-known GUID `GUID_DELETED_OBJECTS_CONTAINER` defined in Ntdsapi.h. For more information about binding to well-known GUIDs, see [Binding to Well-Known Objects Using WKGUID](#).

Specify the `ADS_FAST_BIND` option when binding to the Deleted Objects container. This means that the ADSI interfaces used to work with an object in Active Directory Domain Services, such as [IADs](#) and [IADsPropertyList](#), cannot be used on the Deleted Objects container. For more information and a code example that shows how to bind to the Deleted Objects container, see the `GetDeletedObjectsContainer` example function below.

- [Enumerating Deleted Objects](#)
- [Finding a Specific Deleted Object](#)
 - [GetDeletedObjectsContainer](#)
 - [EnumDeletedObjects](#)
 - [FindDeletedObjectByGUID](#)

Enumerating Deleted Objects

The [IDirectorySearch](#) interface is used to search for deleted objects.

To enumerate deleted objects

1. Obtain the [IDirectorySearch](#) interface for the Deleted Objects container. This is accomplished by binding to the Deleted Objects container and requesting the [IDirectorySearch](#) interface. For more information and a code example that shows how to bind to the Deleted Objects container, see the following `GetDeletedObjectsContainer` function example.
2. Set the `ADS_SEARCHPREF_SEARCH_SCOPE` search preference to `ADS_SCOPE_ONELEVEL` using the [IDirectorySearch::SetSearchPreference](#) method. The `ADS_SCOPE_SUBTREE` preference can also be used, but the Deleted Objects container is only one level, so using `ADS_SCOPE_SUBTREE` is redundant.
3. Set the `ADS_SEARCHPREF_TOMBSTONE` search preference to `TRUE`. This causes the search to include deleted objects.
4. Set the `ADS_SEARCHPREF_PAGESIZE` search preference to a value less than, or equal to, 1000. This is optional, but if this is not done, no more than 1000 deleted objects can be retrieved.
5. Set the search filter in the [IDirectorySearch::ExecuteSearch](#) call to "`(isDeleted=TRUE)`". This causes the search to only retrieve objects with the `isDeleted` attribute set to `TRUE`.

For a code example code that shows how to enumerate deleted objects, see the following `EnumDeletedObjects` function example.

The search can be refined further by adding to the search filter as shown in [LDAP Dialect](#). For example, to search for all of the deleted objects with a name that begins with "Jeff", the search filter would be set to "`(&(isDeleted=TRUE)(cn=Jeff*))`".

Because deleted objects have most of their attributes removed when they are deleted, it is not possible to bind directly to a deleted object. The **ADS_FAST_BIND** option must be specified when binding to a deleted object. This means that the ADSI interfaces used to work with an Active Directory Domain Services object, such as **IADs** and **IADsPropertyList**, cannot be used on a deleted object container.

Finding a Specific Deleted Object

It is also possible to find a specific deleted object. If the **objectGUID** of the object is known, it can be used to search for the object with that specific **objectGUID**. For more information and a code example that shows how to find a specific deleted object, see **FindDeletedObjectByGUID** below.

GetDeletedObjectsContainer

The following C++ code example shows how to bind to the Deleted Objects container.

```
//*****
//  GetDeletedObjectsContainer()
//
//  Binds to the Deleted Object container.
//
//*****  
  
HRESULT GetDeletedObjectsContainer(IADsContainer **ppContainer)  
{  
    if(NULL == ppContainer)  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr;  
    IADs *pRoot;  
  
    *ppContainer = NULL;  
  
    // Bind to the rootDSE object.  
    hr = ADsOpenObject(L"LDAP://rootDSE",  
                      NULL,  
                      NULL,  
                      ADS_SECURE_AUTHENTICATION,  
                      IID_IADs,  
                      (LPVOID*)&pRoot);  
    if(SUCCEEDED(hr))  
    {  
        VARIANT var;  
  
        VariantInit(&var);  
  
        // Get the current domain DN.  
        hr = pRoot->Get(CComBSTR("defaultNamingContext"), &var);  
        if(SUCCEEDED(hr))  
        {  
            // Build the binding string.  
            LPWSTR pwszFormat = L"LDAP://<WKGUID=%s,%s>";  
            LPWSTR pwszPath;  
  
            pwszPath = new WCHAR[wcslen(pwszFormat) + wcslen(GUID_DELETED_OBJECTS_CONTAINER_W) +  
                           wcslen(var.bstrVal)];  
            if(NULL != pwszPath)  
            {  
                swprintf_s(pwszPath, pwszFormat, GUID_DELETED_OBJECTS_CONTAINER_W, var.bstrVal);  
  
                // Bind to the object.  
                hr = ADsOpenObject(pwszPath,  
                                  NULL,  
                                  NULL);  
            }  
        }  
    }  
}
```

```

        NULL,
        ADS_FAST_BIND | ADS_SECURE_AUTHENTICATION,
        IID_IADsContainer,
        (LPVOID*)ppContainer);

    delete pwszPath;
}
else
{
    hr = E_OUTOFMEMORY;
}

VariantClear(&var);
}

pRoot->Release();
}

return hr;
}

```

EnumDeletedObjects

The following C++ code example shows how to enumerate the objects in the Deleted Objects container.

```

//*****
//  EnumDeletedObjects()
//
//  Enumerates all of the objects in the Deleted Objects container.
//
//*****

HRESULT EnumDeletedObjects()
{
    HRESULT hr;
    IADsContainer *pDeletedObjectsCont = NULL;
    IDirectorySearch *pSearch = NULL;

    hr = GetDeletedObjectsContainer(&pDeletedObjectsCont);
    if(FAILED(hr))
    {
        goto cleanup;
    }

    hr = pDeletedObjectsCont->QueryInterface(IID_IDirectorySearch, (LPVOID*)&pSearch);
    if(FAILED(hr))
    {
        goto cleanup;
    }

    ADS_SEARCH_HANDLE hSearch;

    // Only search for direct child objects of the container.
    ADS_SEARCHPREF_INFO rgSearchPrefs[3];
    rgSearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    rgSearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
    rgSearchPrefs[0].vValue.Integer = ADS_SCOPE_ONELEVEL;

    // Search for deleted objects.
    rgSearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_TOMBSTONE;
    rgSearchPrefs[1].vValue.dwType = ADSTYPE_BOOLEAN;
    rgSearchPrefs[1].vValue.Boolean = TRUE;

    // Set the page size.
    rgSearchPrefs[2].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;
    rgSearchPrefs[2].vValue.dwType = ADSTYPE_INTEGER;

```

```

rgSearchPrefs[2].vValue.Integer = 1000;

// Set the search preference
hr = pSearch->SetSearchPreference(rgSearchPrefs, ARRAYSIZE(rgSearchPrefs));
if(FAILED(hr))
{
    goto cleanup;
}

// Set the search filter.
LPWSTR pszSearch = L"(cn=*)";

// Set the attributes to retrieve.
LPWSTR rgAttributes[] = {L"cn", L"distinguishedName", L"lastKnownParent"};

// Execute the search
hr = pSearch->ExecuteSearch(    pszSearch,
                                rgAttributes,
                                ARRAYSIZE(rgAttributes),
                                &hSearch);
if(SUCCEEDED(hr))
{
    // Call IDirectorySearch::GetNextRow() to retrieve the next row of data
    while(S_OK == (hr = pSearch->GetNextRow(hSearch)))
    {
        ADS_SEARCH_COLUMN col;
        UINT i;

        // Enumerate the retrieved attributes.
        for(i = 0; i < ARRAYSIZE(rgAttributes); i++)
        {
            hr = pSearch->GetColumn(hSearch, rgAttributes[i], &col);
            if(SUCCEEDED(hr))
            {
                switch(col.dwADsType)
                {
                    case ADSTYPE_CASE_IGNORE_STRING:
                    case ADSTYPE_DN_STRING:
                    case ADSTYPE_PRINTABLE_STRING:
                    case ADSTYPE_NUMERIC_STRING:
                    case ADSTYPE_OCTET_STRING:
                        wprintf(L"%s: ", rgAttributes[i]);
                        for(DWORD x = 0; x < col.dwNumValues; x++)
                        {
                            wprintf(col.pADsValues[x].CaseIgnoreString);
                            if((x + 1) < col.dwNumValues)
                            {
                                wprintf(L",");
                            }
                        }
                        wprintf(L"\n");
                        break;
                }

                pSearch->FreeColumn(&col);
            }
        }

        wprintf(L"\n");
    }

    // Close the search handle to cleanup.
    pSearch->CloseSearchHandle(hSearch);
}

cleanup:

if(pDeletedObjectsCont)
{

```

```

        pDeletedObjectsCont->Release();
    }

    if(pSearch)
    {
        pSearch->Release();
    }

    return hr;
}

```

FindDeletedObjectByGUID

The following C++ code example shows how to find a specific deleted object from the object's **objectGUID** property.

```

HRESULT FindDeletedObjectByGUID(IADS *padsDomain, GUID *pguid)
{
    HRESULT hr;
    IDirectorySearch *pSearch = NULL;
    LPWSTR pwszGuid = NULL;

    hr = padsDomain->QueryInterface(IID_IDirectorySearch, (LPVOID*)&pSearch);
    if(FAILED(hr))
    {
        goto cleanup;
    }

    ADS_SEARCH_HANDLE hSearch;

    // Search the entire tree.
    ADS_SEARCHPREF_INFO rgSearchPrefs[2];
    rgSearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    rgSearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
    rgSearchPrefs[0].vValue.Integer = ADS_SCOPE_SUBTREE;

    // Search for deleted objects.
    rgSearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_TOMBSTONE;
    rgSearchPrefs[1].vValue.dwType = ADSTYPE_BOOLEAN;
    rgSearchPrefs[1].vValue.Boolean = TRUE;

    // Set the search preference.
    hr = pSearch->SetSearchPreference(rgSearchPrefs, 2);
    if(FAILED(hr))
    {
        goto cleanup;
    }

    // Set the search filter.
    hr = ADsEncodeBinaryData((LPBYTE)pguid, sizeof(GUID), &pwszGuid);
    if(FAILED(hr))
    {
        goto cleanup;
    }

    LPWSTR pwszFormat = L"(objectGUID=%s)";

    LPWSTR pwszSearch = new WCHAR[lstrlenW(pwszFormat) + lstrlenW(pwszGuid) + 1];
    if(NULL == pwszSearch)
    {
        goto cleanup;
    }

    swprintf_s(pwszSearch, pwszFormat, pwszGuid);

    FreeADsMem(pwszGuid);
}

```

```

        .....
```

```

// Set the attributes to retrieve.
LPWSTR rgAttributes[] = {L"cn", L"distinguishedName", L"lastKnownParent"};

// Execute the search.
hr = pSearch->ExecuteSearch(    pwszSearch,
                                    rgAttributes,
                                    3,
                                    &hSearch);
if(SUCCEEDED(hr))
{
    // Call IDirectorySearch::GetNextRow() to retrieve the next row of data.
    while(S_OK == (hr = pSearch->GetNextRow(hSearch)))
    {
        ADS_SEARCH_COLUMN col;
        UINT i;

        // Enumerate the retrieved attributes.
        for(i = 0; i < ARRAYSIZE(rgAttributes); i++)
        {
            hr = pSearch->GetColumn(hSearch, rgAttributes[i], &col);
            if(SUCCEEDED(hr))
            {
                switch(col.dwADsType)
                {
                    case ADSTYPE_CASE_IGNORE_STRING:
                    case ADSTYPE_DN_STRING:
                    case ADSTYPE_PRINTABLE_STRING:
                    case ADSTYPE_NUMERIC_STRING:
                        wprintf(L"%s: ", rgAttributes[i]);
                        for(DWORD x = 0; x < col.dwNumValues; x++)
                        {
                            wprintf(col.pADsValues[x].CaseIgnoreString);
                            if((x + 1) < col.dwNumValues)
                            {
                                wprintf(L",");
                            }
                        }
                        wprintf(L"\n");
                        break;

                    case ADSTYPE_OCTET_STRING:
                        wprintf(L"%s: ", rgAttributes[i]);
                        for(DWORD x = 0; x < col.dwNumValues; x++)
                        {
                            GUID guid;
                            LPBYTE pb = col.pADsValues[x].OctetString.lpValue;
                            WCHAR wszGUID[MAX_PATH];

                            // Convert the octet string into a GUID.
                            guid.Data1 = *((long*)pb);
                            pb += sizeof(guid.Data1);
                            guid.Data2 = *((short*)pb);
                            pb += sizeof(guid.Data2);
                            guid.Data3 = *((short*)pb);
                            pb += sizeof(guid.Data3);
                            CopyMemory(&guid.Data4, pb, sizeof(guid.Data4));

                            // Convert the GUID into a string.
                            StringFromGUID2(guid, wszGUID, MAX_PATH);
                            wprintf(wszGUID);

                            if((x + 1) < col.dwNumValues)
                            {
                                wprintf(L",");
                            }
                            OutputDebugStringW(wszGUID);
                            OutputDebugStringW(L"\n");
                        }
                }
            }
        }
    }
}

```

```

    {
        DWORD a;

        for(a = 0, *wszGUID = 0; a < col.pADsValues[x].OctetString.dwLength;
a++)
    {
        swprintf_s(wszGUID + lstrlenW(wszGUID), L"%02X", *(LPBYTE)
(col.pADsValues[x].OctetString.lpValue + a));
    }

        OutputDebugStringW(wszGUID);
        OutputDebugStringW(L"\n");
    }
}

wprintf(L"\n");
}

// Close the search handle to cleanup.
pSearch->CloseSearchHandle(hSearch);
}

cleanup:

if(pwszSearch)
{
    delete pwszSearch;
}

if(pSearch)
{
    pSearch->Release();
}

return hr;
}

```

Restoring Deleted Objects

6/3/2022 • 6 minutes to read • [Edit Online](#)

Windows Server 2003 includes the "restore deleted objects" feature.

To enable deleted object restoration, at least one domain controller in the domain must be running on Windows Server 2003 or a later version of Windows. By default, only domain administrators can restore deleted objects, though this can be delegated to others.

The following limitations apply to restoring deleted objects:

- An object cannot be restored when the tombstone lifetime for the object has expired because when the tombstone lifetime has expired, the object is permanently deleted.
- Objects that exist at the root of the naming context, such as a domain or application partition, cannot be restored.
- Schema objects cannot be restored. Schema objects should never be deleted.
- It is possible to restore deleted containers, but the restoration of the deleted objects that were in the container before the deletion is difficult because the tree structure under the container must be manually reconstructed.

Permissions Required to Restore a Deleted Object

When an object is deleted, the object security descriptor is retained. Although the owner is identifiable from the security descriptor, for security reasons, only domain administrators are allowed to restore deleted objects.

Domain administrators can grant the permission to restore delete objects to other users and groups by granting the user or group the "Reanimate Tombstone" control access right. The "Reanimate Tombstone" control access right is granted at the Naming Context root. Only users that had read access permission on an object and its attributes are permitted to read the object and accessible attributes after the object is deleted.

NOTE

Granting a user this permission can be a security risk because it could permit the user to restore an account object that has access to resources that the user would not normally have access to. By restoring an account, the user essentially gains control of this account because the user must set the initial password on the account when the account is restored.

To completely restore a deleted object, the user must:

- Have, or be a member of a group that has, the "Reanimate Tombstone" control access right.
- Have write access for each mandatory attribute that requires updating.
- Have write access to the Relative Distinguished Name (RDN).
- Have write access to each optional attribute that needs to be updated.
- Have child-creation rights on the destination container for the object class of restored object.

NOTE

The **isDeleted** attribute is not verified during the restore operation. Neither will the delete-child permission on the "Deleted Objects" container be verified.

Restoring a Deleted Object

To restore a deleted object, the object must first be located in the Deleted Objects container. For more information about retrieving deleted objects, see [Retrieving Deleted Objects](#).

When the object has been located, the following operations must be completed in a single LDAP operation. This requires the use of the **ldap_modify_ext_s** function with the **LDAP_SERVER_SHOW_DELETED_OID** control.

- Remove the **isDeleted** attribute value. The **isDeleted** attribute value must be removed, not set to **FALSE**.
- Replace the distinguished name of the object so that it is moved to a container other than the Deleted Objects container. This can be any container that can normally contain the object. The distinguished name of the previous container of the object can be found in the deleted object's **lastKnownParent** attribute. The **lastKnownParent** attribute is only set if the object was deleted on a Windows Server 2003 domain controller. Thus, it is possible that the content of the **lastKnownParent** attribute is inaccurate.
- Restore the mandatory attributes for the object that were cleared during deletion.

NOTE

The **objectCategory** attribute can also be set when the object is restored, but is not required. If no **objectCategory** value is specified, the default **objectCategory** for the object's **objectClass** is used.

After the object is restored, it can be accessed as it was before it was deleted. At this point, any optional attributes that are important should be restored. Any references to the object from other objects in the directory must also be restored.

As a security measure, user objects are disabled when they are restored. User objects must be enabled after restoring the optional attributes to allow the user object to be used.

For more information and a code example that shows how to restore a deleted object, see the **RestoreDeletedObject** function below.

RestoreDeletedObject

The following C++ code example shows how to restore a deleted object.

```
*****  
//  
//  RestoreDeletedObject()  
//  
//  Restores a deleted object.  
//  
//  pwszDeletedDN - Contains the fully qualified distinguished name of the  
//  deleted object.  
//  
//  pwszDestContainerDN - Contains the fully qualified distinguished name of  
//  the folder that the deleted object should be moved to.  
//  
//  Returns S_OK if successful or an HRESULT or LDAP error code otherwise.
```

```

//  RESTORES A_DELETED OR ABANDONED OBJECT IN THE DESTINATION CONTAINER.
// ****
HRESULT RestoreDeletedObject(LPCWSTR pwszDeletedDN, LPCWSTR pwszDestContainerDN)
{
    if((NULL == pwszDeletedDN) || (NULL == pwszDestContainerDN))
    {
        return E_POINTER;
    }

    HRESULT hr = E_FAIL;

    // Build the new distinguished name.
    LPWSTR pwszNewDN = new WCHAR[lstrlenW(pwszDeletedDN) + lstrlenW(pwszDestContainerDN) + 1];
    if(pwszNewDN)
    {
        wcscpy_s(pwszNewDN, pwszDeletedDN);

        // Search for the first 0x0A character. This is the delimiter in the deleted object name.
        LPWSTR pwszChar;
        for(pwszChar = pwszNewDN; *pwszChar; pwszChar = CharNextW(pwszChar))
        {
            if('\\' == *pwszChar) && ('0' == *(pwszChar + 1)) && ('A' == *(pwszChar + 2)))
            {
                break;
            }
        }

        if(0 != *pwszChar)
        {
            // Truncate the name string at the delimiter.
            *pwszChar = 0;

            // Add the last known parent DN to complete the DN.
            wcscat_s(pwszNewDN, L",");
            wcscat_s(pwszNewDN, pwszDestContainerDN);

            PLDAP ld;

            // Initialize LDAP.
            ld = ldap_init(NULL, LDAP_PORT);
            if(NULL != ld)
            {
                ULONG ulRC;
                ULONG version = LDAP_VERSION3;

                // Set the LDAP version.
                ulRC = ldap_set_option(ld, LDAP_OPT_PROTOCOL_VERSION, (void*)&version);
                if(LDAP_SUCCESS == ulRC)
                {
                    // Establish a connection with the server.
                    ulRC = ldap_connect(ld, NULL);
                    if(LDAP_SUCCESS == ulRC)
                    {
                        // Bind to the LDAP server.
                        ulRC = ldap_bind_s(ld, NULL, NULL, LDAP_AUTH_NEGOTIATE);
                        if(LDAP_SUCCESS == ulRC)
                        {
                            // Setup the new values.
                            LPWSTR rgNewVals[] = {pwszNewDN, NULL};

                            /*
                             Remove the isDeleted attribute. This cannot be set
                             to FALSE or the restore operation will not work.
                             */
                            LDAPModW modIsDeleted = { LDAP_MOD_DELETE, L"isDeleted", NULL };

                            /*

```

```

        /*
         Set the new DN, in effect, moving the deleted
         object to where it resided before the deletion.
        */
        LDAPModW modDN = { LDAP_MOD_REPLACE, L"distinguishedName", rgNewVals };

        // Initialize the LDAPMod structure.
        PLDAPModW ldapMods[] =
        {
            &modIsDeleted,
            &modDN,
            NULL
        };

        /*
         Use the LDAP_SERVER_SHOW_DELETED_OID control to
         modify deleted objects.
        */
        LDAPControlW showDeletedControl;
        showDeletedControl.ldctl_oid = LDAP_SERVER_SHOW_DELETED_OID_W;
        showDeletedControl.ldctl_value.bv_len = 0;
        showDeletedControl.ldctl_value.bv_val = NULL;
        showDeletedControl.ldctl_iscritical = TRUE;

        // Initialize the LDAPControl structure
        PLDAPControlW ldapControls[] = { &showDeletedControl, NULL };

        /*
         Modify the specified attributes. This must performed
         in one step, which is why the LDAP APIs must be used
         to restore a deleted object.
        */
        ulRC = ldap_modify_ext_sw(ld, (PWCHAR)pwszDeletedDN, ldapMods, ldapControls,
NULL);
        if(LDAP_SUCCESS == ulRC)
        {
            hr = S_OK;
        }
        else if(LDAP_ALREADY_EXISTS == ulRC)
        {
            /*
             An object already exists with the specified name
             in the specified target container. At this point,
             a new name must be selected.
            */
        }
    }

    if(LDAP_SUCCESS != ulRC)
    {
        hr = ulRC;

        OutputDebugString(ldap_err2string(ulRC));
    }

    // Release the LDAP session.
    ldap_unbind(ld);
}
else
{
    /*
     If the end of the string is reached before the delimiter is found, just
     end and fail.
    */
    hr = E_INVALIDARG;
}

```

```
    delete pwszNewDN;
}
else
{
    hr = E_OUTOFMEMORY;
}

return hr;
}
```

Moving Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

To move an object within a domain, use the **IADsContainer.MoveHere** method.

To move an object within a domain

1. Bind to the **IADs** interface of the object to move.
2. Get the ADsPath of the object to move from the **IADs.ADsPath** property.
3. Bind to the **IADsContainer** interface of the container where the object will be moved to.
4. Move the object with the **IADsContainer.MoveHere** method.

If an interface exists to the object that was moved, the interface is not valid after the object is moved because the directory object the interface represents no longer exists.

For more information and a code example that shows how to move an object, see [Example Code for Moving an Object](#).

Example Code for Moving an Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes code examples used to move an object in the domain.

The following C++ code example shows how to use ADSI to move an object in the domain.

```
#include <stdio.h>
#include <atldbbase.h>
#include <activeds.h>
#include <ntldap.h>

//*****************************************************************************
MoveObject()

Moves an object in the directory.

Parameters:

padsToMove - Contains an IADs interface pointer that represents the
object to move.

padsDestination - Contains an IADs interface pointer that represents the
container to move the object to.

***** */

HRESULT MoveObject(IADs *padsToMove, IADs *padsDestination)
{
    _ASSERT(padsToMove);
    _ASSERT(padsDestination);

    HRESULT hr;

    // Get the ADsPath of the object to move.
    CComBSTR sbstrPathToMove;
    hr = padsToMove->get_ADsPath(&sbstrPathToMove);
    if(FAILED(hr))
    {
        return hr;
    }

    // Get an IADsContainer instance from the destination.
    CComPtr<IADsContainer> spContainerDestination;
    hr = padsDestination->QueryInterface(IID_IADsContainer, (void**)&spContainerDestination);
    if(FAILED(hr))
    {
        return hr;
    }

    // Move the object.
    CComPtr<IDispatch> spDispNewObject;
    hr = spContainerDestination->MoveHere(sbstrPathToMove, NULL, &spDispNewObject);

    return hr;
}
```

The following Visual Basic code example shows how to use ADSI to move an object in the domain.

```
'.....  
  
' MoveObject  
  
' Moves an object in the directory.  
  
' Parameters:  
  
' DNToMove - Contains the distinguished name of the object to move.  
  
' DNDDestination - Contains the distinguished name of the destination  
' container.  
  
'.....  
  
Public Sub MoveObject(DNToMove As String, DNDDestination As String)  
    Dim oContainer As IADsContainer  
  
    ' Bind to the destination container.  
    Set oContainer = GetObject("LDAP://" & DNDDestination)  
  
    oContainer.MoveHere "LDAP://" & DNToMove, vbNullString  
End Sub
```

Reading and Writing Attributes of Objects in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Objects in Active Directory Domain Services have attributes that contain data about the specific object. How the attributes are accessed and modified depends upon the programming technology being used. For more information about reading and modifying attributes in Active Directory Domain Services with a specific programming technology, see the following listed topics.

PROGRAMMING TECHNOLOGY	FOR MORE INFORMATION
Active Directory Service Interfaces	Accessing and Manipulating Data with ADSI
Lightweight Directory Access Protocol	Modifying a Directory Entry
System.DirectoryServices	Directory Object Properties

For more information about attributes, see [Determining an Attribute Type](#).

Determining an Attribute Type

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **systemFlags** attribute of an **attributeSchema** object contains a set of flags that indicate various qualities of the attribute object, such as whether the attribute is constructed or non-replicated. The following table lists the flags of the **systemFlags** attribute that affect the storage type of the attribute.

FLAG VALUE	DESCRIPTION
0x00000001	If this flag is present in the systemFlags attribute, the attribute is not replicated.
0x00000004	If this flag is present in the systemFlags attribute, the attribute is constructed.

It is possible to construct a query string that can be used to query for constructed or non-replicated attributes. For example, the following query string finds all non-replicated **attributeSchema** objects. Be aware that the query string requires the decimal equivalent of the value, not the hexadecimal equivalent of the value. For more information about the matching rule OID used by this query string, see [How to Specify Comparison Values](#).

```
(&(objectCategory=attributeSchema)(systemFlags:1.2.840.113556.1.4.804:=1))
```

The **searchFlags** attribute of each attribute's **attributeSchema** object defines whether an attribute is indexed; an indexed attribute has a value of 1, a non-indexed attribute has a value of 0. For example, the following query string finds the **attributeSchema** objects representing indexed attributes.

```
(&(objectCategory=attributeSchema)(searchFlags=1))
```

The **isMemberOfPartialAttributeSet** attribute of each attribute's **attributeSchema** object defines whether an attribute is replicated to the global catalog. This attribute has a value of **TRUE** if the attribute is a member of the global catalog or **FALSE** if the attribute is not in the global catalog. For example, the following query string searches the **attributeSchema** objects that are replicated to the global catalog.

```
(&(objectCategory=attributeSchema)(isMemberOfPartialAttributeSet=TRUE))
```

Controlling object access in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Each Active Directory directory service object is protected by Windows 2000 security. This security protection controls the operations that each security principal can perform in the directory. The following sections describe how a directory-enabled application can use the access control features in Active Directory.

- [How Access Control Works in Active Directory Domain Services](#)
- [How access control affects read operations, write operation, object creation and deletion.](#)
- [Using the IADs and IDirectoryObject interfaces to work with an object's security descriptor](#)
- [Modifying the access permissions on an object](#)
- [How security descriptors are set on new directory objects](#)
- [Creating a Security Descriptor for a New Directory Object](#)
- [Using inheritance of access permissions to enable administrative access to an entire subtree of the directory](#)
- [Creating, modifying, and reading the default security descriptor for an object class](#)
- [Creating, setting, and checking control access rights for operations that go beyond those covered by the predefined rights](#)
- [Using DsAddSidHistory](#)
- [Controlling Object Visibility](#)
- [Null DACLs and Empty DACLs](#)

How Access Control Works in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Access control for objects in Active Directory Domain Services is based on Windows NT and Windows 2000 access-control models. For more information and a detailed description of this model and its components such as security descriptors, access tokens, SIDs, ACLs, and ACEs, see [Access Control Model](#).

Access privileges for resources in Active Directory Domain Services are usually granted through the use of an access control entry (ACE). An ACE defines an access or audit permission on an object for a specific user or group. An access-control list (ACL) is the ordered collection of access control entries defined for an object. A security descriptor supports properties and methods that create and manage ACLs. For more information about security models, see [Security](#) or the *Windows 2000 Server Resource Kit*. (This resource may not be available in some languages and countries or regions.)

The basic outline of the security model is:

- Security descriptor. Each directory object has its own security descriptor that contains security data that protects the object. The security descriptor can contain a discretionary access-control list (DACL). A DACL contains a list of ACEs. Each ACE grants or denies a set of access rights to a user or group. The access rights correspond to the operations, such as reading and writing properties, that can be performed on the object.
- Security context. When a directory object is accessed, the application specifies the credentials of the security principal that is making the access attempt. When authenticated, these credentials determine the application's security context, which includes the group memberships and privileges associated with the security principal. For more information about security contexts, see [Security Contexts and Active Directory Domain Services](#).
- Access check. The system grants access to an object only if the object's security descriptor grants the necessary access rights to the security principal attempting the operation (or to groups to which the security principal belongs).

The following table lists ADSI interfaces used to manipulate the access control features of Active Directory Domain Services.

INTERFACE	DESCRIPTION
IADsSecurityDescriptor	Used to read and write security properties of a directory service object.
IADsAccessControlList	Used to manage and enumerate all ACEs for a directory service object.
IADsAccessControlEntry	Used to read and write ACE properties.

Controlling Access to Objects and Their Properties

6/3/2022 • 2 minutes to read • [Edit Online](#)

To control access to application objects, work with the object security descriptor, and specifically, with the discretionary access-control list (DACL) and its list of access-control entries (ACEs).

When an object is created, it receives a security descriptor. For more information, and a description of the rules that the system uses to create the DACL for a new object, see [How Security Descriptors are Set on New Directory Objects](#). These rules reveal that you can:

- Create a new security descriptor and attach it to the object at creation time. For more information, see [Creating a Security Descriptor for a New Directory Object](#).
- Apply an inheritable ACE, at any point in the directory hierarchy, such that an ACE is inherited by objects down the tree. An object can inherit an ACE from its parent container. For more information, see [Inheritance and Delegation of Administration](#).
- Specify an ACE in the default DACL in the schema, if you have the necessary access rights. Every object class definition in the schema includes a default security descriptor that can have a default DACL. For more information, see [Default Security Descriptor](#).

In addition the DACL of an existing object can be modified. You can:

- Replace the DACL with a new one.
- Read the existing DACL, modify it, and apply the modified DACL. For more information, see [Setting Access Rights on an Object](#).

The following list describes the most important function of an ACE in Active Directory Domain Services. With an ACE, you can:

- Control who can perform specific operations on an object.
- Control who has access to a specific property, or set of properties, of an object.
- Control who can create child objects in a container, including who can create a specific type of child object.
- Define private control-access rights for an object type and control who can perform the operations protected by the private rights.
- Apply an ACE to a container object at the root of a directory subtree, such that the protections can be inherited automatically by all child objects down the tree.
- Apply an ACE that is inherited automatically by a specific type of child object in a subtree.
- Create an ACE that grants rights to a security group, rather than to a single user.
- Apply an ACE to Group Policy Objects to control the accounts and computers affected by the policy.

Access Rights for Active Directory Domain Services Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

All directory objects use the same set of predefined access rights that correspond to the common directory operations. These rights are described in the [ADS_RIGHTS_ENUM](#) enumeration type.

In addition, you can define private access rights (control access rights) for operations that go beyond those covered by the predefined rights. See [Control Access Rights](#).

Security Contexts and Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

When an application binds to an Active Directory Domain Controller (DC), it does so in the security context of a security principal, which can be a user or an entity such as a computer or a system service. The security context is the user account that the system uses to enforce security when a thread attempts to access a securable object. This data includes the user security identifier (SID), group memberships, and privileges.

A user establishes a security context by presenting credentials for authentication. If the credentials are authenticated, the system produces an access token that identifies the group memberships and privileges associated with the user account. The system verifies your access token when you attempt to access a directory object. It compares the data in your access token to the accounts and groups granted or denied access by the object security descriptor.

Use the following methods to control the security context with which you bind to an Active Directory DC:

- Bind using the **ADS_SECURE_AUTHENTICATION** option with the [ADsOpenObject](#) function or [IADsOpenDSObject::OpenDSObject](#) method and explicitly specify a user name and password. The system authenticates these credentials and generates an access token that it uses for access verification for the duration of that binding. For more information, see [Authentication](#).
- Bind using the **ADS_SECURE_AUTHENTICATION** option, but without specifying credentials. If you are not impersonating a user, the system uses the primary security context of your application, that is, the security context of the user who started your application. In the case of a system service, this is the security context of the service account or the LocalSystem account.
- Impersonate a user and then bind with **ADS_SECURE_AUTHENTICATION**, but without specifying credentials. In this case, the system uses the security context of the client that is impersonated. For more information, see [Client Impersonation](#).
- Bind using [ADsOpenObject](#) or [IADsOpenDSObject::OpenDSObject](#) with the **ADS_NO_AUTHENTICATION** option. This method binds without authentication and results in "Everyone" as the security context. Only the LDAP provider supports this option.

If possible, bind without specifying credentials. That is, use the security context of the logged-on user or the impersonated client. This enables you to avoid caching credentials. If you must use alternate user credentials, prompt for the credentials, bind with them, but do not cache them. To use the same security context in multiple bind operations, you can specify the user name and password for the first bind operation and then specify only the user name to make subsequent binds. For more information about using this technique, see [Authentication](#).

Some security contexts are more powerful than others. For example, the LocalSystem account on a domain controller has complete access to Active Directory Domain Services, whereas a typical user has only limited access to a some of the objects in the directory. In general, your application should not run in a powerful security context, such as LocalSystem, when a less powerful security context is sufficient to perform the operations. This means that you may want to divide your application into separate components, each of which runs in a security context appropriate to the operations to perform. For example, your application setup can be divided as follows:

- Perform schema changes and extensions in the context of a user who is a member of the Schema Administrators group.
- Perform configuration container changes in the context of a user who is a member of the Enterprise Administrators group.

- Perform domain container changes in the context of a user who is a member of the Domain Administrators group.

How Security Affects Operations in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services use access control to grant or deny access to objects, properties, and operations based on the identity of the user performing the access attempt. When your application binds to the directory, it binds with specific user credentials. When authenticated, these credentials determine your application's security context. Regardless of whether the credentials are those of the logged-on user, a specified user, a service account, a computer account, or an unauthenticated user (Guest/Everyone), the Active Directory server verifies the user's right to access an object before any operation is performed on that object. The user may, or may not, have access to a particular object, its children, its properties, or operations on that object, which means that your application must handle the potential errors caused by denied access.

For more information about security contexts and the effects of access control on various operations, see:

- [Access Control and Read Operations](#)
- [Access Control and Write Operations](#)
- [Access Control and Object Creation](#)
- [Access Control and Object Deletion](#)

Access Control and Read Operations

6/3/2022 • 2 minutes to read • [Edit Online](#)

Security is an implicit filter for searching, enumerating containers, or reading properties. If you do not have the necessary access rights, attempts to list objects or read properties can fail with the following error codes even though the object or property exists:

- E_ADS_INVALID_DOMAIN_OBJECT
- E_ADS_PROPERTY_NOT_SUPPORTED
- E_ADS_PROPERTY_NOT_FOUND

Be aware that a caller with **ADS_RIGHT_ACTRL_DS_LIST** access to a container can enumerate the child objects in the container. But an attempt to access a child object can still fail with an error such as **E_ADS_UNKNOWN_OBJECT** if the caller does not have **ADS_RIGHT_ACTRL_DS_LIST_OBJECT** access to the child object.

The impact of security on read operations is not necessarily manifested as an error. For example, a search operation can succeed, but the search results do not include objects or properties to which the caller does not have access.

Access Control and Write Operations

6/3/2022 • 2 minutes to read • [Edit Online](#)

Property modifications fail if the caller does not have sufficient rights. For write operations that batch modifications to multiple properties, the entire operation fails if the caller does not have the necessary rights to a single one of the modified properties. For example, you can make multiple **IADs::Put** calls to set multiple properties on an object. However, when you call **IADs::SetInfo** to write the new data from the local cache to the directory, **SetInfo** will fail if the caller does not have write access to all the modified properties. Similarly, the **IDirectoryObject::SetObjectAttributes** method fails to set any properties if the caller does not have access to all the properties being set. So you should batch multiple modify operations only if you know that all modifications will succeed. To determine the attributes of a directory object that the caller has the ability to modify, read the object's **allowedAttributesEffective** attribute.

If the caller does not have sufficient rights to modify a property, the following return codes may be returned:

E_ADS_PROPERTY_NOT_SET E_ADS_PROPERTY_NOT_MODIFIED

Access Control and Object Creation

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory server will fail to create a child object if the caller does not have the **ADS_RIGHT_DS_CREATE_CHILD** for that object type on the parent container. To determine the types of child objects that the caller can create in a directory object, read the object's **allowedChildClassesEffective** attribute.

When you use the **IADsContainer::Create** method to create a child object, the object is not made persistent until **IADs::SetInfo** is called on the new object. Between the **Create** and **SetInfo** calls, the creating thread can put values into any of the new object's properties. After the **SetInfo** call, the creating thread does not necessarily have the access rights to set the new object's properties. To ensure that the caller has these rights, specify an explicit security descriptor during creation. The DACL should have an ACE that gives the caller the necessary access rights on the object.

For more information about access control and object creation, see [How Security Descriptors are Set on New Directory Objects](#).

Access Control and Object Deletion

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services enable you to delete an object if you have one of the following access rights:

- DELETE access to the object itself
- ADS_RIGHT_DS_DELETE_CHILD access for that object type on the parent container

Be aware that the system verifies the security descriptor for both the object and its parent before denying the deletion. This means that an ACE that explicitly denies DELETE access to a user has no effect if the user has DELETE_CHILD access on the parent. Similarly, an ACE that denies DELETE_CHILD access on the parent can be overridden if DELETE access is allowed on the object itself.

To perform a tree-delete operation, for example using the [IADsDeleteOps::DeleteObject](#) method, you must have ADS_RIGHT_DS_DELETE_TREE access to the object. If you have this access right, you can delete the object and any child objects regardless of the protections on the child objects. To delete a tree if you do not have ADS_RIGHT_DS_DELETE_TREE access, you must recursively traverse the tree, deleting each object individually. In this case, you must have the necessary DELETE or DELETE_CHILD access for each object in the tree.

WARNING

If users have ADS_RIGHT_DS_DELETE_TREE access for an object, this gives them the ability to delete a whole subtree, including all child objects. For this reason, you may consider revoking "Delete Subtree" access permission for all users on a parent container.

APIs for Working with Security Descriptors

6/3/2022 • 2 minutes to read • [Edit Online](#)

Each object in Active Directory Domain Services has an **nTSecurityDescriptor** attribute that contains the object security descriptor. There are two primary ways to read and manipulate a directory object security descriptor:

- Use the [IADs::Get](#) method to retrieve the security descriptor as an ADSI COM object with an [IADsSecurityDescriptor](#) interface. The [IADsSecurityDescriptor](#), [IADsAccessControlList](#), and [IADsAccessControlEntry](#) interfaces can be used to handle the security descriptor and its components (ACLs, ACEs, and so on). For more information and a code example, see [Using IADs to Get a Security Descriptor](#).
- Use the [IDirectoryObject::GetObjectAttributes](#) method to retrieve an object security descriptor as a pointer to a **SECURITY_DESCRIPTOR** structure. Use this pointer with a Win32 access-control function, such as [AccessCheck](#) or [BuildSecurityDescriptor](#). For more information and a code example, see [Using IDirectoryObject to Get a Security Descriptor](#).

The recommended technique, and the one used by most of the code examples in this guide, is to use the **IADs*** interfaces because they simplify handling security descriptors, ACLs, and ACEs. For Visual Basic programmers, the **IADs*** interfaces are the most efficient way to handle security descriptors.

The [IDirectoryObject](#) technique is useful when a **SECURITY_DESCRIPTOR** structure is required. For example, the code example in [Checking a Control Access Right in an Object's ACL](#) uses this method to retrieve a security descriptor to pass to the [AccessCheckByTypeResultList](#) function.

For more information, see:

- [Using IADs to Get a Security Descriptor](#)
- [Using IDirectoryObject to Get a Security Descriptor](#)
- [Security Descriptor Components](#)
- [Retrieving an Object's DACL](#)
- [Retrieving an Object's SACL](#)
- [Reading an Object's Security Descriptor](#)
- [Example Code for Creating a Security Descriptor](#)
- [Example Code for Enumerating the ACL of an Object in Active Directory Domain Services](#)

Using IADs to Get a Security Descriptor

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code examples use the [IADs::Get](#) method to retrieve an [IADsSecurityDescriptor](#) pointer to the **ntSecurityDescriptor** property of an object in Active Directory Domain Services.

```
Dim rootDSE As IADs
Dim ADUser As IADs
Dim sd As IADsSecurityDescriptor

On Error GoTo Cleanup

' Bind to the Users container in the local domain.
Set rootDSE = GetObject("LDAP://rootDSE")
Set ADUser = GetObject("LDAP://cn=users," & rootDSE.Get("defaultNamingContext"))

' Get the security descriptor on the Users container.
Set sd = ADUser.Get("ntSecurityDescriptor")
Debug.Print sd.Control
Debug.Print sd.Group
Debug.Print sd.Owner
Debug.Print sd.Revision

Exit Sub

Cleanup:
    Set rootDSE = Nothing
    Set ADUser = Nothing
    Set sd = Nothing
```

```
HRESULT GetSDFromIADs(
    IADs *pObject,
    IADsSecurityDescriptor **ppSD )
{
    VARIANT var;
    HRESULT hr;

    if(!pObject || !ppSD)
    {
        return E_INVALIDARG;
    }

    // Set *ppSD to NULL.
    *ppSD = NULL;

    VariantInit(&var);

    // Get the nTSecurityDescriptor.
    hr = pObject->Get(CComBSTR("nTSecurityDescriptor"), &var);
    if (SUCCEEDED(hr))
    {
        // Type should be VT_DISPATCH - an IDispatch pointer to the security descriptor object.
        if (var.vt == VT_DISPATCH)
        {
            // Use V_DISPATCH macro to get the IDispatch pointer from the
            // VARIANT structure and QueryInterface for the IADsSecurityDescriptor pointer.
            hr = V_DISPATCH(&var)->QueryInterface(IID_IADsSecurityDescriptor, (void**)ppSD);
        }
        else
        {
            hr = E_FAIL;
        }
    }

    VariantClear(&var);
    return hr;
}
```

Using IDirectoryObject to Get a Security Descriptor

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes a code example used to get a security descriptor.

The following C++ code example:

- Creates a buffer.
- Uses the [IDirectoryObject](#) interface to get the security descriptor of the specified object.
- Copies the security descriptor to the buffer.
- Returns a pointer to a [SECURITY_DESCRIPTOR](#) structure that contains the security descriptor data.

```

HRESULT GetSDFromIDirectoryObject(
    IDirectoryObject *pObject,
    PSECURITY_DESCRIPTOR *pSecurityDescriptor)
{
    HRESULT hr = E_FAIL;
    PADS_ATTR_INFO pAttrInfo;
    DWORD dwReturn= 0;
    LPWSTR pAttrName= L"nTSecurityDescriptor";
    PSECURITY_DESCRIPTOR pSD = NULL;

    if(!pObject || !pSecurityDescriptor)
    {
        return E_INVALIDARG;
    }

    // Get the nTSecurityDescriptor.
    hr = pObject->GetObjectAttributes( &pAttrName,
                                         1,
                                         &pAttrInfo,
                                         &dwReturn );
    if((FAILED(hr)) || (dwReturn != 1))
    {
        wprintf(L" failed: 0x%x\n", hr);
        return hr;
    }

    // Check the attribute name and type.
    if(_wcsicmp(pAttrInfo->pszAttrName,L"nTSecurityDescriptor") == 0) &&
       (pAttrInfo->dwADsType==ADSTYPE_NT_SECURITY_DESCRIPTOR)
    {
        // Get a pointer to the security descriptor.
        pSD = (PSECURITY_DESCRIPTOR)(pAttrInfo->pADsValues->SecurityDescriptor.lpValue);

        DWORD SDSIZE = pAttrInfo->pADsValues->SecurityDescriptor.dwLength;

        // Allocate memory for the buffer and copy the security descriptor to the buffer.
        *pSecurityDescriptor = (PSECURITY_DESCRIPTOR)CoTaskMemAlloc(SDSIZE);
        if (*pSecurityDescriptor)
        {
            CopyMemory((PVOID)*pSecurityDescriptor, (PVOID)pSD, SDSIZE);
        }
        else
        {
            hr = E_FAIL;
        }

        // Caller must free the memory for pSecurityDescriptor using CoTaskMemFree.
    }

    // Free memory used for the attributes retrieved.
    FreeADsMem(pAttrInfo);

    return hr;
}

```

Security Descriptor Components

6/3/2022 • 2 minutes to read • [Edit Online](#)

Having used the [IADs.Get](#) method to retrieve an **IADsSecurityDescriptor** interface pointer, you can use the **IADsSecurityDescriptor** properties to read or write the components of a directory object's security descriptor. For example, to get or set the object's DACL, use the **DiscretionaryAcl** property.

A security descriptor can store the following data:

- A security identifier (SID) that identifies the owner of the object: The owner of an object has the implicit right to modify the DACL and owner data in the object's security descriptor.
- A discretionary access-control list (DACL) that identifies the users and groups who can perform various operations on the object: A DACL contains a list of access-control entries (ACEs). Each ACE allows or denies a specified set of access rights to a specified user account, group account, or other trustee. For more information, see [Retrieving an Object's DACL](#).
- A system access-control list (SACL) that controls how the system audits attempts to access the object: Each ACE in a SACL specifies the types of access attempts that generate an audit log entry for a specified user account, group account, or other trustee. For more information, see [Retrieving an Object's SACL](#).
- A set of **SECURITY_DESCRIPTOR_CONTROL** control flags that qualify the meaning of a security descriptor or its components: For example, the **SE_DACL_PROTECTED** flag protects the security descriptor's DACL from inheriting ACEs from its parent object.
- A security identifier (SID) that identifies the primary group of the object: Active Directory Domain Services do not use this component.

For more information and a code example that can be used to read and display the data in an object security descriptor and DACL, see [Reading an Object's Security Descriptor](#).

Retrieving an Object's DACL

6/3/2022 • 2 minutes to read • [Edit Online](#)

An object's security descriptor may contain a discretionary access-control list (DACL). A DACL contains zero or more access-control entries (ACEs) that identify the users and groups who can access the object. If a DACL is empty (that is, it contains zero ACEs), no access is explicitly granted, so access is implicitly denied. However, if an object's security descriptor does not have a DACL, the object is unprotected and everyone has complete access.

To retrieve an object's DACL, you must be the object's owner or have READ_CONTROL access to the object.

To get and set the DACL of a directory object, use the [IADsSecurityDescriptor](#) interface. Using C++, the [IADsSecurityDescriptor::get_DiscretionaryAcl](#) method returns an [IDispatch](#) pointer. Call [QueryInterface](#) on that [IDispatch](#) pointer to get an [IADsAccessControlList](#) interface, and use the methods on that interface to access the individual ACEs in the DACL. The procedure for modifying a DACL is described in [Setting Access Rights on an Object](#).

To enumerate the ACEs, use the [IADsAccessControlList::get_NewEnum](#) method. The method returns an [IUnknown](#) pointer. Call [QueryInterface](#) on that [IUnknown](#) pointer to get an [IEnumVARIANT](#) interface. Use the [IEnumVARIANT::Next](#) method to enumerate the ACEs in the ACL. Each ACE is returned as a VARIANT containing an [IDispatch](#) pointer (the vt member is VT_DISPATCH). Call [QueryInterface](#) on that [IDispatch](#) pointer to get an [IADsAccessControlEntry](#) interface for the ACE. You can use the methods of the [IADsAccessControlEntry](#) interface to set or retrieve the components of an ACE.

For more information about DACLs and ACEs, see the following topics in the Platform Software Development Kit (SDK).

- [Access-Control Lists \(ACLs\)](#)
- [Access-Control Entries \(ACEs\)](#)

Retrieving an Object's SACL

6/3/2022 • 2 minutes to read • [Edit Online](#)

The security descriptor of an object in Active Directory Domain Services may contain a system access-control list (SACL). A SACL contains access-control entries (ACEs) that specify the types of access attempts that generate audit records in the security event log of a domain controller. Be aware that a SACL generates log entries only on the domain controller where the access attempt occurred, not on every DC that contains a replica of the object.

To set or get the SACL from an object security descriptor, the SE_SECURITY_NAME privilege must be enabled in the access token of the requesting thread. The administrators group has this privilege by default, and it can be assigned to other users or groups. For more information, see [SACL Access Right](#).

To get and set the SACL of a directory object, use the [IADsSecurityDescriptor](#) interface. Using C++, the [IADsSecurityDescriptor::get_SystemAcl](#) method returns an [IDispatch](#) pointer. Call [QueryInterface](#) on that [IDispatch](#) pointer to get an [IADsAccessControlList](#) interface, and use the methods on that interface to access the individual ACEs in the SACL. For more information about the procedure for modifying a SACL, which is similar to that for modifying a DACL, see [Setting Access Rights on an Object](#).

To enumerate the ACEs in a SACL, use the [IADsAccessControlList::get__NewEnum](#) method, which returns an [IUnknown](#) pointer. Call [QueryInterface](#) on that [IUnknown](#) pointer to get an [IEnumVARIANT](#) interface. Use the [IEnumVARIANT::Next](#) method to enumerate the ACEs in the ACL. Each ACE is returned as a **VARIANT** that contains an [IDispatch](#) pointer; be aware that the **vt** member is **VT_DISPATCH**. Call [QueryInterface](#) on that [IDispatch](#) pointer to get an [IADsAccessControlEntry](#) interface for the ACE. Use the [IADsAccessControlEntry](#) interface methods to set or get the components of an ACE.

For more information about SACLs, see:

- [Access-Control Lists \(ACLs\)](#)
- [Audit Generation](#)

Reading an Object's Security Descriptor

6/3/2022 • 7 minutes to read • [Edit Online](#)

The following code example uses the **IADs** interfaces to enumerate the properties of a directory object's security descriptor, DACL, and the ACEs of the DACL.

The following code example uses the **IADs.Get** method to retrieve the **nTSecurityDescriptor** property of the directory object. The C++ version of this method returns a **VARIANT** that contains an **IDispatch** pointer. The code example then calls **QueryInterface** on that **IDispatch** pointer to get an **IADsSecurityDescriptor** interface to the object's security descriptor.

The code example then uses **IADsSecurityDescriptor** methods to retrieve data from the security descriptor. Be aware that the data, available through **IADsSecurityDescriptor**, depends on the access rights of the caller. The **IADsSecurityDescriptor.DiscretionaryAcl** and **IADsSecurityDescriptor.Owner** properties will fail if the caller does not have **READ_CONTROL** access to the object. Similarly, a call to the **get_SystemAcl** method would fail if the caller does not have the **SE_SECURITY_NAME** privilege enabled.

```
Dim rootDSE As IADs
Dim dsobject As IADs
Dim sd As IADsSecurityDescriptor
Dim dacl As IADsAccessControlList
Dim ace As IADsAccessControlEntry

Private Sub Form_Load()

On Error GoTo Cleanup

' Bind to the Users container in the local domain.
Set rootDSE = GetObject("LDAP://rootDSE")
Set dsobject = GetObject("LDAP://cn=users," & rootDSE.Get("defaultNamingContext"))

' Read the security descriptor on the Users container.
Set sd = dsobject.Get("ntSecurityDescriptor")
Debug.Print "****SECURITY DESCRIPTOR PROPERTIES****"
Debug.Print "Revision: " & sd.Revision

SDParseControlMasks (sd.Control)
Debug.Print "Owner: " & sd.Owner
Debug.Print "Owner Defaulted: " & sd.OwnerDefaulted
Debug.Print "Group: " & sd.Group
Debug.Print "Group Defaulted: " & sd.GroupDefaulted
Debug.Print "System ACL Defaulted: " & sd.SaclDefaulted
Debug.Print "Discretionary ACL Defaulted: " & sd.DaclDefaulted
Debug.Print "****DACL PROPERTIES****"

Set dacl = sd.DiscretionaryAcl
AceCount = 0
For Each ace In dacl
    AceCount = AceCount + 1
    Debug.Print "**** Properties of ACE " & AceCount & " ****"
    Debug.Print "Trustee: " & ace.Trustee

    SDParseAccessMask (ace.AccessMask)

    AceType = ace.AceType
    If (AceType = &H5) Then
        Debug.Print "ACE Type: ADS_ACETYPE_ACCESS_ALLOWED_OBJECT"
    ElseIf (AceType = &H6) Then
        Debug.Print "ACE Type: ADS_ACETYPE_ACCESS_DENIED_OBJECT"
    ElseIf (AceType = &H9) Then
```

```

        ElseIf (AceType = &H0) Then
            Debug.Print "ACE Type: ADS_ACETYPE_ACCESS_ALLOWED"
        ElseIf (AceType = &H1) Then
            Debug.Print "ACE Type: ADS_ACETYPE_ACCESS_DENIED"
        Else
            Debug.Print "ACE Type: UNKNOWN TYPE: " & Hex(AceType)
        End If

        AceFlags = ace.Flags
        If (AceFlags And &H1) Then
            Debug.Print "Flags: ADS_FLAG_OBJECT_TYPE_PRESENT"
            Debug.Print "ObjectType: " & ace.ObjectType
        End If

        If (AceFlags And &H2) Then
            Debug.Print "Flags: ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT"
            Debug.Print "InheritedObjectType: " & ace.InheritedObjectType
        End If
    Next ace

    Cleanup:
        Set rootDSE = Nothing
        Set dsobject = Nothing
        Set sd = Nothing
        Set dacl = Nothing
        Set ace = Nothing

End Sub

' SDParseControlMasks
' Print flags set in the security descriptor Control.
Sub SDParseControlMasks(lCtrl As Long)
    Debug.Print "Control Mask: "
    If (lCtrl And &H1) Then Debug.Print " SE_OWNER_DEFAULTED"
    If (lCtrl And &H2) Then Debug.Print " SE_GROUP_DEFAULTED"
    If (lCtrl And &H4) Then Debug.Print " SE_DACL_PRESENT"
    If (lCtrl And &H8) Then Debug.Print " SE_DACL_DEFAULTED"
    If (lCtrl And &H10) Then Debug.Print " SE_SACL_PRESENT"
    If (lCtrl And &H20) Then Debug.Print " SE_SACL_DEFAULTED"
    If (lCtrl And &H400) Then Debug.Print " SE_DACL_AUTO_INHERITED"
    If (lCtrl And &H800) Then Debug.Print " SE_SACL_AUTO_INHERITED"
    If (lCtrl And &H1000) Then Debug.Print " SE_DACL_PROTECTED"
    If (lCtrl And &H2000) Then Debug.Print " SE_SACL_PROTECTED"
    If (lCtrl And &H8000) Then Debug.Print " SE_SELF_RELATIVE"
End Sub

' SDParseAccessMask
' Print the access rights in an access mask.
Sub SDParseAccessMask(lMask As Long)
    Debug.Print "AccessMask: "
    If (lMask And &H10000) Then Debug.Print " ADS_RIGHT_DELETE"
    If (lMask And &H20000) Then Debug.Print " ADS_RIGHT_READ_CONTROL"
    If (lMask And &H40000) Then Debug.Print " ADS_RIGHT_WRITE_DAC"
    If (lMask And &H80000) Then Debug.Print " ADS_RIGHT_WRITE_OWNER"
    If (lMask And &H8000000) Then Debug.Print " ADS_RIGHT_GENERIC_READ"
    If (lMask And &H4000000) Then Debug.Print " ADS_RIGHT_GENERIC_WRITE"
    If (lMask And &H2000000) Then Debug.Print " ADS_RIGHT_GENERIC_EXECUTE"
    If (lMask And &H1000000) Then Debug.Print " ADS_RIGHT_GENERIC_ALL"
    If (lMask And &H1) Then Debug.Print " ADS_RIGHT_DS_CREATE_CHILD"
    If (lMask And &H2) Then Debug.Print " ADS_RIGHT_DS_DELETE_CHILD"
    If (lMask And &H4) Then Debug.Print " ADS_RIGHT_ACTRL_DS_LIST"
    If (lMask And &H8) Then Debug.Print " ADS_RIGHT_DS_SELF"
    If (lMask And &H10) Then Debug.Print " ADS_RIGHT_DS_READ_PROP"
    If (lMask And &H20) Then Debug.Print " ADS_RIGHT_DS_WRITE_PROP"
    If (lMask And &H40) Then Debug.Print " ADS_RIGHT_DS_DELETE_TREE"
    If (lMask And &H80) Then Debug.Print " ADS_RIGHT_DS_LIST_OBJECT"
    If (lMask And &H100) Then Debug.Print " ADS_RIGHT_DS_CONTROL_ACCESS"
End Sub

```

```

HRESULT ReadSecurityDescriptor( IAdS *pObject )
{
    HRESULT hr = E_FAIL;
    VARIANT var,varACE;
    VARIANT_BOOL boolVal;
    IAdSSecurityDescriptor *pSD = NULL;
    IAdSAccessControlList *pACL = NULL;
    IAdSAccessControlEntry *pACE = NULL;
    IEnumVARIANT *pEnum = NULL;
    IDispatch *pDisp = NULL;
    LPUNKNOWN pUnk = NULL;
    ULONG lFetch;
    BSTR szObjectType = NULL;
    BSTR szTrustee = NULL;
    BSTR szProp = NULL;
    long lRev,lControl;
    long lAceType, lAccessMask, lTypeFlag;
    DWORD dwAces = 0;

    if (NULL == pObject)
        return hr;

    VariantClear(&var);

    // Get the nTSecurityDescriptor.
    // Type should be VT_DISPATCH - an IDispatch pointer to the security descriptor object.
    hr = pObject->Get(CComBSTR("nTSecurityDescriptor"), &var);
    if ( FAILED(hr) || var.vt != VT_DISPATCH ) {
        wprintf(L"get nTSecurityDescriptor failed: 0x%x\n", hr);
        goto Cleanup;
    }

    hr = V_DISPATCH( &var )->QueryInterface(IID_IAdSSecurityDescriptor,(void**)&pSD);
    if ( FAILED(hr) ) {
        wprintf(L"QueryInterface for IAdSSecurityDescriptor failed: 0x%x\n", hr);
        goto Cleanup;
    }

    wprintf(L"****SECURITY DESCRIPTOR PROPERTIES****\n");

    // Get properties using IAdSSecurityDescriptor property methods.
    hr = pSD->get_Revision(&lRev);
    wprintf(L"Revision: %d\n", lRev);

    // Print the control mask bits.
    hr = pSD->get_Control(&lControl);
    wprintf(L"Control Mask: \n");
    SDParseControlMasks(lControl);

    hr = pSD->get_Owner(&szProp);
    wprintf(L"Owner: %s\n", szProp);
    if (szProp)
        SysFreeString(szProp);

    hr = pSD->get_OwnerDefaulted(&boolVal);
    wprintf(L"Owner Defaulted: %s\n", boolVal ? L"True" : L"False");

    hr = pSD->get_Group(&szProp);
    wprintf(L"Group: %s\n", szProp);
    if (szProp)
        SysFreeString(szProp);

    hr = pSD->get_GroupDefaulted(&boolVal);
    wprintf(L"Group Defaulted: %s\n", boolVal ? L"True" : L"False");

    hr = pSD->get_SaclDefaulted(&boolVal);
    wprintf(L"System ACL Defaulted: %s\n", boolVal ? L"True" : L"False");

```

```

hr = pSD->get_DaclDefaulted(&boolVal);
wprintf(L"Discretionary ACL Defaulted: %s\n", boolVal ? L"True" : L"False");

// Get the DACL.
wprintf(L"****DACL PROPERTIES****\n");
hr = pSD->get_DiscretionaryAcl(&pDisp);
if (SUCCEEDED(hr))
{
    hr = pDisp->QueryInterface(IID_IADsAccessControlList, (void**)&pACL);
    if (SUCCEEDED(hr))
    {
        hr = pACL->get__NewEnum( &pUnk );
        if (SUCCEEDED(hr))
        {
            hr = pUnk->QueryInterface( IID_IEnumVARIANT, (void**) &pEnum );
        }
    }
}
if ( FAILED(hr) ) {
    wprintf(L"Could not get DACL: 0x%x\n", hr);
    goto Cleanup;
}

// Loop to read all ACEs on the object.
hr = pEnum->Next( 1, &varACE, &lFetch );
while ( (hr == S_OK) && (lFetch == 1) && (varACE.vt==VT_DISPATCH) )
{
    // QueryInterface for an IADsAccessControlEntry to use to read the ACE.
    hr = V_DISPATCH(&varACE)->QueryInterface(
        IID_IADsAccessControlEntry, (void**)&pACE );
    if ( FAILED(hr) ) {
        wprintf(L"QI for IADsAccessControlEntry failed: 0x%x\n", hr);
        break;
    }

    wprintf(L"**** Properties of ACE %u ****\n", ++dwAces);

    // Get the trustee that the ACE applies to and print it.
    hr = pACE->get_Trustee(&szTrustee);
    if (SUCCEEDED(hr))
    {
        wprintf(L"Trustee: %s\n", szTrustee);
        if (szTrustee)
            SysFreeString(szTrustee);
    }

    // Get the AceMask.
    hr = pACE->get_AccessMask(&lAccessMask);
    if (SUCCEEDED(hr))
    {
        wprintf(L"AccessMask: \n");
        SDParseAccessMask(lAccessMask);
    }

    // Get the AceType.
    hr = pACE->get_AceType(&lAceType);
    if (SUCCEEDED(hr))
    {
        if (lAceType == ADS_ACETYPE_ACCESS_ALLOWED_OBJECT)
            wprintf(L"ACE Type: ADS_ACETYPE_ACCESS_ALLOWED_OBJECT\n");
        else if (lAceType == ADS_ACETYPE_ACCESS_DENIED_OBJECT)
            wprintf(L"ACE Type: ADS_ACETYPE_ACCESS_DENIED_OBJECT\n");
        else if (lAceType == ADS_ACETYPE_ACCESS_ALLOWED)
            wprintf(L"ACE Type: ADS_ACETYPE_ACCESS_ALLOWED\n");
        else if (lAceType == ADS_ACETYPE_ACCESS_DENIED)
            wprintf(L"ACE Type: ADS_ACETYPE_ACCESS_DENIED\n");
        else
            wprintf(L"ACE Type: UNKNOWN TYPE: %x\n", lAceType);
    }
}

```

```

        }

        // Get the flags. Based on flags, get objecttype and inheritedobjecttype.
        hr = pACE->get_Flags(&lTypeFlag);
        if (SUCCEEDED(hr))
        {
            // If the object type GUID is present, print it.
            if (lTypeFlag & ADS_FLAG_OBJECT_TYPE_PRESENT)
            {
                wprintf(L"Flags: ADS_FLAG_OBJECT_TYPE_PRESENT\n");
                hr = pACE->get_ObjectType(&szObjectType);
                if (SUCCEEDED(hr))
                {
                    wprintf(L"ObjectType: %s\n", szObjectType);
                    if (szObjectType)
                        SysFreeString(szObjectType);
                }
            }

            // If the inherited object type GUID is present, print it.
            if (lTypeFlag & ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT)
            {
                wprintf(L"Flags: ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT\n");
                hr = pACE->get_InheritedObjectType(&szObjectType);
                if (SUCCEEDED(hr))
                {
                    wprintf(L"InheritedObjectType: %s\n", szObjectType);
                    if (szObjectType)
                        SysFreeString(szObjectType);
                }
            }
        }

        // Cleanup the enumerated ACE item.
        if (pACE)
            pACE->Release();

        // Cleanup the VARIANT for the ACE item.
        VariantClear(&varACE);

        // Get the next ACE.
        hr = pEnum->Next( 1, &varACE, &lFetch );

    } // End of While loop

    Cleanup:
    if (pEnum)
        pEnum->Release();
    if (pUnk)
        pUnk->Release();
    if (pACL)
        pACL->Release();
    if (pDisp)
        pDisp->Release();
    if (pSD)
        pSD->Release();
    VariantClear(&var);
    return hr;
}

// ****
// SDParseControlMasks
// Function to print Control flags.
// ****
int SDParseControlMasks( long lCtrl )
{
    int iReturn = TRUE;

    if (lCtrl & SE_OWNER_DEFAULTED)
        wprintf(L" SE_OWNER_DEFAULTED\n");
}

```

```

wprintf(L" SE_OWNER_DEFAULTED\n");

if (lCtrl & SE_GROUP_DEFAULTED)
    wprintf(L" SE_GROUP_DEFAULTED\n");

if (lCtrl & SE_DACL_PRESENT)
    wprintf(L" SE_DACL_PRESENT\n");

if (lCtrl & SE_DACL_DEFAULTED)
    wprintf(L" SE_DACL_DEFAULTED\n");

if (lCtrl & SE_SACL_PRESENT)
    wprintf(L" SE_SACL_PRESENT\n");

if (lCtrl & SE_SACL_DEFAULTED)
    wprintf(L" SE_SACL_DEFAULTED\n");

if (lCtrl & SE_DACL_AUTO_INHERIT_REQ)
    wprintf(L" SE_DACL_AUTO_INHERIT_REQ\n");

if (lCtrl & SE_SACL_AUTO_INHERIT_REQ)
    wprintf(L" SE_SACL_AUTO_INHERIT_REQ\n");

if (lCtrl & SE_DACL_AUTO_INHERITED)
    wprintf(L" SE_DACL_AUTO_INHERITED\n");

if (lCtrl & SE_SACL_AUTO_INHERITED)
    wprintf(L" SE_SACL_AUTO_INHERITED\n");

if (lCtrl & SE_DACL_PROTECTED)
    wprintf(L" SE_DACL_PROTECTED\n");

if (lCtrl & SE_SACL_PROTECTED)
    wprintf(L" SE_SACL_PROTECTED\n");

if (lCtrl & SE_SELF_RELATIVE)
    wprintf(L" SE_OWNER_DEFAULTED\n");

return iReturn;
}

// *****
// SDParseAccessMask
// Function to print AccessMask flags.
// *****

int SDParseAccessMask( long lCtrl )
{
    int iReturn = TRUE;

    if (lCtrl & ADS_RIGHT_DELETE)
        wprintf(L" ADS_RIGHT_DELETE\n");

    if (lCtrl & ADS_RIGHT_READ_CONTROL)
        wprintf(L" ADS_RIGHT_READ_CONTROL\n");

    if (lCtrl & ADS_RIGHT_WRITE_DAC)
        wprintf(L" ADS_RIGHT_WRITE_DAC\n");

    if (lCtrl & ADS_RIGHT_WRITE_OWNER)
        wprintf(L" ADS_RIGHT_WRITE_OWNER\n");

    if (lCtrl & ADS_RIGHT_GENERIC_READ)
        wprintf(L" ADS_RIGHT_GENERIC_READ\n");

    if (lCtrl & ADS_RIGHT_GENERIC_WRITE)
        wprintf(L" ADS_RIGHT_GENERIC_WRITE\n");

    if (lCtrl & ADS_RIGHT_GENERIC_EXECUTE)
        wprintf(L" ADS_RIGHT_GENERIC_EXECUTE\n");
}

```

```
wprintf(L"  ADS_RIGHT_GENERIC_EXECUTE\n");

if (lCtrl & ADS_RIGHT_GENERIC_ALL)
    wprintf(L"  ADS_RIGHT_GENERIC_ALL\n");

if (lCtrl & ADS_RIGHT_DS_CREATE_CHILD)
    wprintf(L"  ADS_RIGHT_DS_CREATE_CHILD\n");

if (lCtrl & ADS_RIGHT_DS_DELETE_CHILD)
    wprintf(L"  ADS_RIGHT_DS_DELETE_CHILD\n");

if (lCtrl & ADS_RIGHT_ACTRL_DS_LIST)
    wprintf(L"  ADS_RIGHT_ACTRL_DS_LIST\n");

if (lCtrl & ADS_RIGHT_DS_SELF)
    wprintf(L"  ADS_RIGHT_DS_SELF\n");

if (lCtrl & ADS_RIGHT_DS_READ_PROP)
    wprintf(L"  ADS_RIGHT_DS_READ_PROP\n");

if (lCtrl & ADS_RIGHT_DS_WRITE_PROP)
    wprintf(L"  ADS_RIGHT_DS_WRITE_PROP\n");

if (lCtrl & ADS_RIGHT_DS_DELETE_TREE)
    wprintf(L"  ADS_RIGHT_DS_DELETE_TREE\n");

if (lCtrl & ADS_RIGHT_DS_LIST_OBJECT)
    wprintf(L"  ADS_RIGHT_DS_LIST_OBJECT\n");

if (lCtrl & ADS_RIGHT_DS_CONTROL_ACCESS)
    wprintf(L"  ADS_RIGHT_DS_CONTROL_ACCESS\n");

return iReturn;

}
```

Example Code for Creating a Security Descriptor

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes a Visual Basic code example that shows you how to create a security descriptor object for an ADSI object.

```
' This code example assumes that you have the credentials
' required to create objects.

Dim MyObject As IADs
Dim MySecDes As SecurityDescriptor
Dim Var As Variant
Dim SecDes As New SecurityDescriptor
Dim Dacl As New AccessControlList
Dim Ace As New AccessControlEntry

On Error GoTo Cleanup

' Create an Access Control Entry (ACE) for Group objects
' at Fabrikam on an LDAP namespace.
Ace.AccessMask = 0
Ace.AceType = 1
Ace.AceFlags = 1
Ace.Trustee = "cn=Groups,o=Fabrikam"

' Add the new ACE object as the only ACE in a new
' access-control list (ACL).
Dacl.AceCount = 1
Dacl.AclRevision = 4
Dacl.AddAce Ace

' Use this ACL as the discretionary ACL (DACL) for
' this Security Descriptor object and use this DACL
' instead of the default.
SecDes.Revision = 1
SecDes.OwnerDefaulted = True
SecDes.GroupDefaulted = True
SecDes.DaclDefaulted = False
SecDes.SaclDefaulted = True
SecDes.DiscretionaryAcl = Dacl

' Attach this security descriptor to the ADSI object.
MyObject.Put "ntSecurityDescriptor", SecDes
' Commit the changes to the underlying directory service.
MyObject.SetInfo
' Read the properties back in to the property cache.
MyObject.GetInfo
' Get the SecurityDescriptor object.
Set MySecDes = MyObject.Get("ntSecurityDescriptor")

Cleanup:
If (Err.Number<>0) Then
    MsgBox("An error has occurred. " & Err.Number)
End If
Set MyObject = Nothing
Set MySecDes = Nothing
Set SecDes = Nothing
Set Dacl = Nothing
Set Ace = Nothing
```


Example Code for Enumerating the ACL of an Object in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code examples can be used to enumerate the access control list (ACL) of an object in Active Directory Domain Services.

The following code example shows a function that enumerates the trustees of an object .

```
//*****
//  EnumTrustees()
//*****
HRESULT EnumTrustees(IADsAccessControlList *pACL)
{
    if(NULL == pACL)
    {
        return E_INVALIDARG;
    }

    HRESULT hr;
    IUnknown *pUnk;

    /*
    Get the enumerator from the access control list.
    */
    hr = pACL->get__NewEnum(&pUnk);
    if(SUCCEEDED(hr))
    {
        IEnumVARIANT *pEnum;

        hr = pUnk->QueryInterface(IID_IEnumVARIANT, (LPVOID*)&pEnum);
        if(SUCCEEDED(hr))
        {
            VARIANT var;
            ULONG ulFetched;

            wprintf(L"Trustees:\n");

            VariantInit(&var);

            /*
            Enumerate the access control entries
            in the access control list.
            */
            while( SUCCEEDED(hr = pEnum->Next(1, &var, &ulFetched))
                  && (ulFetched > 0) )
            {
                IADsAccessControlEntry *pACE;

                /*
                Get the access control entry.
                */
                hr = V_DISPATCH(&var)->QueryInterface(IID_IADsAccessControlEntry,
                                                (LPVOID*)&pACE);
                if(SUCCEEDED(hr))
                {
                    CComBSTR sbstrTrustee;
```

```

        /*
         Get the Trustee for this ACE and print
         it to the console window.
        */
        hr = pACE->get_Trustee(&sbstrTrustee);
        if(SUCCEEDED(hr))
        {
            wprintf(L"\t");
            wprintf(sbstrTrustee);
            wprintf(L"\n");
        }

        pACE->Release();
    }

    VariantClear(&var);
}

pEnum->Release();
}

pUnk->Release();
}

return hr;
}

//*****
//  EnumAccessInfo()
//*****
//*****


HRESULT EnumAccessInfo(IADs *pads)
{
    if(NULL == pads)
    {
        return E_INVALIDARG;
    }

    HRESULT hr;
    VARIANT var;

    // Get the ntSecurityDescriptor attribute
    VariantInit(&var);
    hr = pads->Get(CComBSTR("ntSecurityDescriptor"), &var);
    if(SUCCEEDED(hr))
    {
        if(VT_DISPATCH == var.vt)
        {
            /*
             Get the security descriptor from the
             ntSecurityDescriptor attribute.
            */
            IADsSecurityDescriptor *pSD;
            hr = V_DISPATCH(&var)->QueryInterface(IID_IADsSecurityDescriptor,
                                                       (LPVOID*)&pSD);
            if(SUCCEEDED(hr))
            {
                IDispatch *pDisp;

                /*
                 Get the DACL from the security descriptor.
                */
                hr = pSD->get_DiscretionaryAcl(&pDisp);
                if(SUCCEEDED(hr))
                {
                    IADsAccessControlList *pACL;

```

```

        hr = pDisp->QueryInterface(IID_IADsAccessControlList,
                                    (LPVOID*)&pACL);
        if(SUCCEEDED(hr))
        {
            /*
             * Enumerate the trustees of this ACL.
             */
            hr = EnumTrustees(pACL);

            pACL->Release();
        }

        pDisp->Release();
    }

    pSD->Release();
}

VariantClear(&var);
}

return hr;
}

```

The following code example shows a function that enumerates the trustees of an object.

```

Private Sub EnumAccessInfo(ByVal oObject As IADs)
    Dim SecDesc As SecurityDescriptor
    Dim Dacl As AccessControlList

    On Error GoTo CleanUp

    ' Get the security descriptor for the object.
    Set SecDesc = oObject.Get("ntSecurityDescriptor")

    ' Get the DACL for the object.
    Set Dacl = SecDesc.DiscretionaryAcl

    Debug.Print "Trustees:"

    ' Enumerate the ACEs in the DACL, printing the Trustee for each.
    For Each oACE In Dacl
        Debug.Print vbTab + oACE.Trustee
    Next

CleanUp:
    Set SecDesc = Nothing
    Set Dacl = Nothing
End Sub

```

Setting Access Rights on an Object

6/3/2022 • 4 minutes to read • [Edit Online](#)

When using the ADSI COM objects **IADsSecurityDescriptor** (security descriptor), **IADsAccessControlList** (DACLs and SACLs), and **IADsAccessControlEntry** (ACE) to add an ACE to a ACL, you are making changes to the **nTSecurityDescriptor** property of the specified object in the property cache. This means put methods on the objects that contain the new ACE and the **IADs.SetInfo** method must be called in order to write the updated security descriptor to the directory from the property cache.

For more information and a code example that sets an ACE on an object in Active Directory Domain Services, see [Example Code for Setting an ACE on a Directory Object](#).

Use the following general process to create an ACE for an access right and setting that ACE on the DACL of an object.

1. Get an **IADs** interface pointer to the object.
2. Use the **IADs.Get** method to get the security descriptor of the object. The name of the property containing the security descriptor is **nTSecurityDescriptor**. The property will be returned as a **VARIANT** containing an **IDispatch** pointer (the **vt** member is **VT_DISPATCH**). Call **QueryInterface** on that **IDispatch** pointer to get an **IADsSecurityDescriptor** interface to use the methods on that interface to access the security descriptor's ACL.
3. Use the **IADsSecurityDescriptor.DiscretionaryAcl** property to get the DACL. The method returns an **IDispatch** pointer. Call **QueryInterface** on that **IDispatch** pointer to get an **IADsAccessControlList** interface to use the methods on that interface to access the individual ACEs in the ACL.
4. Use **CoCreateInstance** to create the ADSI COM object for the new ACE and get an **IADsAccessControlEntry** interface pointer to that object. Be aware that the class ID is **CLSID_AccessControlEntry**.
5. Set the properties of the ACE using the **IADsAccessControlEntry** methods:
 - a. Use **IADsAccessControlEntry::put_Trustee** to set the trustee to whom this ACE applies. The trustee is a user, group, or other security principal. Your application should use the value from the appropriate property from the user or group object of the trustee to which you want to apply the ACE. The trustee is specified as a **BSTR** and can take the following forms:
 - Domain account (the logon name used in a previous version of Windows NT) of the form "**<domain>\<user account>**" where "**<domain>**" is the name of the Windows NT domain that contains the user and "**<user account>**" is the **sAMAccountName** property of the specified user. For example: "fabrikam\jeffsmith".
 - Well-known security principal that represents special identities defined by the Windows NT security system, such as everyone, local system, principal self, authenticated user, creator owner, and so on. The objects representing the well-known security principals are stored in the Well Known Security Principals container beneath the Configuration container. For example, anonymous logon.
 - Built-in group that represent the built-in user groups defined by the Windows NT security system. It has the form "**BUILTIN\<group name>**" where "**<group name>**" is the name of the built-in user group. The objects representing the built-in groups are stored in the **Builtin** container beneath the domain container. For example, "BUILTIN\Administrators".
 - SID (string format) of the specified user, which is the **objectSID** property of the specified user.

You can convert to string form using the [ConvertSidToStringSid](#) function in the Win32 Security API. For example: "S-1-5-32-548".

- b. Use the [IADsAccessControlEntry.AccessMask](#) property to set the mask that specifies the access right. The [ADS_RIGHTS_ENUM](#) enumeration specifies the access rights you can set on a directory object.
 - c. Use the [IADsAccessControlEntry.AceType](#) property to specify whether to allow or deny the access rights set by [AccessMask](#). For standard rights, this can be [ADS_ACETYPE_ACCESS_ALLOWED](#) or [ADS_ACETYPE_ACCESS_DENIED](#). For object-specific rights (rights that apply to a specific part of an object or to a specific type of object), use [ADS_ACETYPE_ACCESS_ALLOWED_OBJECT](#) or [ADS_ACETYPE_ACCESS_DENIED_OBJECT](#). The [ADS_ACETYPE_ENUM](#) enumeration specifies the access types you can set on an ACE.
 - d. Use the [IADsAccessControlEntry.AceFlags](#) property to specify whether other containers or objects beneath the specified object can inherit the ACE. The [ADS_ACEFLAG_ENUM](#) enumeration specifies the inheritance flags you can set on an ACE.
 - e. Use the [IADsAccessControlEntry.Flags](#) property to specify whether the right applies to a specific part of the object, an inherited object type, or both.
 - f. If [Flags](#) is set to [ADS_FLAG_OBJECT_TYPE_PRESENT](#), set the [IADsAccessControlEntry.ObjectType](#) property to specify a string containing the GUID of the object class (for [ADS_RIGHT_DS_CREATE_CHILD](#) or [ADS_RIGHT_DS_DELETE_CHILD](#)), property, property set, validated write, or extended right that the ACE applies to. The GUID must be specified as a string of the form produced by the [StringFromGUID2](#) function in the COM library.
 - g. If [Flags](#) is set to [ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT](#), set the [IADsAccessControlEntry.InheritedObjectType](#) property to specify a string that contains the GUID of the inherited object class that the ACE applies to. The GUID must be specified as a string of the form produced by the [StringFromGUID2](#) function in the COM library.
6. Use the [QueryInterface](#) method on the [IADsAccessControlEntry](#) object to get an [IDispatch](#) pointer. The [IADsAccessControlList.AddAce](#) method requires an [IDispatch](#) interface pointer to the ACE.
 7. Use [IADsAccessControlList.AddAce](#) to add the new ACE to the DACL. Be aware that the order of the ACEs within the ACL can affect the evaluation of access to the object. The correct access to the object may require you to create a new ACL, add the ACEs from the existing ACL in the correct order to the new ACL, and then replace the existing ACL in the security descriptor with the new ACL. For more information, see [Order of ACEs in a DACL](#).
 8. Use the [IADsSecurityDescriptor.DiscretionaryAcl](#) property to write the DACL containing the new ACE to the security descriptor. For more information about DACLs, see [Null DACLs and Empty DACLs](#).
 9. Use the [IADs.Put](#) method to write the security descriptor to the object's [nTSecurityDescriptor](#) property to the property cache.
 10. Use the [IADs.SetInfo](#) method to update the property on the object in the directory.

Example Code for Setting an ACE on a Directory Object

6/3/2022 • 9 minutes to read • [Edit Online](#)

Define the SetRight function

The following code example defines a function that adds an Access Control Entry (ACE) to the Discretionary Access Control List (DACL) of the security descriptor of a specified object in Active Directory Domain Services. The subroutine enables you to:

- Grant or deny access to the entire object.
- Grant or deny access to a specific property on the object.
- Grant or deny access to a set of properties on the object.
- Grant or deny the right to create a specific type of child object.
- Set an ACE that can be inherited by all child objects or by child objects of a specified object class.

Following this Visual Basic code example are several code examples that show how to use the **SetRight** function to set different types of ACEs.

```
Const ACL_REVISION_DS = &H4

Public Function SetRight(objectDN As String, _
    accessrights As Long, _
    accesstype As Long, _
    aceinheritflags As Long, _
    objectGUID As String, _
    inheritedObjectGUID As String, _
    trustee As String) As Boolean

    Dim dsobject As IADs
    Dim sd As IADsSecurityDescriptor
    Dim dacl As IADsAccessControlList
    Dim newace As New AccessControlEntry
    Dim lflags As Long

    On Error GoTo Cleanup

    ' Bind to the specified object.
    Set dsobject = GetObject(objectDN)

    ' Read the security descriptor on the object.
    Set sd = dsobject.Get("ntSecurityDescriptor")

    ' Get the DACL from the security descriptor.
    Set dacl = sd.DiscretionaryAcl

    ' Set the properties of the new ACE.
    newace.AccessMask = accessrights
    newace.AceType = accesstype
    newace.AceFlags = aceinheritflags
    newace.trustee = trustee

    ' Set the GUID for the object type or inherited object type.
    lflags = 0

    If Not objectGUID = vbNullString Then
        newace.ObjectType = objectGUID
    End If

    ' Add the new ACE to the DACL.
    dacl.AddAce(newace)
    sd.SetSecurityDescriptor(dacl)
    SetRight = True

Cleanup:
    If Not dsobject Is Nothing Then
        Set dsobject = Nothing
    End If
    If Not sd Is Nothing Then
        Set sd = Nothing
    End If
    If Not dacl Is Nothing Then
        Set dacl = Nothing
    End If
    If Not newace Is Nothing Then
        Set newace = Nothing
    End If
End Function
```

```

lflags = lflags Or &H1 'ADS_FLAG_OBJECT_TYPE_PRESENT
End If

If Not inheritedObjectGUID = vbNullString Then
    newace.InheritedObjectType = inheritedObjectGUID
    lflags = lflags Or &H2 'ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT
End If

If Not (lflags = 0) Then newace.Flags = lflags

' Set the ACL Revision.
dacl.AclRevision = ACL_REVISION_DS

' Add the ACE to the DACL and to the security descriptor.
dacl.AddAce newace
sd.DiscretionaryAcl = dacl

' Apply it to the object.
dsobject.Put "ntSecurityDescriptor", sd
dsobject.SetInfo
SetRight = True
Exit Function

Cleanup:
Set dsobject = Nothing
Set sd = Nothing
Set dacl = Nothing
Set newace = Nothing
SetRight = False

End Function

```

```

HRESULT SetRight(
    IADs *pObject,
    long lAccessMask,
    long lAccessType,
    long lAccessInheritFlags,
    LPOLESTR szObjectGUID,
    LPOLESTR szInheritedObjectGUID,
    LPOLESTR szTrustee)
{
VARIANT varSD;
HRESULT hr = E_FAIL;
IADsAccessControlList *pACL = NULL;
IADsSecurityDescriptor *pSD = NULL;
IDispatch *pDispDACL = NULL;
IADsAccessControlEntry *pACE = NULL;
IDispatch *pDispACE = NULL;
long lFlags = 0L;

// The following code example takes the szTrustee in an expected naming format
// and assumes it is the name for the correct trustee.
// The application should validate the specified trustee.
if (!szTrustee || !pObject)
    return E_INVALIDARG;

VariantInit(&varSD);

// Get the nTSecurityDescriptor.
// Type should be VT_DISPATCH - an IDispatch pointer to the security descriptor object.
hr = pObject->Get(_bstr_t("nTSecurityDescriptor"), &varSD);
if (FAILED(hr) || varSD.vt != VT_DISPATCH ) {
    wprintf(L"get nTSecurityDescriptor failed: 0x%x\n", hr);
    return hr;
}

hr = V_DISPATCH( &varSD )->QueryInterface(IID_IADsSecurityDescriptor,(void**)&pSD);

```

```

if ( FAILED(hr) ) {
    wprintf(L"QueryInterface for IADsSecurityDescriptor failed: 0x%x\n", hr);
    goto cleanup;
}

// Get the DACL.
hr = pSD->get_DiscretionaryAcl(&pDispDACL);
if (SUCCEEDED(hr))
    hr = pDispDACL->QueryInterface(IID_IADsAccessControlList,(void**)&pACL);
if ( FAILED(hr) ) {
    wprintf(L"Could not get DACL: 0x%x\n", hr);
    goto cleanup;
}

// Create the COM object for the new ACE.
hr = CoCreateInstance(
    CLSID_AccessControlEntry,
    NULL,
    CLSCTX_INPROC_SERVER,
    IID_IADsAccessControlEntry,
    (void **)&pACE
);
if ( FAILED(hr) ) {
    wprintf(L"Could not create ACE object: 0x%x\n", hr);
    goto cleanup;
}

// Set the properties for the new ACE.

// Set the mask that specifies the access right.
hr = pACE->put_AccessMask( lAccessMask );

// Set the trustee.
hr = pACE->put_Trustee( szTrustee );

// Set AceType.
hr = pACE->put_AceType( lAccessType );

// Set AceFlags to specify whether other objects can inherit the ACE from the specified object.
hr = pACE->put_AceFlags( lAccessInheritFlags );

// If an szObjectGUID is specified, add ADS_FLAG_OBJECT_TYPE_PRESENT
// to the lFlags mask and set the ObjectType.
if (szObjectGUID)
{
    lFlags |= ADS_FLAG_OBJECT_TYPE_PRESENT;
    hr = pACE->put_ObjectType( szObjectGUID );
}

// If an szInheritedObjectGUID is specified, add ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT
// to the lFlags mask and set the InheritedObjectType.
if (szInheritedObjectGUID)
{
    lFlags |= ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT;
    hr = pACE->put_InheritedObjectType( szInheritedObjectGUID );
}

// Set flags if ObjectType or InheritedObjectType were set.
if (lFlags)
    hr = pACE->put_Flags(lFlags);

// Add the ACE to the ACL to the SD to the cache to the object.
// Call the QueryInterface method for the IDispatch pointer to pass to the AddAce method.
hr = pACE->QueryInterface(IID_IDispatch, (void**)&pDispACE);
if (SUCCEEDED(hr))
{
    // Set the ACL revision.
    hr = pACL->put_AclRevision(ACL_REVISION_DS);
}

```

```

// Add the ACE.
hr = pACL->AddAce(pDispACE);
if (SUCCEEDED(hr))
{
    // Write the DACL.
    hr = pSD->put_DiscretionaryAcl(pDispDACL);
    if (SUCCEEDED(hr))
    {
        // Write the ntSecurityDescriptor property to the property cache.
        hr = pObject->Put(CComBSTR("nTSecurityDescriptor"), varSD);
        if (SUCCEEDED(hr))
        {
            // Call SetInfo to update the property on the object in the directory.
            hr = pObject->SetInfo();
        }
    }
}

cleanup:
if (pDispACE)
    pDispACE->Release();
if (pACE)
    pACE->Release();
if (pACL)
    pACL->Release();
if (pDispDACL)
    pDispDACL->Release();
if (pSD)
    pSD->Release();

VariantClear(&varSD);
return hr;
}

```

Grant or deny access to the entire object

The following Visual Basic code example builds a binding string for the Users container and then calls the **SetRight** function to set an ACE on the Users container. This example sets an ACE that grants the trustee the right to read or write any property on the object.

```

Dim rootDSE As IADs
Dim objectDN As String
Dim bResult As Boolean
Const ADS_RIGHT_READ_PROP = &H10
Const ADS_RIGHT_WRITE_PROP = &H20

' Bind to the Users container in the local domain.
Set rootDSE = GetObject("LDAP://rootDSE")
objectDN = "LDAP://cn=users," & rootDSE.Get("defaultNamingContext")

' Grant trustee the right to read/write any property.
bResult = SetRight objectDN, _
    ADS_RIGHT_READ_PROP Or ADS_RIGHT_WRITE_PROP, _
    ADS_ACETYPE_ACCESS_ALLOWED, _
    0, _
    vbNullString, _
    vbNullString, _
    "someone@fabrikam.com" ' Trustee

If bResult = True Then
    MsgBox ("The trustee can read or write any property.")
Else
    MsgBox ("An error occurred.")
End If

```

The following C++ code example sets an ACE that grants the trustee permission to read or write any property on the object. The code example assumes that *pObject* and *szTrustee* are set to valid values. For more information, see [Setting Access Rights on an Object](#).

```

HRESULT hr;
IADs *pObject;
LPWSTR szTrustee;

hr = SetRight(
    pObject, // IADs pointer to the object
    ADS_RIGHT_READ_PROP | ADS_RIGHT_WRITE_PROP,
    ADS_ACETYPE_ACCESS_ALLOWED,
    0, // Not inheritable
    NULL, // No object type GUID
    NULL, // No inherited object type GUID
    szTrustee
);

```

Grant or deny access to a specific property on the object

This code example calls the `SetRight` function to grant the trustee the right to read or write a specific property on the object. Be aware that you must specify the `schemaIDGUID` of the property and you must specify `ADS_ACETYPE_ACCESS_ALLOWED_OBJECT` to indicate that this is an object-specific ACE. This code example also specifies the `ADS_ACEFLAG_INHERIT_ACE` flag which indicates that the ACE can be inherited by child objects.

```

' Grant trustee the right to read the Telephone-Number property
' of all child objects in the Users container.
' {bf967a49-0de6-11d0-a285-00aa003049e2} is the schemaIDGUID of
' the Telephone-Number property.

bResult = SetRight objectDN, _
    ADS_RIGHT_WRITE_PROP Or ADS_RIGHT_READ_PROP, _
    ADS_ACETYPE_ACCESS_ALLOWED_OBJECT, _
    ADS_ACEFLAG_INHERIT_ACE, _
    "{bf967a49-0de6-11d0-a285-00aa003049e2}", _
    vbNullString, _
    "someone@fabrikam.com" ' Trustee

If bResult = True Then
    MsgBox ("The trustee can read the telephone number property.")
Else
    MsgBox ("An error occurred.")
End If

```

```

// Grant trustee the right to read the Telephone-Number property
// of all child objects in the Users container.
// {bf967a49-0de6-11d0-a285-00aa003049e2} is the schemaIDGUID of
// the Telephone-Number property.
hr = SetRight(
    pObject, // IADs pointer to the object.
    ADS_RIGHT_READ_PROP | ADS_RIGHT_WRITE_PROP,
    ADS_ACETYPE_ACCESS_ALLOWED_OBJECT,
    ADS_ACEFLAG_INHERIT_ACE,
    L"{bf967a49-0de6-11d0-a285-00aa003049e2}",
    NULL, // No inherited object type GUID.
    szTrustee
);

```

Grant or deny access to a set of properties on the object

This code example calls the **SetRight** function to grant the trustee the right to read or write a specific set of properties on the object. Specify **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT** to indicate that this is an object-specific ACE.

A property set is defined by a **controlAccessRight** object in the Extended Rights container of the Configuration partition. To identify the property set in the ACE, specify the **rightsGUID** property of a **controlAccessRight** object. Be aware that this property set GUID is also set in the **attributeSecurityGUID** property of every **attributeSchema** object included in the property set. For more information, see [Control Access Rights](#).

This code example also specifies inheritance flags that set the ACE as inheritable by child objects, but ineffective on the immediate object. In addition, the sample specifies the GUID of the User class, which indicates that the ACE can be inherited only by objects of that class.

```

' Grant trustee permission to read or write a set of properties.
' {77B5B886-944A-11d1-AEBD-0000F80367C1} is a GUID that identifies
' a property set.
' {bf967aba-0de6-11d0-a285-00aa003049e2} is a GUID that identifies the
' User class, so this ACE is inherited only by objects of that class.

bResult = SetRight objectDN, _
    ADS_RIGHT_READ_PROP Or ADS_RIGHT_WRITE_PROP, _
    ADS_ACETYPE_ACCESS_ALLOWED_OBJECT, _
    ADSACEFLAG_INHERIT_ACE Or ADSACEFLAG_INHERIT_ONLY_ACE, _
    "{77B5B886-944A-11d1-AEBD-0000F80367C1}", _
    "{bf967aba-0de6-11d0-a285-00aa003049e2}", _
    "someone@fabrikam.com" ' Trustee

If bResult = True Then
    MsgBox ("The trustee can read or write a set of properties.")
Else
    MsgBox ("An error occurred.")
End If

```

```

// Grant trustee the right to read or write a set of properties.
// {77B5B886-944A-11d1-AEBD-0000F80367C1} is a GUID that identifies
// a property set (rightsGUID of a controlAccessRight object).
// {bf967aba-0de6-11d0-a285-00aa003049e2} is the schemaIDGUID of the
// User class, so this ACE is inherited only by objects of that class.

hr = SetRight(
    pObject, // IADS pointer to the object.
    ADS_RIGHT_READ_PROP | ADS_RIGHT_WRITE_PROP,
    ADS_ACETYPE_ACCESS_ALLOWED_OBJECT,
    ADSACEFLAG_INHERIT_ACE | ADSACEFLAG_INHERIT_ONLY_ACE,
    L"{77B5B886-944A-11d1-AEBD-0000F80367C1}",
    L"{bf967aba-0de6-11d0-a285-00aa003049e2}",
    szTrustee
);

```

Grant or deny permission to create a specific type of child object

The following code example calls the **SetRight** function to grant a specified trustee the right to create and delete User objects in the subtree under the specified object. Be aware that the sample specifies the GUID of the User class, which means that the ACE allows only the trustee to create User objects, not objects of other classes. Specify **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT** to indicate that this is an object-specific ACE.

```

' Grant trustee the right to create or delete User objects
' in the specified object.
' {bf967aba-0de6-11d0-a285-00aa003049e2} is a GUID that identifies the
' User class.

bResult = SetRight objectDN, _
    ADS_RIGHT_DS_CREATE_CHILD Or ADS_RIGHT_DS_DELETE_CHILD, _
    ADS_ACETYPE_ACCESS_ALLOWED_OBJECT, _
    0, _
    "{bf967aba-0de6-11d0-a285-00aa003049e2}", _
    vbNullString, _
    "jeffsmith@fabrikam.com" 'trustee

If bResult = True Then
    MsgBox ("The trustee can create or delete User objects.")
Else
    MsgBox ("An error occurred.")
End If

```

```
// Grant trustee the right to create or delete User objects
// in the specified object.
// {bf967aba-0de6-11d0-a285-00aa003049e2} is the schemaIDGUID of the
// User class.
hr = SetRight(
    pObject, // IADs pointer to the object.
    ADS_RIGHT_DS_CREATE_CHILD | ADS_RIGHT_DS_DELETE_CHILD,
    ADS_ACETYPE_ACCESS_ALLOWED_OBJECT,
    0, // Not inheritable.
    L"{bf967aba-0de6-11d0-a285-00aa003049e2}",
    NULL, // No inherited object type GUID.
    szTrustee
);
```

For more information, and the **schemaIDGUID** of a predefined attribute or class, see the attribute or class reference page in the [Active Directory Schema](#) reference. For more information, and a code example that can be used to obtain a **schemaIDGUID** programmatically, see [Reading attributeSchema and classSchema Objects](#).

Setting Access Rights on the Entire Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

Certain permissions can be set only for an entire object, such as Delete and List Contents. Operation-specific permissions, such as the Read permission, can also be set for entire object so that they apply to an entire object.

The following procedure can be used to set permissions for an entire object.

To set permissions for an entire object

1. Set **IADsAccessControlEntry.AceType** to **ADS_ACETYPE_ACCESS_ALLOWED** or **ADS_ACETYPE_ACCESS_DENIED**.
2. Set **IADsAccessControlEntry.ObjectType** and **IADsAccessControlEntry.InheritedObjectType** to **NULL**.

For more information about how to create an ACE, see [Setting Access Rights on an Object](#).

For more information and a code example that can be used to set an ACE, see [Example Code for Setting an ACE on a Directory Object](#).

Example Code for Setting Read Property Rights on an Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example contains a function that creates an ACE that assigns read access to all properties of the object to the specified trustee.

```
*****  
CreateAceEffectiveReadAllProperties()  
  
Create an ACE that assigns read property rights to all properties on the  
object. This ACE is not inherited; that is, it applies only to the  
current object.  
*****  
  
HRESULT CreateAceEffectiveReadAllProperties(LPWSTR pwszTrustee,  
                                            IDispatch **ppDispACE)  
{  
    if(!pwszTrustee || !ppDispACE)  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr = E_FAIL;  
    CComPtr<IADsAccessControlEntry> spACE;  
  
    // Create the COM object for the new ACE.  
    hr = CoCreateInstance(CLSID_AccessControlEntry,  
                         NULL,  
                         CLSCTX_INPROC_SERVER,  
                         IID_IADsAccessControlEntry,  
                         (void **)&spACE);  
    if (SUCCEEDED(hr))  
    {  
        // Set the properties of the new ACE.  
  
        //Set the access mask containing the rights to assign.  
        //This function assigns read property rights.  
        hr = spACE->put_AccessMask(ADS_RIGHT_DS_READ_PROP);  
        if(FAILED(hr))  
        {  
            return hr;  
        }  
  
        // Set the trustee.  
        hr = spACE->put_Trustee(pwszTrustee);  
        if(FAILED(hr))  
        {  
            return hr;  
        }  
  
        // Set the AceType.  
        hr = spACE->put_AceType(ADS_ACETYPE_ACCESS_ALLOWED);  
        if(FAILED(hr))  
        {  
            return hr;  
        }  
  
        // For this function, set AceFlags so that ACE is not inherited by child objects.  
    }  
}
```

```
// For this function, set AceFlags so that ACE is not inherited by child objects.  
// You can set AceFlags to 0 or let it default to 0 by not calling put_AceFlags.  
hr = spACE->put_AceFlags(0);  
if(FAILED(hr))  
{  
    return hr;  
}  
  
// For this function, set ObjectType to NULL because the right applies to all properties  
// and set Flags to 0. You can also not call these two methods and let them default to NULL.  
hr = spACE->put_ObjectType(NULL);  
if(FAILED(hr))  
{  
    return hr;  
}  
  
hr = spACE->put_Flags(0);  
if(FAILED(hr))  
{  
    return hr;  
}  
  
// Is not inherited; set object type to NULL or let it default to NULL by not calling the method.  
hr = spACE->put_InheritedObjectType(NULL);  
if(FAILED(hr))  
{  
    return hr;  
}  
  
// QueryInterface for a IDispatch pointer to pass to the AddAce method.  
hr = spACE->QueryInterface(IID_IDispatch, (void**)ppDispACE);  
if(FAILED(hr))  
{  
    return hr;  
}  
}  
  
return hr;  
}
```

Setting Permissions to a Specific Property

6/3/2022 • 2 minutes to read • [Edit Online](#)

Permissions can be set to apply to a specific property of an object.

To set permissions that apply to a specific property of an object

1. Set the [IADsAccessControlEntry.AccessMask](#) property to `ADS_RIGHT_DS_READ_PROP` and/or `ADS_RIGHT_DS_WRITE_PROP`.
2. Set the [IADsAccessControlEntry.AceType](#) property to `ADS_ACETYPE_ACCESS_ALLOWED_OBJECT` or `ADS_ACETYPE_ACCESS_DENIED_OBJECT`.
3. Set the [IADsAccessControlEntry.ObjectType](#) property to the `schemaIDGUID` of the property. This is the `schemaIDGUID` of the `attributeSchema` object that defines the property in the schema. The GUID must be specified as a string of the form produced by the [StringFromGUID2](#) function in the COM library.
4. Set [IADsAccessControlEntry.Flags](#) to `ADS_FLAG_OBJECT_TYPE_PRESENT`.

For more information about the `schemaIDGUID` of a predefined attribute, see [Active Directory Domain Services Reference](#).

For more information and a code example that can be used to retrieve a `schemaIDGUID`, see [Reading attributeSchema and classSchema Objects](#).

For more information about how to create an ACE, see [Setting Access Rights on an Object](#).

For more information and a code example that can be used to set a property-specific ACE, see [Example Code for Setting an ACE on a Directory Object](#).

Setting Permissions on a Group of Properties

6/3/2022 • 2 minutes to read • [Edit Online](#)

Permissions can be applied to a group of properties. A property set is identified by the GUID in the **rightsGUID** attribute of a **controlAccessRight** object. This GUID is set in the **attributeSecurityGUID** attribute of the **attributeSchema** object of each attribute in the group.

The following procedure shows how to set permissions that apply to a group of object properties.

To set permissions that apply to a group of object properties

1. Set the **IADsAccessControlEntry.AccessMask** property to **ADS_RIGHT_DS_READ_PROP**, **ADS_RIGHT_DS_WRITE_PROP** or both values combined.
2. Set the **IADsAccessControlEntry.AceType** property to either **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT** or **ADS_ACETYPE_ACCESS_DENIED_OBJECT**.
3. Set the **IADsAccessControlEntry.ObjectType** property to the GUID of the property set. This is the **rightsGUID** property of the **controlAccessRight** object that identifies the property set. This GUID is also set as the **attributeSecurityGUID** in the **attributeSchema** object of each property in the group.
4. Set the **IADsAccessControlEntry.Flags** property to **ADS_FLAG_OBJECT_TYPE_PRESENT**.

Be aware that you should not set the **ADS_RIGHT_DS_CONTROL_ACCESS** flag in the **IADsAccessControlEntry.AccessMask** property. This flag is only used to specify a control access right.

For more information and a code example that can be used to set access rights for a property set, see [Example Code for Setting Permissions on a Group of Properties](#).

For more information about creating an ACE, see [Setting Access Rights on an Object](#).

For more information and a code example that can be used to set an ACE for a property set, see [Example Code for Setting an ACE on a Directory Object](#).

Example Code for Setting Permissions on a Group of Properties

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C and C++ code examples create an ACE that assigns read and write access to the attributes of the **Personal Information** property set of user objects to the specified trustee.

```
*****  
CreateAceChangePersonalInfoPropGroupOfUsers()  
  
Create an ACE that assigns change (Read/Write) property rights to the  
attributes of the Personal Information property set for user objects.  
For this function, the ACE is only inherited; therefore, it is not an  
effective right on the current object.  
*****/  
  
HRESULT CreateAceChangePersonalInfoPropGroupOfUsers(LPWSTR pwszTrustee,  
                                                 BOOL fAllowed,  
                                                 IDispatch **ppDispACE)  
{  
    if(!pwszTrustee || !ppDispACE)  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr;  
    CComPtr<IADSAccessControlEntry> spACE;  
  
    // Create the COM object for the new ACE.  
    hr = spACE.CoCreateInstance(CLSID_AccessControlEntry);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    // Set the properties of the new ACE.  
  
    /*  
    Set the access mask containing the rights to assign. This function assigns  
    ADS_RIGHT_DS_READ_PROP | ADS_RIGHT_DS_WRITE_PROP to control change.  
    */  
    hr = spACE->put_AccessMask(ADS_RIGHT_DS_READ_PROP | ADS_RIGHT_DS_WRITE_PROP);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    // Set the trustee.  
    hr = spACE->put_Trustee(CComBSTR(pwszTrustee));  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    // AceType must be ADS_ACETYPE_ACCESS_ALLOWED_OBJECT or ADS_ACETYPE_ACCESS_DENIED_OBJECT.  
    if(fAllowed)  
    {  
        hr = spACE->put_AceType(ADS_ACETYPE_ACCESS_ALLOWED_OBJECT);  
    }  
}
```

```

        }

        else
        {
            hr = spACE->put_AceType(ADS_ACETYPE_ACCESS_DENIED_OBJECT);
        }
        if(FAILED(hr))
        {
            return hr;
        }

        /*
        Set Flags to ADS_FLAG_OBJECT_TYPE_PRESENT | ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT
        so that the right applies only to a specific property of the specified
        object class.
        */
        hr = spACE->put_Flags(ADS_FLAG_OBJECT_TYPE_PRESENT | ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT);
        if(FAILED(hr))
        {
            return hr;
        }

        // Set ObjectType to the rightsGUID of the Personal Information controlAccessRight object.
        hr = spACE->put_ObjectType(CComBSTR("{77B5B886-944A-11d1-AEBD-0000F80367C1}"));
        if(FAILED(hr))
        {
            return hr;
        }

        /*
        For this function, set AceFlags so that ACE is inherited by child objects,
        but not effective on the current object. Set AceFlags to ADS_ACEFLAG_INHERIT_ACE
        and ADS_ACEFLAG_INHERIT_ONLY_ACE.
        */
        hr = spACE->put_AceFlags(ADS_ACEFLAG_INHERIT_ACE | ADS_ACEFLAG_INHERIT_ONLY_ACE);
        if(FAILED(hr))
        {
            return hr;
        }

        // Set InheritedObjectType to schemaIDGUID of the user class.
        hr = spACE->put_InheritedObjectType(CComBSTR("{BF967ABA-0DE6-11D0-A285-00AA003049E2}"));
        if(FAILED(hr))
        {
            return hr;
        }

        // Call the QueryInterface method for the IDispatch pointer to pass to the AddAce method.
        hr = spACE->QueryInterface(IID_IDispatch, (void**)ppDispACE);
        if(FAILED(hr))
        {
            return hr;
        }

        return hr;
    }
}

```

Setting Permissions on Child Object Operations

6/3/2022 • 2 minutes to read • [Edit Online](#)

Permissions, such as Create Child and Delete Child, can also be granted or denied for operations on all subobjects or subobjects of a specific class.

The following procedure can be used to set permissions for a specific subobject type.

To set permissions for a specific subobject type

1. Set the [IADsAccessControlEntry.AceType](#) property to **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT** or **ADS_ACETYPE_ACCESS_DENIED_OBJECT**.
2. Set the [IADsAccessControlEntry.ObjectType](#) property to the GUID for object class. This is the **schemaIDGUID** property of the classSchema object that defines the object class. If the **ObjectType** property is **NULL**, the ACE applies to subobjects of any class.
3. Set the [IADsAccessControlEntry.Flags](#) property to **ADS_FLAG_OBJECT_TYPE_PRESENT**.

For more information and a procedure for creating an ACE, see [Setting Access Rights on an Object](#).

For more information and a code example that can be used to set an ACE that controls child object operations, see [Example Code for Setting an ACE on a Directory Object](#).

Example Code for Setting Permissions on Child Object Operations

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C and C++ code example creates an ACE that assigns creation rights for user objects to the specified trustee.

```
*****  
CreateAceCreateUsers()  
  
Create an ACE that assigns the right to create User objects under the  
current object. For this function, the ACE is inherited by all subobjects  
and is an effective right on the current object.  
*****  
  
HRESULT CreateAceCreateUsers(LPWSTR pwszTrustee, BOOL fAllowed, IDispatch **ppDispACE)  
{  
    if(!pwszTrustee || !ppDispACE)  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr;  
    CComPtr<IADsAccessControlEntry> spACE;  
  
    // Create the COM object for the new ACE.  
    hr = spACE.CoCreateInstance(CLSID_AccessControlEntry);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    // Set the properties of the new ACE.  
  
    /*  
    Set the access mask that contains the rights to assign. This function  
    assigns rights to create objects.  
    */  
    hr = spACE->put_AccessMask(ADS_RIGHT_DS_CREATE_CHILD);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    // Set the trustee.  
    hr = spACE->put_Trustee(CComBSTR(pwszTrustee));  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    /*  
    The AceType property must be ADS_ACETYPE_ACCESS_ALLOWED_OBJECT or  
    ADS_ACETYPE_ACCESS_DENIED_OBJECT.  
    */  
    if(fAllowed)  
    {  
        hr = spACE->put_AceType(ADS_ACETYPE_ACCESS_ALLOWED_OBJECT);  
    }  
}
```

```

        }

        else
        {
            hr = spACE->put_AceType(ADS_ACETYPE_ACCESS_DENIED_OBJECT);
        }
        if(FAILED(hr))
        {
            return hr;
        }

        /*
        Set Flags to ADS_FLAG_OBJECT_TYPE_PRESENT so that the right applies to
        the creation of a specific object class within the current object and
        all its subobjects.
        */
        hr = spACE->put_Flags(ADS_FLAG_OBJECT_TYPE_PRESENT);
        if(FAILED(hr))
        {
            return hr;
        }

        // Set ObjectType to the schemaIDGUID of the user class so that the right
        // controls creation of user objects.
        hr = spACE->put_ObjectType(CComBSTR("BF967ABA-0DE6-11D0-A285-00AA003049E2"));
        if(FAILED(hr))
        {
            return hr;
        }

        // For this function, set AceFlags so that ACE is inherited by child objects
        hr = spACE->put_AceFlags(ADS_ACEFLAG_INHERIT_ACE);
        if(FAILED(hr))
        {
            return hr;
        }

        // Set InheritedObjectType to NULL so that it is inherited by all subobjects.
        hr = spACE->put_InheritedObjectType(NULL);
        if(FAILED(hr))
        {
            return hr;
        }

        // QueryInterface for the IDispatch pointer to pass to the AddAce method.
        hr = spACE->QueryInterface(IID_IDispatch, (void**)ppDispACE);
        if(FAILED(hr))
        {
            return hr;
        }

        return hr;
    }
}

```

How Security Descriptors are Set on New Directory Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

When you create a new object in Active Directory Domain Services, you can explicitly create a security descriptor and then set that security descriptor as the object's **nTSecurityDescriptor** property. For more information, see [Creating a Security Descriptor for a New Directory Object](#).

Active Directory Domain Services use the following rules to set the DACL in the new object's security descriptor:

- If you explicitly specify a security descriptor when you create the object, the system merges any inheritable ACEs from the parent object into the specified DACL unless the **SE_DACL_PROTECTED** bit is set in the security descriptor's control bits.
- If you do not specify a security descriptor, the system builds the object's DACL by merging any inheritable ACEs from the parent object into the default DACL from the **classSchema** object for the object's class.
- If the schema does not have a default DACL, the object's DACL is the default DACL from the primary or impersonation token of the creator.
- If there is no specified, inherited, or default DACL, the system creates the object with no DACL, which allows everyone full access to the object.

The system uses a similar algorithm to build a SACL for a directory service object.

The owner and primary group in the new object's security descriptor are set to the values you specify in the **nTSecurityDescriptor** property when you create the object. If you do not set these values, Active Directory Domain Services use the rules listed in the following table to set them.

RULE	DESCRIPTION
Owner	The owner in a default security descriptor is set to the default owner SID from the primary or impersonation token of the creating process. For most users, the default owner SID is the same as the SID that identifies the user's account. Be aware that for users who are members of the built-in administrators group, the system automatically sets the default owner SID in the access token to the administrators group; therefore, objects created by a member of the administrators group are typically owned by the administrators group. To get or set the default owner in an access token, call the GetTokenInformation or SetTokenInformation function with the TOKEN_OWNER structure.
Primary Group	The primary group in a default security descriptor is set to the default primary group from the creator's primary or impersonation token. Be aware that primary group is not used in the context of Active Directory Domain Services.

For more information about ACE inheritance, see [Inheritance and Delegation of Administration](#).

For more information about the default security descriptors in the schema, see [Default Security Descriptor](#).

For more information about **classSchema** objects, see [Active Directory Schema](#).

Creating Security Descriptors for new directory objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

You can use ADSI to create a security descriptor and set it as a new object's **nTSecurityDescriptor** property or use it to replace an existing object's **nTSecurityDescriptor** property.

To create a security descriptor for an object:

1. Use [CoCreateInstance](#) to create the ADSI COM object for the new security descriptor and get an [IADsSecurityDescriptor](#) interface pointer to that object. Be aware that the class ID is **CLSID_SecurityDescriptor**.
2. Use the [IADsSecurityDescriptor::put_Owner](#) method to set the owner of the object. The trustee is a user, group, or other security principal. An application should use the value from the appropriate property from the user or group object of the trustee to which to apply the ACE.
3. Use the [IADsSecurityDescriptor::put_Control](#) method to control whether DACLs and SACLs are inherited by the object from its parent container.
4. Use [CoCreateInstance](#) to create the ADSI COM object for the DACL for the new security descriptor and get an [IADsAccessControlList](#) interface pointer to that object. Be aware that the class ID is **CLSID_AccessControlList**.
5. For each ACE to add to the DACL, use [CoCreateInstance](#) to create the ADSI COM object for the new ACE and get an [IADsAccessControlEntry](#) interface pointer to that object. Be aware that the class ID is **CLSID_AccessControlEntry**.
6. For each ACE to add to the DACL, set the properties of the ACE using the property methods of the ACE's [IADsAccessControlEntry](#) object. For more information about the properties to set on an ACE, see [Setting Access Rights on an Object](#).
7. For each ACE to add to the DACL, use the [QueryInterface](#) method on the [IADsAccessControlEntry](#) object to get an [IDispatch](#) pointer. The [IADsAccessControlList::AddAce](#) method requires an [IDispatch](#) interface pointer to the ACE.
8. For each ACE to add to the DACL, use [IADsAccessControlList::AddAce](#) to add the new ACE to the DACL. Be aware that the order of the ACEs within the ACL can affect the evaluation of access to the object. The correct access to the object may require you to create a new ACL, add the ACEs from the existing ACL in the correct order to the new ACL, and then replace the existing ACL in the security descriptor with the new ACL. For more information, see [Order of ACEs in a DACL](#).
9. Follow Steps 4-8 to create the SACL for the new security descriptor.
10. Use the [IADsSecurityDescriptor::put_DiscretionaryAcl](#) method to set the DACL. For more information about DACLs, see [Null DACLs and Empty DACLs](#).
11. Use the [IADsSecurityDescriptor::put_SystemAcl](#) method to set the SACL.
12. Convert the [IADsSecurityDescriptor](#) object to a [VARIANT](#) by using the [QueryInterface](#) method of the [IADsSecurityDescriptor](#) object to obtain an [IDispatch](#) interface. Then set the **vt** member of the [VARIANT](#) to **VT_DISPATCH** and set the **pdispVal** member of the [VARIANT](#) equal to the [IDispatch](#) pointer.
13. Obtain an [IADs](#) interface pointer to the object.
14. Use the [IADs::Put](#) method with "nTSecurityDescriptor" and the [VARIANT](#) created above to write the new security descriptor to the property cache.
15. Use the [IADs::SetInfo](#) method to update the property on the object in the directory.

Inheritance and Delegation of Administration

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services supports the inheritance of permissions down the object tree to allow administration tasks to be performed at higher levels in the tree. This enables administrators to set up inheritable permissions on objects near the root, such as domain and organizational units, and have those permissions distributed to various objects in the tree.

Inheritance can be set on a per-ACE basis. The following table lists flags that can be specified in the AceFlags to control inheritance of the ACE.

FLAG	DESCRIPTION
ADS_ACEFLAG_INHERIT_ACE	Causes the ACE to be inherited down in the tree.
ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE	Causes the ACE to be inherited down only one level in the tree.
ADS_ACEFLAG_INHERIT_ONLY_ACE	Causes the ACE to be ignored on the object it is specified on, only be inherited down, and be effective where it has been inherited.

In addition to setting inheritance, Active Directory Domain Services supports object-specific inheritance. This allows the inheritable ACEs to be inherited down the tree, but be effective only on a specific type of object. This is extremely useful in delegating administration. For example, this can be used to set an object-specific inheritable ACE at an organizational unit that enables a group to have full control on all user objects in the organizational unit, but nothing else. Thereby, the management of users in that organizational unit gets delegated to the users in that group.

Delegating Service Administration with Security Groups

Use Security groups to define and delegate administrative roles associated with your application server. For example, your service may be associated with a group MyService Administrators. Users who are identified as the MyService administrators will be added to MyService Administrators group. The setup program for MyService can set ACLs on the directory to enable MyService Administrators sufficient permissions to read/write MyService-related attributes or create MyService-specific objects for example.

Roles in Security Groups for Computers Running Your Service

Use security groups to define the set of computers that are granted access to your service's objects in the directory. For example, your service may be associated with a group MyService Servers. All computers running the MyService server are added to MyService Servers group and this group can then be given access to parts of the directory where MyService servers need to read/write data. The setup program for MyService can set ACLs on the directory to enable MyService Servers sufficient permissions to read/write MyService-related attributes or create MyService-specific objects for example.

Access Control Inheritance

6/3/2022 • 2 minutes to read • [Edit Online](#)

Access-control entries (ACEs) in an object access-control list (ACL) can belong to one of two categories:

- Effective ACL: ACEs in this category apply to the object.
- Inherit ACL: ACEs in this category are inherited by objects created in the container.

Each ACE in the DACL can belong to one, or more, categories. The categories for where an ACE belongs are determined by the inheritance control flags set in the ACE.

Three inheritance-control flags can be set in the [AceFlags](#) property of an ACE.

FLAG	DESCRIPTION
ADS_ACEFLAG_INHERIT_ACE	This flag indicates that the ACE is part of the inherit ACL and that child objects inherit the inheritance control flags of this ACE.
ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE	This flag indicates that the ACE is part of the inherit ACL, but that no inheritance control flags are propagated to direct child objects (direct descendants) and the ACE is effective on the direct child objects.
ADS_ACEFLAG_INHERIT_ONLY_ACE	This flag indicates that the ACE is not part of effective ACL. If this flag is not set, then the ACE is part of the effective ACL. This flag is useful for setting permissions inheritable by subobjects, but do not affect accessibility of the container. For example, if an ACE is intended to be inherited by user objects in a organizational unit, it is likely that it should not be enforced for access to the organizational unit itself.

The `ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE` and `ADS_ACEFLAG_INHERIT_ONLY_ACE` flags are meaningful only if `ADS_ACEFLAG_INHERIT_ACE` is present. This is because the `ADS_ACEFLAG_INHERIT_ACE` flag adds inheritance behavior to an inheritable ACE, but does not define the type of inheritance. The `ADS_ACEFLAG_NO_PROPAGATE_INHERIT_ACE` and `ADS_ACEFLAG_INHERIT_ONLY_ACE` flags define a specific type of inheritance behavior.

It is important to remember that the system also sets the following flags based on the type and state of the ACE.

FLAG	DESCRIPTION
<code>ADS_ACEFLAG_INHERITED_ACE</code>	This flag indicates that the ACE was inherited.
<code>ADS_ACEFLAG_VALID_INHERIT_FLAGS</code>	This flag indicates that the inherit flags are valid.

The following table lists the effects of the different flag combinations for the [AceFlags](#) property of an ACE.

FLAG	EFFECT ON OBJECT CONTAINING THE ACE	EFFECT ON DIRECT CHILD OBJECTS	EFFECT ON OBJECTS BELOW DIRECT CHILDREN
No flags set.	Effective ACE: ACE applies to the object.	ACE is not inherited.	ACE is not inherited.

FLAG	EFFECT ON OBJECT CONTAINING THE ACE	EFFECT ON DIRECT CHILD OBJECTS	EFFECT ON OBJECTS BELOW DIRECT CHILDREN
ADS_ACEFLAG_INHERIT_ ACE	Effective ACE	ACE is inherited. ACE is an effective ACE.	ACE is inherited. ACE is an effective ACE.
ADS_ACEFLAG_INHERIT_ ACE ADS_ACEFLAG_INHERIT_ONLY_ ACE	Not an Effective ACE: ACE does not apply to the object.	ACE is inherited. ACE is an effective ACE.	ACE is inherited. ACE is an effective ACE.
ADS_ACEFLAG_INHERIT_ ACE ADS_ACEFLAG_NO_PRO_PAGATE_INHERIT_ ACE	Effective ACE	ACE is inherited but without inheritance flags. ACE is an Effective ACE	ACE is not inherited.
ADS_ACEFLAG_INHERIT_ ACE ADS_ACEFLAG_INHERIT_ONLY_ ACE ADS_ACEFLAG_NO_PRO_PAGATE_INHERIT_ ACE	Not an Effective ACE.	ACE is inherited but without inheritance flags. ACE is an Effective ACE.	ACE is not inherited.

Setting Rights to Specific Types of Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following procedure shows how to set an ACE that can be inherited only by a specific class of objects.

To set an ACE that can be inherited only by a specific class of objects

1. Set the [IADsAccessControlEntry.AceType](#) property to **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT** or **ADS_ACETYPE_ACCESS_DENIED_OBJECT**.
2. Set the [IADsAccessControlEntry.AceFlags](#) property to include the **ADSACEFLAG_INHERIT_ACE** flag.
3. Set the [IADsAccessControlEntry.InheritedObjectType](#) property to the **schemaIDGUID** of the object class that can inherit the ACE.
4. Set the [IADsAccessControlEntry.Flags](#) property to **ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT**.

IMPORTANT

Set **ADSACEFLAG_INHERIT_ACE** to cause the ACE to be inherited. In addition, you must set **ADSACEFLAG_INHERIT_ONLY_ACE** if the object type this ACE applies to does not match the object type of the container where the ACE is specified. If this is not done, the ACE will also become effective on the container and can grant unintended rights.

For more information and code examples that can be used to set this type of ACE, see [Example Code for Setting an ACE on a Directory Object](#).

Example Code for Setting Rights to Specific Types of Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C/C++ code example creates an ACE that assigns rights that are inherited by the specified type of object, but are not effective on the current object.

```
// Create an ACE that is inherited by child objects of the specified type,
// but does not apply to the current object.
// This ACE is also propagated to all descendants of the current object.
HRESULT CreateAceNoEffectiveInheritObject(
    LPWSTR pwszTrustee,
    long lAccessRights,
    long lAccessType,
    LPWSTR pwszObjectGUID,
    LPWSTR pwszInheritedObjectGUID,
    IDispatch **ppDispACE)
{
    HRESULT hr = E_FAIL;
    IADsAccessControlEntry *pACE = NULL;
    long lFlags = 0L;

    // Create the COM object for the new ACE.
    hr = CoCreateInstance( CLSID_AccessControlEntry,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_IADsAccessControlEntry,
        (void **) &pACE);
    if (SUCCEEDED(hr))
    {
        // Set the properties of the new ACE.

        // Set the access mask that contains the rights to assign.
        hr = pACE->put_AccessMask(lAccessRights);

        // Set the trustee.
        hr = pACE->put_Trustee(pwszTrustee);

        // Set the AceType.
        hr = pACE->put_AceType(lAccessType);

        /*
        For this function, set AceFlags so that ACE is inherited by child
        objects, but not effective on the current object.
        */

        // Set AceFlags to ADS_ACEFLAG_INHERIT_ACE and ADS_ACEFLAG_INHERIT_ONLY_ACE.
        hr = pACE->put_AceFlags(ADS_ACEFLAG_INHERIT_ACE | ADS_ACEFLAG_INHERIT_ONLY_ACE);

        /*
        If an szObjectGUID is specified, add ADS_FLAG_OBJECT_TYPE_PRESENT flag
        to the lFlags mask and set the ObjectType.
        */
        if (pwszObjectGUID)
        {
            lFlags |= ADS_FLAG_OBJECT_TYPE_PRESENT;
            hr = pACE->put_ObjectType(pwszObjectGUID);
        }
    }
}
```

```
/*
If an szInheritedObjectGUID is specified, add
ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT flag to the lFlags mask and set
the InheritedObjectType.
*/
if (pwszInheritedObjectGUID)
{
    lFlags |= ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT;
    hr = pACE->put_InheritedObjectType(pwszInheritedObjectGUID);
}

// Set flags if ObjectType or InheritedObjectType were set.
if (lFlags)
{
    hr = pACE->put_Flags(lFlags);
}

// QueryInterface for a IDispatch pointer to pass to the AddAce method.
hr = pACE->QueryInterface(IID_IDispatch, (void**)ppDispACE);
}

return hr;
}
```

Setting Rights to Specific Properties of Specific Types of Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

Property-specific permissions can be used in combination with object specific inheritance to provide the powerful and detailed delegation of administration. You can set a property-specific object-inheritable ACE to allow a specified user or group to read and/or write a specific attribute on a specified class of child objects in a container. For example, you can set an ACE on an organizational unit (OU) to enable a group to read and write the telephone number attribute of all user objects in the OU.

To set property-specific object-inheritable ACEs

1. Set [IADsAccessControlEntry.AceType](#) to `ADS_ACETYPE_ACCESS_ALLOWED_OBJECT` or `ADS_ACETYPE_ACCESS_DENIED_OBJECT`.
2. Set [IADsAccessControlEntry.ObjectType](#) to the `schemaIDGUID` of the attribute. For example, the `schemaIDGUID` of the `telephoneNumber` attribute is `{bf967a49-0de6-11d0-a285-00aa003049e2}`.
3. Set [IADsAccessControlEntry.AceFlags](#) to `ADSACEFLAG_INHERIT_ACE`.
4. Set [IADsAccessControlEntry.InheritedObjectType](#) to the `schemaIDGUID` of the object class that can inherit the ACE. For example, the `schemaIDGUID` of the `user` class is `{bf967aba-0de6-11d0-a285-00aa003049e2}`.
5. Set [IADsAccessControlEntry.Flags](#) to `ADS_FLAG_OBJECT_TYPE_PRESENT` and `ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT`.

IMPORTANT

Set `ADSACEFLAG_INHERIT_ACE` to cause the ACE to be inherited. In addition, set `ADSACEFLAG_INHERIT_ONLY_ACE` if the object type this ACE applies to does not match the object type of the container where the ACE is specified. If this is not done, the ACE will also become effective on the container and can grant unexpected rights.

For more information and code examples that can be used to set this kind of ACE, see [Example Code for Setting an ACE on a Directory Object](#).

Protecting Objects from the Effects of Inherited Rights

6/3/2022 • 2 minutes to read • [Edit Online](#)

As discussed in the topic [Inheritance and Delegation of Administration](#), ACEs can be set on a container object, such as an **organizationalUnit**, **domainDNS**, **container**, and so on, and propagated to child objects based on the ACE flags set on those ACEs.

If you have a secure object or an object whose ACEs you want to explicitly control, such as a private OU or a special user, you can prevent ACEs from being propagated to the object by its parent container or its parent container's predecessors.

Use the **IADsSecurityDescriptor.Control** property to control whether DACLs and SACLs are inherited by the object from its parent container.

The **IADsSecurityDescriptor.Control** property can be used to protect an object from the effects of inherited ACEs. The following flags force access control to be set explicitly on the object and prevent a user from modifying access control to the object by setting inheritable ACEs on the object's parent container, or its parent container's predecessors.

FLAG	DESCRIPTION
SE_DACL_PROTECTED	Prevents ACEs set on the DACL of the parent container, and any objects above the parent container in the directory hierarchy, from being applied to the object DACL.
SE_SACL_PROTECTED	Prevents ACEs set on the SACL of the parent container, and any objects above the parent container in the directory hierarchy, from being applied to the object SACL.

Be aware that the **SE_DACL_PRESENT** flag must be present to set **SE_DACL_PROTECTED** and **SE_SACL_PRESENT** must be present to set **SE_SACL_PROTECTED**.

Example Code for Setting and Removing SACL and DACL Protection

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes a code example used to set and remove SACL and DACL protection.

The following C and C++ code example sets and removes the `SE_DACL_PROTECTED` and `SE_SACL_PROTECTED` elements in the `IADsSecurityDescriptor.Control` property of an object security descriptor.

```
*****
SetSDInheritProtect()

This function sets and removes the SE_DACL_PROTECTED and
SE_SACL_PROTECTED elements in the Control property. Valid values for
lControl are:
    0 - Remove both SE_DACL_PROTECTED and SE_SACL_PROTECTED if they are
        set.

    SE_DACL_PROTECTED - Add SE_DACL_PROTECTED and remove SE_SACL_PROTECTED.

    SE_SACL_PROTECTED - Add SE_SACL_PROTECTED and remove SE_DACL_PROTECTED.

Be aware that SE_DACL_PRESENT must be present to set SE_DACL_PROTECTED
and SE_SACL_PRESENT must be present to set SE_SACL_PROTECTED.

*****
HRESULT SetSDInheritProtect(IADs *pObject, long lControl)
{
    if(!pObject)
    {
        return E_INVALIDARG;
    }

    HRESULT hr = E_FAIL;
    CComVariant svar;
    long lSetControl;
    long lOriginalControl;

    // Get the nTSecurityDescriptor
    CComBSTR sbstrAttribute = "nTSecurityDescriptor";
    hr = pObject->Get(sbstrAttribute, &svar);
    if(FAILED(hr))
    {
        return hr;
    }

    /*
    The type should be VT_DISPATCH which is an IDispatch pointer to the security
    descriptor object.
    */
    if(svar.vt != VT_DISPATCH)
    {
        return E_FAIL;
    }

    /*
    Get the IDispatch pointer from VARIANT structure and call the QueryInterface method for
    
```

```

the IADsSecurityDescriptor pointer.

*/
CComPtr<IADsSecurityDescriptor> spSD;
hr = svar.pdispVal->QueryInterface(IID_IADsSecurityDescriptor, (void**)&spSD);
if(FAILED(hr))
{
    return hr;
}

// Get the Control property.
hr = spSD->get_Control(&lSetControl);
if(FAILED(hr))
{
    return hr;
}

// Save the original value to see if the value changes.
lOriginalControl = lSetControl;

if(lControl & SE_DACL_PROTECTED)
{
    lSetControl |= SE_DACL_PROTECTED;
    lSetControl &= !SE_SACL_PROTECTED;
}
else if(lControl & SE_SACL_PROTECTED)
{
    lSetControl |= SE_SACL_PROTECTED;
    lSetControl &= !SE_DACL_PROTECTED;
}
else
{
    lSetControl &= !SE_DACL_PROTECTED;
    lSetControl &= !SE_SACL_PROTECTED;
}

/*
If there was change to the Control property, write it to the Security
Descriptor, write the SD to object, and then call SetInfo to write the
object to the directory.
*/
if(lOriginalControl != lSetControl)
{
    hr = spSD->put_Control(lSetControl);
    if(SUCCEEDED(hr))
    {
        hr = pObject->Put(sbstrAttribute, svar);
        if(SUCCEEDED(hr))
        {
            hr = pObject->SetInfo();
        }
    }
}

return hr;
}

```

Default Security Descriptor

6/3/2022 • 2 minutes to read • [Edit Online](#)

With Active Directory Domain Services, you can also specify default security for each type of object. This is specified in the [defaultSecurityDescriptor](#) attribute in the [classSchema](#) object definition in the [Active Directory schema](#). This security descriptor is used to provide default protection on the object if there is no security descriptor specified during the creation of the object.

NOTE

ACEs from a default security descriptor are handled as if they were specified as part of object creation. Therefore, the default ACEs are placed preceding inherited ACEs and override them as appropriate. For more information, see [Order of ACEs in a DACL](#).

The [defaultSecurityDescriptor](#) is specified in a special string format using the [Security Descriptor Definition Language](#) (SDDL). Two functions can be used to convert binary form of the security descriptor to string format and vice versa. These functions are:

- [ConvertSecurityDescriptorToStringSecurityDescriptor](#)
- [ConvertStringSecurityDescriptorToSecurityDescriptor](#)

For more information and the default security descriptors of the predefined object classes, see the class reference pages in the Active Directory Schema Reference of the [Active Directory Domain Services Reference](#).

For more information and a code example that reads or modifies the [defaultSecurityDescriptor](#) property of an object class, see [Reading the defaultSecurityDescriptor for an Object Class](#) and [Modifying the defaultSecurityDescriptor for an Object Class](#).

Reading the defaultSecurityDescriptor for an Object Class

6/3/2022 • 2 minutes to read • [Edit Online](#)

Using ADSI, you can obtain the **defaultSecurityDescriptor** attribute for an object class with the **IADs** interface. To obtain the **defaultSecurityDescriptor** attribute for an object class, perform the following steps.

1. Get an **IADs** interface pointer to the **classSchema** object for the object class.
2. Use the **IADs.Get** method to get the default security descriptor of the object. The name of the property that contains the security descriptor is "defaultSecurityDescriptor". The property will be returned as a **VARIANT** containing a BSTR with the default security descriptor in SDDL string format.
3. Use the **ConvertStringSecurityDescriptorToSecurityDescriptor** function to convert the SDDL string form to a security descriptor.
4. Use the **GetSecurityDescriptorDacl**, **GetSecurityDescriptorSacl**, **GetSecurityDescriptorOwner**, and **GetSecurityDescriptorControl** Security APIs to read the parts of the security descriptor.

For a code example that demonstrates how to do this, see [Example Code for Reading defaultSecurityDescriptor](#).

Example Code for Reading defaultSecurityDescriptor

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example reads the **defaultSecurityDescriptor** for a specified object class.

```
// Add adsiid.lib to your project
// Add activeds.lib to your project

#include <stdio.h>
#include "stdafx.h"
#include "iads.h"
#include "adshlp.h"

// Entry point to the application
int wmain(int argc, WCHAR* argv[])
{
    HRESULT hr;
    IADsContainer *pAbsSchema = NULL; // For the abstract schema
    IADsClass *pClass = NULL; // For class objects
    IADsProperty *pProp = NULL; // For attribute objects
    IADsSyntax *pSyntax = NULL; // For syntax objects
    IEnumVARIANT *pEnum = NULL;
    ULONG lFetch;
    VARIANT var;
    VARIANT_BOOL bMulti, bAbstract, bAux;
    BSTR bstrPI, bstrClass, bstrName;
    LONG lCount = 0;
    LONG lVarType = 0;
    IADs *pChild = NULL;
    DWORD dwUnknownClass = 0;

    CoInitialize(NULL);

    // Bind to the abstract schema.
    hr = ADsGetObject(L"LDAP://schema",
                      IID_IADsContainer,
                      (void**)&pAbsSchema);
    if (FAILED(hr))
        goto cleanup;

    // Enumerate the attribute and class entries in the abstract schema.
    hr = ADsBuildEnumerator(pAbsSchema, &pEnum);
    if (FAILED(hr))
        goto cleanup;

    VariantInit(&var);
    hr = ADsEnumerateNext(pEnum, 1, &var, &lFetch);
    while( hr == S_OK && lFetch == 1 )
    {
        // Identify whether this is a class, attribute, or syntax.
        hr = V_DISPATCH(&var)->QueryInterface(IID_IADs,
                                                (void**) &pChild);
        if (FAILED(hr))
            goto cleanuploop;
        hr = pChild->get_Class(&bstrClass);
        if (FAILED(hr))
            goto cleanuploop;
        wprintf(L"%s", bstrClass );
        hr = pChild->get_Name(&bstrName);
        if (FAILED(hr))
```

```

        goto cleanuploop;
wprintf(L"%s", bstrName);

// Retrieve data, depending on the type of schema element.
if (_wcsicmp(L"Class", bstrClass)==0)
{
    hr = pChild->QueryInterface(IID_IADsClass,
                                (void**) &pClass);
    if (FAILED(hr))
        goto cleanuploop;
    pClass->get_Abstract(&bAbstract);
    pClass->get_Auxiliary(&bAux);
    if (bAbstract)
        wprintf(L",Abstract");
    else if (bAux)
        wprintf(L",Auxiliary");
    else
        wprintf(L",Structural");

    // Retrieve the primary ADSI
    // interface to use with this class.
    pClass->get_PrimaryInterface(&bstrPI);
    if (_wcsicmp(L"{FD8256D0-FD15-11CE-ABC4-02608C9E7553}",
                bstrPI)==0)
        wprintf(L",IID_IADS,%s", bstrPI);
    if (_wcsicmp(L"{B15160D0-1226-11CF-A985-00AA006BC149}",
                bstrPI)==0)
        wprintf(L",IID_IADsPrintQueue,%s", bstrPI);
    if (_wcsicmp(L"{A2F733B8-EFFE-11CF-8ABC-00C04FD8D503}",
                bstrPI)==0)
        wprintf(L",IID_IADsOU,%s", bstrPI);
    if (_wcsicmp(L"{A05E03A2-EFFE-11CF-8ABC-00C04FD8D503}",
                bstrPI)==0)
        wprintf(L",IID_IADsLocality,%s", bstrPI);
    if (_wcsicmp(L"{3E37E320-17E2-11CF-ABC4-02608C9E7553}",
                bstrPI)==0)
        wprintf(L",IID_IADsUser,%s", bstrPI);
    if (_wcsicmp(L"{27636B00-410F-11CF-B1FF-02608C9E7553}",
                bstrPI)==0)
        wprintf(L",IID_IADsGroup,%s", bstrPI);
    if (_wcsicmp(L"{00E4C220-FD16-11CE-ABC4-02608C9E7553}",
                bstrPI)==0)
        wprintf(L",IID_IADsDomain,%s", bstrPI);
    SysFreeString(bstrPI);
}
else if (_wcsicmp(L"Property",bstrClass)==0)
{
    hr = pChild->QueryInterface(IID_IADsProperty,
                                (void**) &pProp);
    if (FAILED(hr))
        goto cleanuploop;
    pProp->get_MultiValued(&bMulti);
    wprintf(L"%s", bMulti ? L"Multi-Valued" : L"Single-Valued");
}
else if (_wcsicmp(L"Syntax", bstrClass)==0)
{
    hr = pChild->QueryInterface(IID_IADsSyntax,
                                (void**) &pSyntax);
    if ( FAILED(hr) )
        goto cleanuploop;
    pSyntax->get_OleAutoDataType (&lVarType);
    wprintf(L"%u", lVarType);
}
else
    dwUnknownClass++;

cleanuploop:
    wprintf(L"\n");
    SysFreeString(bstrClass);
}

```

```
    SysFreeString(bstrName);
    pChild->Release();
    VariantClear(&var);
    if (SUCCEEDED(hr))
        hr = ADsEnumerateNext( pEnum, 1, &var, &lFetch );
    }

    wprintf(L"dwUnknownClass: %u\n", dwUnknownClass);
cleanup:
    if (pAbsSchema)
        pAbsSchema->Release();
    if (pEnum)
        pEnum->Release();
    VariantClear(&var);
    CoUninitialize();

    return hr;
}
```

Modifying the defaultSecurityDescriptor for an Object Class

6/3/2022 • 3 minutes to read • [Edit Online](#)

The following code example retrieves the default security descriptor for an object class, adds an ACE to the DACL, and then sets the modified security descriptor on the object class.

Be aware that schema modification is disabled, by default, on all Windows 2000 domain controllers. To enable schema modification at a particular DC, set a REG_DWORD value named "Schema Update Allowed" under the following registry key:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\NTDS\Parameters

Add this value if it does not already exist. Set this value to 1 to enable schema modification. If this value is zero, schema modification is disabled. The Schema Manager MMC snap-in provides a check box that selects or clears this registry key.

```
// Add msrvct.dll to your project
// Add activeds.lib to your project
// Add adsiid.lib to your project

#include "stdafx.h"
#include <wchar.h>
#include <objbase.h>
#include <activeds.h>
#include <ACLAPIO.h>
#include <winnt.h>
#include <Sddl.h>
#include "atibase.h"
#include "stdio.h"
#include "iads.h"

#define _WIN32_WINNT 0x0500

HRESULT ModifyDefaultSecurityDescriptor( IADS *pObject );

// Entry point for the application.
int main(int argc, char *argv[])
{
    LPOLESTR szPath = new OLECHAR[MAX_PATH];
    LPOLESTR pszBuffer = new WCHAR[MAX_PATH];
    HRESULT hr = S_OK;
    IADS *pObject = NULL;
    VARIANT var;

    wprintf(L"This program modifies the default security descriptor of an object class\n");
    wprintf(L"Specify the object class:");
    #ifdef _MBCS
    _getws_s(pszBuffer);
    if (!pszBuffer)
        return TRUE;
    wcscpy_s(szPath,L"LDAP://cn=");
    wcscat_s(szPath,pszBuffer);
    wcscat_s(szPath,L",");
    #endif _MBCS

    // Initialize COM.
    CoInitialize(NULL);
```

```

// Get rootDSE and the schema container DN. Bind to the
// current user's domain using current user's security context.

hr = ADsOpenObject(L"LDAP://rootDSE",
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                    IID_IADs,
                    (void**)&pObject);

if (SUCCEEDED(hr)) {
    hr = pObject->Get(CComBSTR("schemaNamingContext"), &var);
    if (SUCCEEDED(hr)) {
        #ifdef _MBCS
        wcscat_s(szPath,var.bstrVal);
        #endif _MBCS
        VariantClear(&var);
        if (pObject) {
            pObject->Release();
            pObject = NULL;
        }
        hr = ADsOpenObject(szPath,
                           NULL,
                           NULL,
                           ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                           IID_IADs,
                           (void**)&pObject);
        if (SUCCEEDED(hr)) {
            wprintf(L"Modify the default SD for the %s class\n", pszBuffer);
            hr = ModifyDefaultSecurityDescriptor( pObject );
        }
    }
}

if (FAILED(hr))
    wprintf(L"Failed with the following HRESULT: 0x%x\n", hr);

if (pObject)
    pObject->Release();

// Uninitialize COM.

CoUninitialize();
return TRUE;
}

HRESULT ModifyDefaultSecurityDescriptor(
    IADs *pObject
)
{
    HRESULT hr = E_FAIL;
    VARIANT var;
    PSECURITY_DESCRIPTOR pSDCNV = NULL;
    SECURITY_DESCRIPTOR SD = {0};
    DWORD dwSDSize = sizeof(SECURITY_DESCRIPTOR);
    PSID pOwnerSID = NULL;
    DWORD dwOwnerSIDSize = 0;
    PSID pGroupSID = NULL;
    DWORD dwGroupSIDSize = 0;
    PACL pDACL = NULL;
    DWORD dwDACLSize = 0;
    PACL pSACL = NULL;
    DWORD dwSACLSize = 0;
    BOOL bDaclPresent = FALSE;
    BOOL bDaclDefaulted = FALSE;
    PACL pOldDACL, pNewDACL;
    ULONG ulLen;
    EXPLICIT_ACCESS ea;
}

```

```

DWORD dwRes;

// Get the default security descriptor. Type should be VT_BSTR.

hr = pObject->Get(CComBSTR("defaultSecurityDescriptor"), &var);
if (FAILED(hr) || var.vt!=VT_BSTR ) {
    wprintf(L"Error getting default SD: 0x%x\n", hr );
    goto Cleanup;
}

wprintf(L"Old Default SD: %s\n", var.bstrVal);

// Convert the security descriptor string to a security descriptor.

if ( ! ConvertStringSecurityDescriptorToSecurityDescriptor (
        var.bstrVal, SDDL_REVISION_1, &pSDCNV, NULL ) ) {
    wprintf(L"Error converting string security descriptor: %d\n",
           GetLastError() );
    goto Cleanup;
}

// Convert self-relative security descriptor to absolute.
// First, get the required buffer sizes.

if (! MakeAbsoluteSD(pSDCNV, &SD, &dwSDSize,
                     pDACL, &dwDACLSIZE,
                     pSACL, &dwSACLSize,
                     pOwnerSID, &dwOwnerSIDSize,
                     pGroupSID, &dwGroupSIDSize) ) {

    // Allocate the buffers.

    pDACL = (PACL) GlobalAlloc(GPTR, dwDACLSIZE);
    pSACL = (PACL) GlobalAlloc(GPTR, dwSACLSize);
    pOwnerSID = (PACL) GlobalAlloc(GPTR, dwOwnerSIDSize);
    pGroupSID = (PACL) GlobalAlloc(GPTR, dwGroupSIDSize);
    if (! (pDACL && pSACL && pOwnerSID && pGroupSID) ) {
        wprintf(L"GlobalAlloc failed: %d\n", GetLastError() );
        goto Cleanup;
    }

    // Perform the conversion.

    if (! MakeAbsoluteSD(pSDCNV, &SD, &dwSDSize, pDACL, &dwDACLSIZE,
                         pSACL, &dwSACLSize, pOwnerSID, &dwOwnerSIDSize,
                         pGroupSID, &dwGroupSIDSize) ) {
        wprintf(L"MakeAbsoluteSD: %d\n", GetLastError() );
        goto Cleanup;
    }
}

// Get the DACL from the security descriptor.

if ( ! GetSecurityDescriptorDacl(&SD, &bDaclPresent,
                                 &pOldDACL, &bDaclDefaulted) ) {
    wprintf(L"GetSecurityDescriptorDacl failed: %d\n", GetLastError() );
    goto Cleanup;
}

// Initialize an EXPLICIT_ACCESS structure for the new ACE.
// The ACE grants Everyone the right to read properties.

ZeroMemory(&ea, sizeof(EXPLICIT_ACCESS));
ea.grfAccessPermissions = ADS_RIGHT_DS_READ_PROP;
ea.grfAccessMode = GRANT_ACCESS;
ea.grfInheritance= 0;
ea.Trustee.TrusteeForm = TRUSTEE_IS_NAME;
ea.Trustee.ptstrName = TEXT("Everyone");

```

```

// Create a new ACL that merges the new ACE into the existing DACL.

dwRes = SetEntriesInAcl(1, &ea, pOldDACL, &pNewDACL);
if (ERROR_SUCCESS != dwRes) {
    wprintf(L"SetEntriesInAcl Error %u\n", dwRes );
    goto Cleanup;
}

// Put the modified DACL into the security descriptor.

if (! SetSecurityDescriptorDacl(&SD, TRUE, pNewDACL, FALSE) ) {
    wprintf(L"SetSecurityDescriptorOwner failed: %d\n",
           GetLastError() );
    goto Cleanup;
}

// Convert the security descriptor back to string format.

VariantClear(&var);
if ( ! ConvertSecurityDescriptorToStringSecurityDescriptor (
    &SD, SDDL_REVISION_1,
    GROUP_SECURITY_INFORMATION | OWNER_SECURITY_INFORMATION |
    DACL_SECURITY_INFORMATION | SACL_SECURITY_INFORMATION,
    &var.bstrVal, &ulLen ) ) {
    wprintf(L"Error converting security descriptor to string: %d\n",
           GetLastError() );
    goto Cleanup;
}

wprintf(L"New default SD: %s\n", var.bstrVal);
V_VT(&var) = VT_BSTR;

// Use Put and SetInfo to set the modified security descriptor.

hr = pObject->Put(CComBSTR("defaultSecurityDescriptor"), var);
if (FAILED(hr)) {
    wprintf(L"Error putting default SD: 0x%x\n", hr );
    goto Cleanup;
}

hr = pObject->SetInfo();
if (FAILED(hr)) {
    wprintf(L"Error setting default SD: 0x%x\n", hr );
    goto Cleanup;
}

Cleanup:

VariantClear(&var);
if (pSDCNV)
    LocalFree(pSDCNV);
if (pDACL)
    GlobalFree(pDACL);
if (pSACL)
    GlobalFree(pSACL);
if (pOwnerSID)
    GlobalFree(pOwnerSID);
if (pGroupSID)
    GlobalFree(pGroupSID);
if (pNewDACL)
    LocalFree(pNewDACL);

return hr;
}

```


Control Access Rights (AD DS)

6/3/2022 • 3 minutes to read • [Edit Online](#)

All objects in Active Directory Domain Services support a standard set of access rights defined in the **ADS_RIGHTS_ENUM** enumeration. These access rights can be used in the Access Control Entries (ACEs) of an object's security descriptor to control access to the object; that is, to control who can perform standard operations, such as creating and deleting child objects, or reading and writing the object attributes. However, for some object classes, it may be desirable to control access in a way not supported by the standard access rights. To facilitate this, Active Directory Domain Services allow the standard access control mechanism to be extended through the **controlAccessRight** object.

Control access rights are used in three ways:

- For extended rights, which are special operations not covered by the standard set of access rights. For example, the user class can be granted a "Send As" right that can be used by Exchange, Outlook, or any other mail application, to determine whether a particular user can have another user send mail on their behalf. Extended rights are created on **controlAccessRight** objects by setting the **validAccesses** attribute to equal the **ADS_RIGHT_DS_CONTROL_ACCESS** (256) access right.
- For defining property sets, to enable controlling access to a subset of an object's attributes, rather than just to the individual attributes. Using the standard access rights, a single ACE can grant or deny access to all of an object's attributes or to a single attribute. Control access rights provide a way for a single ACE to control access to a set of attributes. For example, the user class supports the **Personal-Information** property set that includes attributes such as street address and telephone number. Property set rights are created on **controlAccessRight** objects by setting the **validAccesses** attribute to contain both the **ACTR_DS_READ_PROP** (16) and the **ACTRL_DS_WRITE_PROP** (32) access rights.
- For validated writes, to require that the system perform value checking, or validation, beyond that which is required by the schema, before writing a value to an attribute on a DS object. This ensures that the value entered for the attribute conforms to required semantics, is within a legal range of values, or undergoes some other special checking that would not be performed for a simple low-level write to the attribute. A validated write is associated to a special permission that is distinct from the "Write <attribute>" permission that would allow any value to be written to the attribute with no value checking performed. The validated write is the only one of the three control access rights that cannot be created as a new control access right for an application. This is because the existing system cannot be programmatically modified to enforce validation. If a control access right was set up in the system as a validated write, the **validAccesses** attribute on the **controlAccessRight** objects will contain the **ADS_RIGHT_DS_SELF** (8) access right.

There are only three validated writes defined in the Windows 2000 Active Directory schema:

- Self-Membership permission on a Group object, which allows the caller's account, but no other account, to be added or removed from a group's membership.
- Validated-DNS-Host-Name permission on a Computer object, which allows a DNS host name attribute that is compliant with the computer name and domain name to be set.
- Validated-SPN permission on a Computer object, which allows an SPN attribute which is compliant with the DNS host name of the computer to be set.

For convenience, each control access right is represented by a **controlAccessRight** object in the Extended-Rights container of the Configuration partition, even though property sets and validated writes are not considered to be extended rights. Because the Configuration container is replicated across the entire forest,

control rights are propagated across all domains in a forest. There are a number of predefined control access rights, and of course, custom access rights can also be defined.

All control access rights can be viewed as permissions in the ACL Editor.

For more information and a C++ and Visual Basic code example that sets an ACE to control read/write access to a property set, see [Example Code for Setting an ACE on a Directory Object](#).

For more information about using control access rights to control access to special operations, see:

- [Creating a Control Access Right](#)
- [Setting a Control Access Right ACE in an Object's ACL](#)
- [Checking a Control Access Right in an Object's ACL](#)
- [Reading a Control Access Right Set in an Object's ACL](#)

Creating a Control Access Right

6/3/2022 • 4 minutes to read • [Edit Online](#)

To add a control access right to an Active Directory server, create a [controlAccessRight](#) object in the Extended-Rights container of the Configuration partition. For more information and a code example, see [Example Code for Creating a Control Access Right](#). To use the control access right, you must complete a few more steps, depending on whether the control access right is for a special operation or a property set.

If you define a control access right for a property set, use the **rightsGUID** of the [controlAccessRight](#) object to identify the properties in the set. Every property is defined by an [attributeSchema](#) object in the Active Directory schema. The **attributeSecurityGUID** property of an [attributeSchema](#) object identifies the property set, if any, that the property belongs to. Be aware that the **attributeSecurityGUID** property is single-valued and stores the GUID in binary format (octet string syntax).

If you define a control access right to place restrictions on access to a particular operation, your application must perform the access check when a user attempts the operation.

To set up access check

1. Create a control access right that defines the type of access to the application or service. For more information, see the following code example.
2. Create an Active Directory Domain Services object that represents the application, service, or resource that you are protecting.
3. Add object ACEs to the DACL in the object security descriptor to grant or deny users or groups the control access right on that object. For more information, see [Setting a Control Access Right ACE in an Object's ACL](#).
4. When a user attempts to perform the operation, verify the user rights by passing the object security descriptor and the user access token to the [AccessCheckByTypeResultList](#) function. For more information, see [Checking a Control Access Right in an Object's ACL](#).

Based on the result of the access check on the object, the application or service can allow or deny the user access to the application or service.

When you create a [controlAccessRight](#) object, set the attributes, listed in the following table, to make the object a legal control access right that is recognized by Active Directory Domain Services and the Windows security systems.

ATTRIBUTE	DESCRIPTION
cn	A single-valued property that is the object's relative distinguished name (RDN) in the Extended-Rights container. The cn is the name of the access control right in Active Directory Domain Services.

ATTRIBUTE	DESCRIPTION
appliesTo	<p>A multi-valued property that lists the object classes that the access control right applies to. For example, the Send-As access control right lists the user and computer object classes in its appliesTo property. In the list, each object class is identified by the schemaIDGUID of its classSchema object. The GUIDs are stored as strings of the form produced by the StringFromGUID2 function in the COM library but without the starting and terminating braces ({}). For example, the following GUID is the schemaIDGUID for the computer class: bf967a86-0de6-11d0-a285-00aa003049e2.</p> <p>Be aware that the schemaIDGUID property of a classSchema object is stored as a binary GUID using the octet string syntax. To convert this octet string format to the string format used in the appliesTo property, use the StringFromGUID2 function and remove the braces from the returned string.</p> <p>For more information about the schemaIDGUID property of one of the predefined object classes, such as user or computer, see the class reference page in the Active Directory Schema Reference in the Active Directory Domain Services Reference. For more information and a code example that retrieves a schemaIDGUID from a classSchema object, see Reading attributeSchema and classSchema Objects.</p>
displayName	<p>The string used to display the access control right in user interfaces such as the Security property page and other places in the Active Directory Users and Computers MMC snap-in.</p>
rightsGuid	<p>A GUID that identifies the control access right in an ACE. The GUID is stored as a string of the form produced by the StringFromGUID2 function, but without the starting and terminating braces. Use Uuidgen.exe for some other utility to generate a GUID for the control access right. If you define a new property set, you use the rightsGuid of the controlAccessRight object to identify the properties in the set. For each property in the property set, set the property's attributeSecurityGUID value to the value of the property set's rightsGUID. A property's attributeSecurityGUID value is stored in the property's attributeSchema definition in the Active Directory schema. The attributeSecurityGUID property is single-valued and stores the GUID in binary format (octet string syntax).</p>
objectClass	<p>This attribute specifies controlAccessRight as the object class.</p>
validAccesses	<p>For property sets, set this attribute to 0x30 (ADS_RIGHT_DS_READ_PROP ADS_RIGHT_DS_WRITE_PROP). For control access rights, set this attribute to 0x100 (ADS_RIGHT_DS_CONTROL_ACCESS). The security property page recognizes control access rights only if the validAccesses attribute is set to the appropriate value. If zero, the control access right is ignored or not displayed by the security property page.</p>

Be aware that the predefined schema classes use the [localizationDisplayId](#) attribute of a

controlAccessRight object to specify a message identifier used to retrieve a localized display name from Dssec.dll. Do not set the **localizationDisplayId** attribute if you define a new **controlAccessRight** object.

Example Code for Creating a Control Access Right

6/3/2022 • 4 minutes to read • [Edit Online](#)

The following Visual Basic example creates a **controlAccessRight** object in the Extended-Rights container.

```
Dim ExContainer As IADsContainer
Dim rootdse As IADs
Dim ExRight As IADs

On Error GoTo CleanUp

Set rootdse = GetObject("LDAP://rootDSE")
configpath = rootdse.Get("configurationNamingContext")
Set ExContainer = GetObject("LDAP://cn=extended-rights," & configpath)

' Create the object, specifying the object class and the cn.
Set ExRight = ExContainer.Create("controlAccessRight", "cn=MyExRight")

' Set the classes that the right applies to.
' Specify the schemaIDGUID of the user and computer classes.
ExRight.PutEx ADS_PROPERTY_UPDATE, "appliesTo", _
    Array("bf967aba-0de6-11d0-a285-00aa003049e2", _
        "bf967a86-0de6-11d0-a285-00aa003049e2")

' Set the display name used in Security property pages and other UI.
ExRight.PutEx ADS_PROPERTY_UPDATE,
    "displayName",
    Array("My-Extended-Right")

' Set rightsGUID to a GUID generated by Uuidgen.exe.
ExRight.PutEx ADS_PROPERTY_UPDATE, "rightsGUID", _
    Array("64ad33ac-ea09-4ded-b798-a0585c50fd5a")

' Set validAccesses to indicate a control access right.
ExRight.PutEx ADS_PROPERTY_UPDATE, "validAccesses", &H100

ExRight.SetInfo

Exit Sub

CleanUp:
    MsgBox ("An error has occurred.")
    ExContainer = Nothing
    rootdse = Nothing
    ExRight = Nothing
```

The following C++ code example is a function that creates a **controlAccessRight** object in the Extended-Rights container. When you call this function, use the following format to specify the GUID string for the *pszRightsGUID* parameter.

```
L"b7b13123-b82e-11d0-af00-0000f80367c1"
```

The **ADSVALUE** array for the **appliesTo** property uses the same GUID format and sets the **dwType** member to **ADSTYPE_CASE_IGNORE_STRING**.

```
#define _WIN32_WINNT 0x0500
#include <windows.h>
```

```

#include <windows.h>
#include <stdio.h>
#include <activeds.h>

// *****
// CreateExtendedRight
// *****

HRESULT CreateExtendedRight(
    LPWSTR pszCommonName,      // cn property
    LPWSTR pszDisplayName,     // displayName property
    LPWSTR pszRightsGUID,     // rightsGUID property
    ADSVALUE *pAdsvAppliesTo, // array of GUIDs for appliesTo property
    int cAppliesTo )          // number of GUIDs in array
{
    HRESULT hr = E_FAIL;
    VARIANT var;
    LPOLESTR szADsPath = NULL;
    IADS *pRootDSE = NULL;
    IDirectoryObject *pExRights = NULL;
    UINT nSize = 0;
    WCHAR *lpszExtRights = L"LDAP://cn=Extended-Rights,"

    const int cAttributes = 6;    // Count of attributes that must be set to create a control access right.
    PADS_ATTR_INFO pAttributeEntries = new ADS_ATTR_INFO[cAttributes]; // array of attributes
    ADSVALUE adsvCN,
        adsvObjectClass,
        adsvDisplayName,
        adsvRightsGUID,
        adsvValidAccesses;

    LPOLESTR pszRightRelPath = new WCHAR[MAX_PATH];
    IDispatch *pNewObject = NULL;

    hr = ADsOpenObject(L"LDAP://rootDSE",
        NULL,
        NULL,
        ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
        IID_IADS,
        (void**)&pRootDSE);
    if (FAILED(hr)) {
        wprintf(L"Bind to rootDSE failed: 0x%x\n", hr);
        return hr;
    }

    // Get the DN to the config container.
    hr = pRootDSE->Get(CComBSTR("configurationNamingContext"), &var);
    if (SUCCEEDED(hr))
    {
        // Determine the buffer size required to store the ADsPath string
        // and allocate the buffer.
        nSize = wcslen(lpszExtRights) + wcslen(var.bstrVal) + 1;
        szADsPath = new OLECHAR[nSize];

        if (szADsPath == NULL)
        {
            wprintf(L"Buffer allocation failed.");
            goto cleanup;
        }

        // Build ADsPath string to Extended-Rights container
        wcsncpy_s(szADsPath,lpszExtRights,nSize);
        wcsncat_s(szADsPath,var.bstrVal,wcslen(var.bstrVal));

        // Get an IDirectory Object pointer to the Extended Rights Container.
        hr = ADsOpenObject(szADsPath,
            NULL,
            NULL,
            ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
            IID_IDirectoryObject,
            (void**)&pExRights);
    }
}

```

```

        (void**) &pExRights);
}

if (FAILED (hr) ) {
    wprintf(L"Bind to Extended Rights Container failed: 0x%lx\n", hr);
    goto cleanup;
}

// Set first attribute: CN
pAttributeEntries[0].pszAttrName = L"CN";                      // Attribute name: CN
pAttributeEntries[0].dwControlCode = ADS_ATTR_APPEND;           // Add the attribute.
pAttributeEntries[0].dwADsType = ADSTYPE_CASE_IGNORE_STRING;   // Attribute syntax is string.
// Fill in the ADSVALUE structure for the CN property
adsvCN.CaseIgnoreString = pszCommonName;
adsvCN.dwType = ADSTYPE_CASE_IGNORE_STRING;
pAttributeEntries[0].pADsValues = &adsvCN;
pAttributeEntries[0].dwNumValues = 1;

// Set second attribute: objectClass
pAttributeEntries[1].pszAttrName = L"objectClass";             // Attribute name: objectClass
pAttributeEntries[1].dwControlCode = ADS_ATTR_APPEND;           // Add the attribute.
pAttributeEntries[1].dwADsType = ADSTYPE_CASE_IGNORE_STRING;   // Attribute syntax is string.
// Fill in the ADSVALUE structure for the objectClass property
adsvObjectClass.CaseIgnoreString = L"controlAccessRight";     // objectClass is controlAccessRight
adsvObjectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
pAttributeEntries[1].pADsValues = &adsvObjectClass;
pAttributeEntries[1].dwNumValues = 1;

// Set third attribute: appliesTo
// Each value for this property is a schemaIDGUID of a class to which the right can be applied.
pAttributeEntries[2].pszAttrName = L"appliesTo";                // Attribute name: appliesTo
pAttributeEntries[2].dwControlCode = ADS_ATTR_APPEND;           // Add the attribute.
pAttributeEntries[2].dwADsType = ADSTYPE_CASE_IGNORE_STRING;   // Attribute syntax is string.
// The ADSVALUE array for this property is passed in as a parameter to this function.
pAttributeEntries[2].pADsValues = pAdsvAppliesTo;
pAttributeEntries[2].dwNumValues = cAppliesTo;

// Set fourth attribute: displayName
pAttributeEntries[3].pszAttrName = L"displayName";              // Attribute name:
CNpAttributeEntries[3].dwControlCode = ADS_ATTR_APPEND;         // Add the attribute.
pAttributeEntries[3].dwADsType = ADSTYPE_CASE_IGNORE_STRING;   // Attribute syntax is string.
// Fill in the ADSVALUE structure for the displayName property.
adsvDisplayName.CaseIgnoreString = pszDisplayName;
adsvDisplayName.dwType = ADSTYPE_CASE_IGNORE_STRING;
pAttributeEntries[3].pADsValues = &adsvDisplayName;
pAttributeEntries[3].dwNumValues = 1;

// Set fifth attribute: rightsGUID
pAttributeEntries[4].pszAttrName = L"rightsGUID";               // Attribute name
pAttributeEntries[4].dwControlCode = ADS_ATTR_APPEND;           // Add the attribute.
pAttributeEntries[4].dwADsType = ADSTYPE_CASE_IGNORE_STRING;   // Attribute syntax is string.
// Fill in the ADSVALUE structure for the rightsGUID property.
adsvRightsGUID.dwType = ADSTYPE_CASE_IGNORE_STRING;
adsvRightsGUID.CaseIgnoreString = pszRightsGUID;
pAttributeEntries[4].pADsValues = &adsvRightsGUID;
pAttributeEntries[4].dwNumValues = 1;

// Set sixth attribute: validAccesses
pAttributeEntries[5].pszAttrName = L"validAccesses";            // Attribute name
pAttributeEntries[5].dwControlCode = ADS_ATTR_APPEND;           // Add the attribute.
pAttributeEntries[5].dwADsType = ADSTYPE_CASE_IGNORE_STRING;   // Attribute syntax is string.
// Fill in the ADSVALUE structure for the rightsGUID property.
adsvValidAccesses.dwType = ADSTYPE_INTEGER;
adsvValidAccesses.Integer = ADS_RIGHT_DS_CONTROL_ACCESS;
pAttributeEntries[5].pADsValues = &adsvValidAccesses;
pAttributeEntries[5].dwNumValues = 1;

// Set up the relative distinguished name for the new object.
wcscpy_s(pszRightRelPath, L"cn=");
wcscat_s(pszRightRelPath, pszCommonName);

```

```

// Create the controlAccessRight
hr = pExRights->CreateDSObject(
    pszRightRelPath,    // Relative path of new object
    pAttributeEntries, // Attributes to be set
    cAttributes,        // Number of attributes being set
    &pNewObject         // receives IDispatch pointer to the new object
);

cleanup:

if (pRootDSE)
    pRootDSE->Release();
if (pExRights)
    pExRights->Release();
if (pNewObject)
    pNewObject->Release();
if (szADsPath)
    delete [] szADsPath;

VariantClear(&var);
return hr;
}

```

This **CreateExtendedRight** sample function can be called with the following code example.

```

ADSVALUE adsvalAppliesTo;

adsvalAppliesTo.dwType = ADSTYPE_CASE_IGNORE_STRING;
adsvalAppliesTo.CaseIgnoreString = L"bf967aba-0de6-11d0-a285-00aa003049e2";

hr = CreateExtendedRight(L"myexright", L"My Extended Right",
                        L"7587d479-441a-480b-9d5d-807b4d067db4",
                        &adsvalAppliesTo,
                        1);

```

Setting a Control Access Right ACE in an Object's ACL

6/3/2022 • 2 minutes to read • [Edit Online](#)

Using ADSI, you set a control access right ACE just as you would a property-specific ACE, except that the [IADsAccessControlEntry.ObjectType](#) property is the **rightsGUID** of the control access right. Be aware that you can also use the Win32 security APIs to set ACLs on directory objects.

The following table lists the [IADsAccessControlEntry](#) properties for control access rights that can be used to set properties for an ACE.

PROPERTY	DESCRIPTION
AccessMask	For control access rights that control extended rights access to special operations, AccessMask must contain the ADS_RIGHT_DS_CONTROL_ACCESS flag. For control access rights that define a property set, AccessMask contains ADS_RIGHT_DS_READ_PROP and/or ADS_RIGHT_DS_WRITE_PROP . For control access rights that control validated writes, AccessMask contains ADS_RIGHT_DS_SELF .
Flags	This value must include the ADS_FLAG_OBJECT_TYPE_PRESENT flag.
ObjectType	This value must be the StringFromGUID2 format of the rightsGUID attribute of the control access right. Be aware that, in an ACE, the GUID string must include the starting and terminating curly braces even though the rightsGUID attribute of the controlAccessRight object does not include the curly braces.
AceType	Either ADS_ACETYPE_ACCESS_ALLOWED_OBJECT to grant the trustee the access control right or ADS_ACETYPE_ACCESS_DENIED_OBJECT to deny the trustee the control access right.
Trustee	The security principal, for example user, group, computer, and so on, to which the ACE applies.

For more information about creating an ACE, see [Setting Access Rights on an Object](#).

For more information and a code example for setting an ACE, see [Example Code for Setting an ACE on a Directory Object](#).

Example Code for Setting a Control Access Right ACE

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C/C++ code example adds an ACE for a control access right to the ACL of an object.

```
*****  
SetExtendedRight()  
  
This function uses the rightsGUID in StringFromGUID2 format and assumes  
it is the GUID for the correct control access right. For control access  
rights in a DACL, lAccessType must be ADS_ACETYPE_ACCESS_ALLOWED_OBJECT  
or ADS_ACETYPE_ACCESS_DENIED_OBJECT.  
*****/  
  
HRESULT SetExtendedRight(IADs *pObject,  
                         LPWSTR pwszRightsGUID,  
                         LONG lAccessType,  
                         LONG fInheritanceFlags,  
                         LONG fAppliesToObjectType,  
                         LPWSTR pwszTrustee)  
  
{  
    if(!pObject || !pwszRightsGUID || !pwszTrustee)  
    {  
        return E_INVALIDARG;  
    }  
  
    if((lAccessType != ADS_ACETYPE_ACCESS_ALLOWED_OBJECT) &&  
       (lAccessType != ADS_ACETYPE_ACCESS_DENIED_OBJECT))  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr;  
  
    // Get the nTSecurityDescriptor attribute.  
    CComBSTR sbstrNTSecDesc = L"nTSecurityDescriptor";  
    CComVariant svarSecDesc;  
    hr = pObject->Get(sbstrNTSecDesc, &svarSecDesc);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    /*  
    The type should be VT_DISPATCH which is an IDispatch pointer to the  
    security descriptor object.  
    */  
    if(VT_DISPATCH != svarSecDesc.vt)  
    {  
        return E_FAIL;  
    }  
  
    // Get the IADsSecurityDescriptor interface from the IDispatch pointer.  
    CComPtr<IADsSecurityDescriptor> spSecDesc;  
    hr = svarSecDesc.pdispVal->QueryInterface(IID_IADsSecurityDescriptor, (void**)&spSecDesc);  
    if(FAILED(hr))  
    {  
        return E_FAIL;  
    }  
  
    // Set the control access right.  
    CComBSTR sbstrAccessRight = L"Control";  
    CComVariant vAccessRight;  
    vAccessRight.vt = VT_BSTR;  
    vAccessRight.bstrVal = sbstrAccessRight;  
    spSecDesc->SetAccessRight(pwszRightsGUID, lAccessType, fInheritanceFlags, fAppliesToObjectType, pwszTrustee, &vAccessRight);  
    if(FAILED(hr))  
    {  
        return E_FAIL;  
    }  
}
```

```

        return hr;
    }

    // Get the DACL object.
    CComPtr spDispDACL;
    hr = spSecDesc->get_DiscretionaryAcl(&spDispDACL);
    if(FAILED(hr))
    {
        return hr;
    }

    // Get the IADsAccessControlList interface from the DACL object.
    CComPtr spACL;
    hr = spDispDACL->QueryInterface(IID_IADsAccessControlList, (void**)&spACL);
    if(FAILED(hr))
    {
        return hr;
    }

    // Create the COM object for the new ACE.
    CComPtr spACE;
    hr = CoCreateInstance(CLSID_AccessControlEntry,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_IAccessControlEntry,
        (void **)&spACE);
    if(FAILED(hr))
    {
        return hr;
    }

    // Set the properties of the new ACE.

    /*
    For an extended control access right, set the mask to
    ADS_RIGHT_DS_CONTROL_ACCESS.
    */
    hr = spACE->put_AccessMask(ADS_RIGHT_DS_CONTROL_ACCESS);
    if(FAILED(hr))
    {
        return hr;
    }

    // Set the trustee.
    hr = spACE->put_Trustee(pwszTrustee);
    if(FAILED(hr))
    {
        return hr;
    }

    /*
    For extended control access rights, set AceType to
    ADS_ACETYPE_ACCESS_ALLOWED_OBJECT or ADS_ACETYPE_ACCESS_DENIED_OBJECT.
    */
    hr = spACE->put_AceType(lAccessType);
    if(FAILED(hr))
    {
        return hr;
    }

    /*
    For this example, set the AceFlags so that ACE is not inherited by child
    objects.
    */
    hr = spACE->put_AceFlags(fInheritanceFlags);
    if(FAILED(hr))
    {
        return hr;
    }
}

```

```

        }

        /*
Flags specifies whether the ACE applies to the current object, child objects,
or both. For this example, fAppliesToInheritedObject is set to
ADS_FLAG_OBJECT_TYPE_PRESENT so that the right applies only to the current
object.
*/
hr = spACE->put_Flags(fAppliesToObjectType);
if(FAILED(hr))
{
    return hr;
}

/*
For extended control access rights, set ObjectType to the rightsGUID of the
extended right.
*/
if(fAppliesToObjectType & ADS_FLAG_OBJECT_TYPE_PRESENT)
{
    hr = spACE->put_ObjectType(pwszRightsGUID);
    if(FAILED(hr))
    {
        return hr;
    }
}

// Set the inherited object type if right applies to child objects.
if(fAppliesToObjectType & ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT)
{
    hr = spACE->put_InheritedObjectType(pwszRightsGUID);
    if(FAILED(hr))
    {
        return hr;
    }
}

// Get the IDispatch pointer for the ACE.
CComPtr spDispACE;
hr = spACE->QueryInterface(IID_IDispatch, (void**)&spDispACE);
if(FAILED(hr))
{
    return hr;
}

// Add the ACE to the ACL.
hr = spACL->AddAce(spDispACE);
if(FAILED(hr))
{
    return hr;
}

// Update the DACL property.
hr = spSecDesc->put_DiscretionaryAcl(spDispDACL);
if(FAILED(hr))
{
    return hr;
}

/*
Write the updated value for the ntSecurityDescriptor attribute to the
property cache.
*/
hr = pObject->Put(sbstrNTSecDesc, svarSecDesc);
if(FAILED(hr))
{
    return hr;
}

// Call SetInfo to update the property on the object in the directory.

```

```
// Call SetInfo to update the property on the object in the directory.  
hr = pObject->SetInfo();
```

```
return hr;
```

```
}
```

Checking a Control Access Right in an Object's ACL

6/3/2022 • 2 minutes to read • [Edit Online](#)

To check a control access right on an object's ACL, use the [AccessCheckByTypeResultList](#) function. To use this function, an application requires a pointer to the [SECURITY_DESCRIPTOR](#) for the object instead of an [IADsSecurityDescriptor](#) interface to an ADSI security descriptor COM object.

Use the following steps to check access for an controlled access right on an object:

1. Get an [IDirectoryObject](#) interface pointer to the object.
2. Use the [IDirectoryObject::GetObjectAttributes](#) method to get the security descriptor of the object. The name of the property containing the security descriptor is [nTSecurityDescriptor](#). The property is returned as a pointer to a [SECURITY_DESCRIPTOR](#) structure.
3. Use the [SECURITY_DESCRIPTOR](#) structure with the [AccessCheckByTypeResultList](#) function to check the permissions for the specified control access right for the specified client.

The example code in [Example Code for Checking a Control Access Right in an Object's ACL](#) shows, in detail, how to do this.

Example Code for Checking a Control Access Right in an Object's ACL

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following example can be used to verify that the currently logged-on user has permissions for a control access right on the specified object.

```
// Define the Generic Mapping structure.  
// Generic read  
#define GENERIC_READ_MAPPING ((STANDARD_RIGHTS_READ) | \  
                           (ADS_RIGHT_ACTRL_DS_LIST) | \  
                           (ADS_RIGHT_DS_READ_PROP) | \  
                           (ADS_RIGHT_DS_LIST_OBJECT))  
  
// Generic execute  
#define GENERIC_EXECUTE_MAPPING ((STANDARD_RIGHTS_EXECUTE) | \  
                               (ADS_RIGHT_ACTRL_DS_LIST))  
  
// Generic right  
#define GENERIC_WRITE_MAPPING ((STANDARD_RIGHTS_WRITE) | \  
                            (ADS_RIGHT_DS_SELF) | \  
                            (ADS_RIGHT_DS_WRITE_PROP))  
  
// Generic all  
#define GENERIC_ALL_MAPPING ((STANDARD_RIGHTS_REQUIRED) | \  
                           (ADS_RIGHT_DS_CREATE_CHILD) | \  
                           (ADS_RIGHT_DS_DELETE_CHILD) | \  
                           (ADS_RIGHT_DS_DELETE_TREE) | \  
                           (ADS_RIGHT_DS_READ_PROP) | \  
                           (ADS_RIGHT_DS_WRITE_PROP) | \  
                           (ADS_RIGHT_ACTRL_DS_LIST) | \  
                           (ADS_RIGHT_DS_LIST_OBJECT) | \  
                           (ADS_RIGHT_DS_CONTROL_ACCESS) | \  
                           (ADS_RIGHT_DS_SELF))  
  
// Standard DS generic access rights mapping  
#define DS_GENERIC_MAPPING {GENERIC_READ_MAPPING, \  
                         GENERIC_WRITE_MAPPING, \  
                         GENERIC_EXECUTE_MAPPING, \  
                         GENERIC_ALL_MAPPING}  
  
HRESULT CheckExtendedRight(  
    HANDLE hToken,  
    IDirectoryObject *pObject,  
    CLSID pclsid,  
    DWORD *dwAccess  
)  
  
{  
    HRESULT hr = E_FAIL;  
    *dwAccess = FALSE;  
    BOOL bSuccess = FALSE;  
    PADS_ATTR_INFO pAttrInfo = NULL;  
    DWORD dwReturn= 0;  
    LPWSTR pAttrNames[] = {L"nTSecurityDescriptor", L"objectSid"};  
    PSECURITY_DESCRIPTOR pSD = NULL;  
    DWORD SDSize;  
    VOID *pAbsoluteSD = NULL;  
    DWORD AbsoluteSDSize = 0;  
    VOID *pDacl = NULL;  
    DWORD DaclSize = 0;  
    VOID *pSacl = NULL;  
    DWORD SaclSize = 0;  
    VOID *pOwner = NULL;  
    DWORD OwnerSize = 0;
```

```

DWORD OwnerSize = 0;
VOID *pGroup = NULL;
DWORD GroupSize = 0;
PSID pSID = NULL;
UINT nGUIDLength = 0;

// Get attributes for security descriptor and SID.
hr = pObject->GetObjectAttributes( pAttrNames,
                                    2,
                                    &pAttrInfo,
                                    &dwReturn );
if ( (SUCCEEDED(hr)) && (dwReturn>0) )
{
    for(DWORD idx=0; idx < dwReturn;idx++, pAttrInfo++)
    {
        // Verify the attribute name.
        if ( _wcsicmp(pAttrInfo->pszAttrName,
                      L"nTSecurityDescriptor") == 0 )
        {
            // Check the attribute type.
            if (pAttrInfo->dwADsType==ADSTYPE_NT_SECURITY_DESCRIPTOR)
            {
                pSD = (PSECURITY_DESCRIPTOR)(pAttrInfo->pADsValues->SecurityDescriptor.lpValue);
                SDSize =
                    (pAttrInfo->pADsValues->SecurityDescriptor.dwLength);
            }
        }
        if ( _wcsicmp(pAttrInfo->pszAttrName,L"objectSID") == 0 )
        {
            // Verify the attribute type.
            if (pAttrInfo->dwADsType==ADSTYPE_OCTET_STRING)
            {
                pSID =
                    (PSID)(pAttrInfo->pADsValues->OctetString.lpValue);
            }
        }
    }
    OBJECT_TYPE_LIST sObjectList;
    sObjectList.Level = ACCESS_OBJECT_GUID;
    sObjectList.Sbz = 0;

    sObjectList.ObjectType = (GUID*)&pclsid;

    CHAR PrivilegeSetBuffer[256];
    PRIVILEGE_SET *PrivilegeSet = (PRIVILEGE_SET *)PrivilegeSetBuffer;
    DWORD dwPrivSetSize = sizeof( PrivilegeSetBuffer );
    DWORD GrantedAccess = 0;
    ZeroMemory(PrivilegeSetBuffer, 256);
    DWORD DesiredAccess = ADS_RIGHT_DS_CONTROL_ACCESS;
    // Use the GENERIC_MAPPING structure to convert any
    // generic access rights to object-specific access rights.
    GENERIC_MAPPING GenericMapping = DS_GENERIC_MAPPING;
    // Before calling AccessCheck, a convert must be performed
    // security descriptor into Absolute form.
    if( ! MakeAbsoluteSD(
            pSD,
            (PSECURITY_DESCRIPTOR)pAbsoluteSD,
            &AbsoluteSDSize,
            (PACL)pDacl,
            &DaclSize,
            (PACL)pSacl,
            &SaclSize,
            (PSID)pOwner,
            &OwnerSize,
            (PSID)pGroup,
            &GroupSize
            ))
    {
        pAbsoluteSD =

```

```

        (PSECURITY_DESCRIPTOR)LocalAlloc(0,AbsoluteSDSize);
    if(!pAbsoluteSD)
    {
        // TODO: handle this.
    }
    pDacl = (PACL)LocalAlloc(0,DaclSize);
    if(!pDacl)
    {
        // TODO: handle this.
    }
    pSacl = (PACL)LocalAlloc(0,SaclSize);
    if(!pSacl)
    {
        // TODO: handle this.
    }
    pOwner = (PSID)LocalAlloc(0,OwnerSize);
    if(!pOwner)
    {
        // TODO: handle this.
    }
    pGroup = (PSID)LocalAlloc(0,GroupSize);
    if(!pGroup)
    {
        // TODO: handle this.
    }
    if( ! MakeAbsoluteSD(
        pSD,
        (PSECURITY_DESCRIPTOR)pAbsoluteSD,
        &AbsoluteSDSize,
        (PACL)pDacl,
        &DaclSize,
        (PACL)pSacl,
        &SaclSize,
        (PSID)pOwner,
        &OwnerSize,
        (PSID)pGroup,
        &GroupSize
    ))
{
    //
    // TODO: handle this.
    //
    // Cleanup and return.
    if (pAttrInfo)
        FreeADSMem( pAttrInfo );
    return E_FAIL;
}

bSuccess = AccessCheckByTypeResultList(
    pSD,           // Security descriptor
    pSID,          // SID of the verified object
    hToken,         // Handle to client access token
    DesiredAccess, // Requested access rights
    &sObjectList,   // An array of object types
    1,             // Number of object type elements
    &GenericMapping, // Map generic to specific rights
    PrivilegeSet,   // Receives privileges used
    &dwPrivSetSize, // Size of privilege-set buffer
    &GrantedAccess, // Retrieves mask of granted rights
    dwAccess        // Retrieves results of
                    // access verification
);
// Verify that access check function call succeeded.
if(bSuccess)
{
    hr = S_OK;
}

```

```
    else
        hr = E_FAIL;
}
// Use FreeADsMem for all memory obtained from ADSI call.
if (pAttrInfo)
    FreeADsMem( pAttrInfo );

return hr;
}
```

Reading a Control Access Right Set in an Object's ACL

6/3/2022 • 2 minutes to read • [Edit Online](#)

Using ADSI, you read a control access right ACE just as you would any other ACE in an ACL. Be aware that you can also use the Win32 security APIs to read ACLs on directory objects. However, control access rights use the properties of the **IADsAccessControlEntry** interface in a manner that is specific to granting and denying control access rights:

- **AccessMask** must contain **ADS_RIGHT_DS_CONTROL_ACCESS**.
- **Flags** value is **ADS_FLAG_OBJECT_TYPE_PRESENT**.
- **ObjectType** is the string form of the **rightsGUID** attribute of the control access right. The string format of the GUID is the same string format as the **StringFromGUID2** COM Library function.
- **AceType** is either **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT** to grant the trustee the control access right or **ADS_ACETYPE_ACCESS_DENIED_OBJECT** to deny the trustee the control access right.
- **Trustee** is the security principal; that is the user, group, computer, and so on, to which the ACE applies.

Use the following procedure to read an ACE for an ADSI object. The following procedure applies to C and C++ applications.

To read an ACE for an ADSI object

1. Get an **IADs** interface pointer to the object.
2. Use the **IADs::Get** method to get the security descriptor of the object. The name of the property that contains the security descriptor is "nTSecurityDescriptor". The property will be returned as a **VARIANT** that contains an **IDispatch** pointer. Be aware that the **vt** member is **VT_DISPATCH**. Call **QueryInterface** on that **IDispatch** pointer to get an **IADsSecurityDescriptor** interface to use the methods on that interface to access the security descriptor ACL.
3. Use the **IADsSecurityDescriptor::get_DiscretionaryAcl** method to get the ACL. The method returns an **IDispatch** pointer. Call **QueryInterface** on that **IDispatch** pointer to get an **IADsAccessControlList** interface to use the methods on that interface to access the individual ACEs in the ACL.
4. Use the **IADsAccessControlList::get_NewEnum** method to enumerate the ACEs. The method returns an **IUnknown** pointer. Call **QueryInterface** on that **IUnknown** pointer to get an **IEnumVARIANT** interface.
5. Use the **IEnumVARIANT::Next** method to enumerate the ACEs in the ACL. The property is returned as a **VARIANT** that contains an **IDispatch** pointer. Be aware that the **vt** member is **VT_DISPATCH**. Call **QueryInterface** on that **IDispatch** pointer to get an **IADsAccessControlEntry** interface to read the ACE.
6. Call the **IADsAccessControlEntry::get_AccessMask** method to get the **AccessMask** and verify that the **AccessMask** value for the **ADS_RIGHT_DS_CONTROL_ACCESS** flag. If it has this flag, the ACE contains a control access right.
7. Call the **IADsAccessControlEntry::get_Flags** method to get the flag for the object type.
8. Check **Flags** value for **ADS_FLAG_OBJECT_TYPE_PRESENT** flag. If **Flags** is set to **ADS_FLAG_OBJECT_TYPE_PRESENT**, call the **IADsAccessControlEntry::get_ObjectType** method to get a string that contains the **rightsGUID** of the control access right that the ACE applies to.
9. Call the **IADsAccessControlEntry::get_AceType** method to get the ACE type. The type will be an **ADS_ACETYPE_ACCESS_ALLOWED_OBJECT** to grant the trustee the control access right or **ADS_ACETYPE_ACCESS_DENIED_OBJECT** to deny the control access right.
10. Call the **IADsAccessControlEntry::get_Trustee** method to get the security principal; that is user, group, computer, and so on to which the ACE applies.

11. When finished with the **ObjectType** and **Trustee** strings, use **SysFreeString** to free the memory for those strings.
12. When finished with the interfaces, call **Release** to decrement or release all the interface references.

Example Code for Checking for a Control Access Right in an ACE

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ code example verifies a specified control access right in an ACE in the ACL of the specified object.

```
*****  
ReadExtendedRight()  
  
DESCRIPTION: ReadExtendedRight verifies the specified control  
access right for the specified object. If an ACE with that control  
access right exists, it displays (using wprintf) the trustee and  
ACE type for the control access right.  
  
FLOW: Get the security descriptor of an object, get the ACL,  
enumerate the ACEs, check for control access rights ACEs,  
verify the specified right, and display the trustee and ACE  
type.  
  
The pszRightsGUID UNICODE string should be a string that  
contains the rightsGUID property value of the control access  
right and the string should have the same format as the COM  
Library function StringFromGUID2.  
  
For example:  
LPCWSTR pszRightsGUID = L"{8186e976-4d8a-11d2-95dd-0000f875b660}";  
The pfExists parameter specifies a BOOL that will receive  
TRUE if an ACE with the specified right exists; otherwise, FALSE.  
*****  
  
HRESULT ReadExtendedRight(IADs *pObject,  
                         LPCWSTR pszRightsGUID,  
                         BOOL *pfExists)  
  
{  
    if(!pObject || !pszRightsGUID)  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr = E_FAIL;  
    BOOL fExists = FALSE;  
    CComVariant svar;  
  
    // Get the nTSecurityDescriptor.  
    hr = pObject->Get(CComBSTR("nTSecurityDescriptor"), &svar);  
    if(SUCCEEDED(hr) && (VT_DISPATCH == svar.vt))  
    {  
        CComPtr<IADsSecurityDescriptor> spSD;  
  
        // Call the QueryInterface method for the  
        // IADsSecurityDescriptor pointer.  
        hr = svar.pdispVal->QueryInterface(IID_IADsSecurityDescriptor,  
                                         (void**)&spSD);  
        if (SUCCEEDED(hr))  
        {  
            CComPtr<IDispatch> spDisp;
```



```

        wprintf(L"\nObjectType: %S\n",
                sbstrObjectType);

        spACE->get_AceType(&lAceType);
        if (lAceType ==
            ADS_ACETYPE_ACCESS_ALLOWED_OBJECT)
        {
            wprintf(L"ACE Type: ADS_ACETYPE_ACCESS_ALLOWED_OBJECT\n");
        }
        else if (lAceType == ADS_ACETYPE_ACCESS_DENIED_OBJECT)
        {
            wprintf(L"ACE Type: ADS_ACETYPE_ACCESS_DENIED_OBJECT\n");
        }

        // Get the trustee, who the
        // right applies to, and print it.
        spACE->get_Trustee(&sbstrTrustee);
        wprintf(L"Trustee: %S\n", sbstrTrustee);
    }
}
}

// Get the next ACE.
hr = spEnum->Next(1, &svarACE, &lFetch);
}// End of While loop.
}
}
}
}

*pfExists = fExists;

return hr;
}

```

Using DsAddSidHistory

6/3/2022 • 14 minutes to read • [Edit Online](#)

The **DsAddSidHistory** function gets the primary account security identifier (SID) of a security principal from one domain (the source domain) and adds it to the **sIDHistory** attribute of a security principal in another (destination) domain in a different forest. When the source domain is in Windows 2000 native mode, this function also retrieves the **sIDHistory** values of the source principal and adds them to the destination principal's **sIDHistory**.

Adding SIDs to a security principal's **sIDHistory** is a security-sensitive operation that effectively grants to the destination principal access to all resources accessible to the source principal, provided that trusts exist from applicable resource domains to the destination domain.

In a native mode Windows 2000 domain a user logon creates an access token that contains the user primary account SID and group SIDs, as well as the user **sIDHistory** and the **sIDHistory** of the groups of which the user is a member. Having these former SIDs (**sIDHistory** values) in the user's token grants the user access to resources protected by access-control lists (ACLs) containing the former SIDs.

This operation facilitates certain Windows 2000 deployment scenarios. In particular, it supports a scenario in which accounts in a new Windows 2000 forest are created for users and groups that already exist in an Windows NT 4.0 production environment. By placing the Windows NT 4.0 account SID in the Windows 2000 account **sIDHistory**, access to network resources is preserved for the user logging onto his new Windows 2000 account.

DsAddSidHistory also supports migration of Windows NT 4.0 backup domain controllers (BDCs) resource servers (or DCs and member servers in a native mode Windows 2000 domain) to a Windows 2000 domain as member servers. This migration requires the creation, in the destination Windows 2000 domain, of domain local groups that contain, in their **sIDHistory**, the primary SIDs of the local groups defined on the BDC (or domain local groups referenced in ACLs on the Windows 2000 servers) in the source domain. By creating a destination local group containing the **sIDHistory** and all members of the source local group, access to the migrated server resources, protected by ACLs referencing the source local group, is maintained for all members.

NOTE

Use of **DsAddSidHistory** requires an understanding of its broader administrative and security implications in these and other scenarios. For more information, see the white paper "Planning Migration from Windows NT to Microsoft Windows 2000", delivered as Dommig.doc in the Windows 2000 Support Tools. This documentation may also be found on the product CD under \support\tools.

Authorization Requirements

DsAddSidHistory requires administrator privileges in the source and destination domains. Specifically, the caller of this API must be a member of the Domain Administrators group in the destination domain. A hard-coded check for this membership is performed. Also, the caller or the account provided in the *SrcDomainCreds* parameter, if not **NULL**, must be a member of either the Administrators or Domain Administrators group in the source domain.

Domain and Trust Requirements

DsAddSidHistory requires that the destination domain be in Windows 2000 native mode or later, because

only this domain type supports the **sIDHistory** attribute. The source domain may be either Windows NT 4.0 or Windows 2000, mixed or native mode. The source and destination domains must not be in the same forest. Windows NT 4.0 domains are by definition not in a forest. This inter-forest constraint ensures that duplicate SIDs, whether appearing as primary SIDs or **sIDHistory** values, are not created in the same forest.

DsAddSidHistory requires an external trust from the source domain to the destination domain in the cases listed in the following table.

CASE	DESCRIPTION
The source domain is Windows 2000.	The source sIDHistory attribute, available only in Windows 2000 source domains, may be read only using LDAP, which requires this trust for integrity protection.
The source domain is Windows NT 4.0 and <i>SrcDomainCreds</i> is NULL .	The impersonation, required to support source domain operations using the caller's credentials, depends on this trust. Impersonation also requires that the destination domain controller has "Trusted for Delegation" enabled by default on domain controllers.

However, there is no trust required between the source and destination domains if the source domain is Windows NT 4.0 and *SrcDomainCreds* is not **NULL**.

Source Domain Controller Requirements

DsAddSidHistory requires that the domain controller, selected as the target for operations in the source domain, be the PDC in Windows NT 4.0 domains or the PDC Emulator in Windows 2000 domains. Source domain auditing is generated by way of write operations, therefore, the PDC is required in Windows NT 4.0 source domains, and the PDC-only restriction ensures that **DsAddSidHistory** audits are generated on a single computer. This reduces the need to review the audit logs of all DCs to monitor use of this operation.

NOTE

In Windows NT 4.0 source domains, the PDC (target of operations in the source domain) must be running Service Pack 4 (SP4) and later to ensure proper auditing support.

The following registry value must be created as a REG_DWORD value and set to 1 on the source domain controller for both Windows NT 4.0 and Windows 2000 source DCs.

```
HKEY_LOCAL_MACHINE  
  System  
    CurrentControlSet  
      Control  
        Lsa  
          TcpipClientSupport
```

Setting this value enables RPC calls over the TCP transport. This is required because, by default, SAMRPC interfaces are remotable only on named pipes. Using named pipes results in a credential management system suitable for interactively logged-on users making networked calls, but is not flexible for a system process that makes network calls with user-supplied credentials. RPC over TCP is more suitable for that purpose. Setting this value does not diminish system security. If this value is created or changed, the source domain controller must be restarted for this setting to take effect.

A new local group, "<SrcDomainName>\$\$\$", must be created in the source domain for auditing purposes.

Auditing

DsAddSidHistory operations are audited to ensure that both source and destination domain administrators are able to detect when this function has been run. Auditing is mandatory in both the source and destination domains. **DsAddSidHistory** verifies that the Audit Mode is on in each domain and that Account Management auditing of Success/Failure events is on. In the destination domain, a unique "Add Sid History" audit event is generated for each successful or failed **DsAddSidHistory** operation.

Unique "Add Sid History" audit events are not available on Windows NT 4.0 systems. To generate audit events that unambiguously reflect use of **DsAddSidHistory** against the source domain, it performs operations on a special group whose name is the unique identifier in the audit log. A local group, "<SrcDomainName>\$\$\$", whose name is composed of the source domain NetBIOS name appended with three dollar signs (\$) (ASCII code = 0x24 and Unicode = U+0024), must be created on the source domain controller prior to calling **DsAddSidHistory**. Each source user and global group that is a target of this operation is added to and then removed from the membership of this group. This generates Add Member and Delete Member audit events in the source domain, which can be monitored by searching for events that reference the group name.

NOTE

DsAddSidHistory operations on local groups in a Windows NT 4.0, or Windows 2000 mixed-mode source domain cannot be audited because local groups cannot be made members of another local group and therefore cannot be added to the special "<SrcDomainName>\$\$\$" local group. This lack of auditing does not present a security issue to the source domain, because source domain resource access is not affected by this operation. Adding the SID of a source local group to a destination local group does not grant access to source resources, protected by that local group, to any additional users. Adding members to the destination local group does not grant them access to source resources. Added members are granted access only to servers in the destination domain migrated from the source domain, which may have resources protected by the source local group SID.

Data Transmission Security

DsAddSidHistory enforces the following security measures:

- Called from a Windows 2000 workstation, the caller's credentials are used to authenticate and privacy-protect the RPC call to the destination domain controller. If *SrcDomainCreds* is not **NULL**, both the workstation and the destination DC must support 128-bit encryption to privacy-protect the credentials. If 128-bit encryption is not available and *SrcDomainCreds* are provided, then the call must be made on the destination DC.
- The destination domain controller communicates with the source domain controller using either *SrcDomainCreds* or the caller's credentials to mutually authenticate and integrity-protect the read of the source account SID (using a SAM lookup) and **sIDHistory** (using an LDAP read).

Threat Models

The following table lists the potential threats associated with the **DsAddSidHistory** call and addresses the security measures pertinent to the particular threat.

POTENTIAL THREAT	SECURITY MEASURE
------------------	------------------

POTENTIAL THREAT	SECURITY MEASURE
<p>Man in the Middle Attack</p> <p>An unauthorized user intercepts the <i>lookup SID of source object</i> return call, replacing the source object SID with an arbitrary SID for insertion into a target object SIDHistory.</p>	<p>The <i>lookup SID of source object</i> is an authenticated RPC, using the caller's administrator credentials, with packet integrity message protection. This ensures that the return call cannot be modified without detection. The destination domain controller creates a unique "Add SID History" audit event that reflects the SID added to the destination account SIDHistory.</p>
<p>Trojan Source Domain</p> <p>An unauthorized user creates a "Trojan Horse" source domain on a private network that has the same domain SID and some of the same account SIDs as the legitimate source domain. The unauthorized user then attempts to run DsAddSidHistory in a destination domain to obtain the SID of a source account. This is done without the need for the real source Domain Administrator credentials and without leaving an audit trail in the real source domain. The unauthorized user's method for creating the Trojan Horse source domain could be one of the following:</p> <ul style="list-style-type: none"> • Obtain a copy (BDC backup) of the source domain SAM. • Create a new domain, altering the domain SID on disk to match the legitimate source domain SID, then create enough users to instantiate an account with the desired SID. • Create a BDC replica. This requires source domain Administrator credentials. Then the unauthorized user takes the replica to a private network to implement the attack. 	<p>Although there are many ways for an unauthorized user to retrieve or create a desired source object SID, the unauthorized user cannot use it to update an account's SIDHistory without being a member of the destination Domain Administrators group. Because the check, on the destination domain controller, for Domain Administrator membership is hard-coded, on the target DC, there is no method for doing a disk modification to change the access control data protecting this function. An attempt to clone a Trojan Horse source account is audited in the destination domain. This attack is mitigated by reserving membership in the Domain Administrators group for only highly trusted individuals.</p>
<p>On-disk Modification of SID History</p> <p>A sophisticated unauthorized user, with Domain Administrator credentials and with physical access to a DC in the destination domain, could modify an account SIDHistory value on disk.</p>	<p>This attempt is not enabled by use of DsAddSidHistory. This attack is mitigated by preventing physical access to domain controllers to all except highly trusted administrators.</p>
<p>Rogue Code Used to Remove Protections</p> <p>An unauthorized user or rogue administrator with physical access to the Directory Service code could create rogue code that:</p> <ol style="list-style-type: none"> 1. Removes the check for membership in the Domain Administrators group in the code. 2. Changes the calls on the source domain controller that points the SID to a <i>LookupSidFromName</i> that is not audited. 3. Removes audit log calls. 	<p>Someone with physical access to the DS code and knowledgeable enough to create rogue code has the capability of arbitrarily modifying the SIDHistory attribute of an account. The DsAddSidHistory API does not increase this security risk.</p>
<p>Resources Vulnerable to Stolen SIDs</p> <p>If an unauthorized user has succeeded in using one of the methods described here to modify an account SIDHistory, and if the resource domains of interest trust the unauthorized user account domain, then the unauthorized user can get unauthorized access to the stolen SID's resources, potentially without leaving an audit trail in the account domain from which the SID was stolen.</p>	<p>Resource domain administrators protect their resources by setting up only those trust relationships that make sense from a security perspective. Use of DsAddSidHistory is restricted, in the trusted target domain, to members of the Domain Administrators group who already have broad permissions within the scope of their responsibilities.</p>

POTENTIAL THREAT	SECURITY MEASURE
Rogue Target Domain An unauthorized user creates a Windows 2000 domain with an account whose sIDHistory contains a SID that has been stolen from a source domain. The unauthorized user uses this account for unauthorized access to resources.	The unauthorized user requires Administrator credentials for the source domain in order to use DsAddSidHistory , and leaves an audit trail on the source domain controller. The rogue target domain gains unauthorized access only in other domains that trust the rogue domain, which requires Administrator privileges in those resource domains.

Operational Constraints

This section describes the operational constraints of using the [DsAddSidHistory](#) function.

The SID of *SrcPrincipal* must not already exist in the destination forest, either as a primary account SID or in the **sIDHistory** of an account. The exception is that [DsAddSidHistory](#) does not generate an error when attempting to add a SID to a **sIDHistory** that contains an identical SID. This behavior enables [DsAddSidHistory](#) to be run multiple times with identical input, resulting in success and a consistent end state, for tool developer ease-of-use.

NOTE

Global Catalog replication latency may provide a window during which duplicate SIDs may be created. However, duplicate SIDs can be easily deleted by an administrator.

SrcPrincipal and *DstPrincipal* must be one of the following types:

- User
- Security Enabled Group, including:

Local group Global group Domain local group (Windows 2000 native mode only) Universal group (Windows 2000 native mode only)

The object types of *SrcPrincipal* and *DstPrincipal* must match.

- If *SrcPrincipal* is a User, *DstPrincipal* must be a User.
- If *SrcPrincipal* is a Local or Domain Local Group, *DstPrincipal* must be a Domain Local Group.
- If *SrcPrincipal* is a Global or Universal Group, *DstPrincipal* must be a Global or Universal Group.

SrcPrincipal and *DstPrincipal* cannot be one of the following types: ([DsAddSidHistory](#) fails with an error in these cases)

- Computer (workstation or domain controller)
- Inter-domain trust
- Temporary duplicate account (virtually unused feature, a legacy of LANman)
- Accounts with Well Known SIDs. Well Known SIDs are identical in every domain; thus adding them to a **sIDHistory** would violate the SID uniqueness requirement of a Windows 2000 forest. Accounts with Well Known SIDs include the following local groups:

Account operators Administrators Backup operators Guests Print operators Replicator Server operators
Users

If *SrcPrincipal* has a well-known relative identifier (RID) and a domain specific prefix, that is, Domain Administrators, Domain Users, and Domain Computers, then *DstPrincipal* must possess the same well-known

RID in order for [DsAddSidHistory](#) to succeed. Accounts with well-known RIDs include the following users and global groups:

- Administrator
- Guest
- Domain administrators
- Domain guests
- Domain users

Setting the Registry Value

The following procedure shows how to set the TcpipClientSupport registry value.

To Set the TcpipClientSupport Registry Value

1. Create the following registry value as a REG_DWORD value on the source domain controller and set its value to 1.

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\TcpipClientSupport`

2. Then, restart the source domain controller. This registry value causes the SAM to listen on TCP/IP. [DsAddSidHistory](#) will fail if this value is not set on the source domain controller.

Enabling Auditing of User/Group Management Events

The following procedure shows how to enable auditing of User/Group management events in a Windows 2000 or Windows Server 2003 source or destination domain.

To enable auditing of User/Group management events in a Windows 2000 or Windows Server 2003 source or destination domain

1. In the Active Directory Users and Computers MMC Snap-in, select the destination domain Domain Controllers container.
2. Right-click **Domain Controllers** and choose **Properties**.
3. Click the **Group Policy** tab.
4. Select the **Default Domain Controllers Policy** and click **Edit**.
5. Under **Computer Configuration\Windows Settings\Security Settings\Local Policies\Audit Policy**, double-click **Audit Account Management**.
6. In the **Audit Account Management** window, select both **Success** and **Failure** auditing. Policy updates take effect after a restart or after a refresh occurs.
7. Verify that auditing is enabled by viewing the effective audit policy in the Group Policy MMC Snap-in.

The following procedure shows how to enable auditing of User/Group management events in a Windows NT 4.0 domain.

To enable auditing of User/Group management events in a Windows NT 4.0 domain

1. In **User Manager for Domains**, click the **Policies** menu and select **Audit**.
2. Select **Audit These Events**.
3. For **User and Group Management**, check **Success and Failure**.

The following procedure shows how to enable auditing of User/Group management events in a Windows NT 4.0, Windows 2000, or Windows Server 2003 source domain.

To enable auditing of User/Group management events in a Windows NT 4.0, Windows 2000, or Windows Server 2003 source domain

1. In User Manager for Domains, click the User menu and select New Local Group.
2. Enter a group name composed of the source domain NetBIOS name appended with three dollar signs (\$), for example, FABRIKAM\$\$. The description field should indicate that this group is used to audit use of [DsAddSidHistory](#) or cloning operations. Ensure there are no members in the group. Click OK.

The [DsAddSidHistory](#) operation fails if source and destination auditing are not enabled as described here.

Set up Trust if Required

If one of the following is true, a trust must be established from the source domain to the destination domain (this must occur in a different forest):

- The source domain is Windows Server 2003.
- The source domain is Windows NT 4.0 and *SrcDomainCreds* is NULL.

Controlling Object Visibility

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services provide the ability to hide objects from users who have been denied certain rights. If an object is hidden, an application that is running with a user's credentials will not be able to enumerate or bind to the object.

If a user is granted the **ADS_RIGHT_ACTRL_DS_LIST** access control right on a container, the user can view any of the child objects of the container. Likewise, if a user is denied the **ADS_RIGHT_ACTRL_DS_LIST** access control right on a container, the user cannot view any of the child objects of the container. This allows the contents of entire containers to be hidden.

The Active Directory server can also be put into a special list object mode by setting the third character of the **dSHeuristics** property to "1". The list object mode can be disabled by setting the third character of the **dSHeuristics** property to "0". When the Active Directory server is in the list object mode, an object will still be visible if the user has been granted the **ADS_RIGHT_ACTRL_DS_LIST** right on the parent object. If, however, the user has been denied the **ADS_RIGHT_ACTRL_DS_LIST** right on the parent, specific child objects can still be made visible if the user is granted the **ADS_RIGHT_DS_LIST_OBJECT** right on both the parent and child objects. The list object mode allows the system administrator to grant or deny access to individual objects for users or groups. The list object mode should be used sparingly because it requires a significantly higher number of access check calls to be made by the directory service to determine if an object is visible to a user. Thus it can have a negative effect on the performance of browsing or reading objects from Active Directory Domain Services.

Null DACLs and Empty DACLs (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

The presence of a null discretionary access-control list (DACL) in the **nTSecurityDescriptor** attribute of any object can create a serious security risk. A null DACL grants full access to any user that requests it; normal security checking is not performed with respect to the object. A null DACL should not be confused with an empty DACL. An empty DACL is a properly allocated and initialized DACL containing no access-control entries (ACEs). An empty DACL grants no access to the object it is assigned to.

For more information, see [Null DACLs and Empty DACLs](#).

Extending the Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory directory service schema defines the attributes and classes used in Active Directory Domain Services. The base schema that is included in the system contains a rich set of class definitions, such as **user**, **computer**, and **organizationalUnit**, and attribute definitions, such as **userPrincipalName**, **telephoneNumber**, and **objectSid**. The existing set of classes and attributes will be sufficient for most applications. However, the schema is extensible, which means that you can define new classes and attributes. This section discusses how to extend the Active Directory schema.

When to Extend the Schema

If the existing classes and attributes do not fit with the type of data you want to store, you should extend the schema. It is important to note that schema additions are permanent; you can disable classes and attributes, but you can never remove them from the schema. Keep this in mind when you are testing code.

Also consider the size of the data that you want to store. Microsoft recommends that no attribute value exceed 500 kilobytes, including the sum of multivalued attributes. Also, objects should not exceed 1 megabyte in size. Consider also the number of instances of the data; if you add a new attribute to the **User** class on a system that has 100,000 users, this can use up considerable space.

Topics in this section include:

- How to bind to the schema container and read the properties of existing classes and attributes.
- How and when to extend the schema by defining new attributes and classes.
- How to install schema extensions using LDIFDE, CSVDE, or programmatically with ADSI.

For more information and an overview of the Active Directory schema, including information about the schema implementation, class definitions, and attribute definitions, see [Active Directory Schema](#).

For more information, including reference pages for the predefined schema classes, attributes, and attribute syntaxes, see the Active Directory Schema Reference in the [Active Directory Domain Services Reference](#).

Guidelines for Binding to the Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

There are two ways to bind to the Active Directory schema:

- Bind directly to the schema container or to a **classSchema** or **attributeSchema** object in the schema container. The **classSchema** or **attributeSchema** objects contain complete, formal definitions of every class and attribute that can exist in an Active Directory Domain forest. For more information, see [Reading attributeSchema and classSchema Objects](#).
- Bind to the abstract schema or to a class or attribute entry in the abstract schema. The abstract schema contains only a subset of the data about each class and attribute, but the data is in a format that is easy to retrieve and use. For more information, see [The Abstract Schema](#) and [Reading the Abstract Schema](#).

To modify or extend the schema, bind directly to the schema container. To read the class and attribute definitions, it is usually easier to read from the abstract schema.

It is easier to read from the abstract schema for the following reasons:

- ADSI provides special binding techniques and a set of interfaces to read the abstract schema.
- The ADSI interfaces that work with the abstract schema return data in a format appropriate for use in other ADSI interfaces. For example, **IADsClass** and **IADsProperty** typically use **IDAPDisplayName** strings to report attribute and class names, even though this data is stored in the directory in the form of object identifiers (OIDs). The **IDAPDisplayName** format is convenient because other ADSI interfaces use it to refer to classes and attributes in search filters and elsewhere.
- The abstract schema entry for an object class contains data collected from multiple **classSchema** objects. For example, the possible parents, mandatory attributes, and optional attributes for an object class are the union of these attributes from the class's superclasses and auxiliary classes. If you read from the actual schema container, you need to collect data from the various **classSchema** objects that the class was derived from. If you read from the abstract schema, the data is in one place.

You should bind directly to the schema container rather than using the abstract schema in the following cases:

- To get specific attributes not exposed in the abstract schema. For example, **oMSyntax**, **attributeSchema**, **defaultSecurityDescriptor**, and other attributes are not exposed in the abstract schema.
- To query for **attributeSchema** and **classSchema** objects. To search for classes or attributes that match a specified filter, bind to the schema container and perform a one-level search.
- To add or modify attributes or classes. The abstract schema is read-only; you cannot use it to modify or extend the schema. Be aware that modifications must be made at the domain controller that is the schema master. For more information, see [Prerequisites for Installing a Schema Extension](#).

Reading the Abstract Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic provides a code example and guidelines for reading from the abstract schema, which provides a subset of the data stored in the **attributeSchema** and **classSchema** objects in the schema container. To retrieve data that is unavailable in the abstract schema, read the data directly from the schema container as described in [Reading attributeSchema and classSchema Objects](#).

Use the "LDAP://schema" binding string to bind to an **IADsContainer** pointer on the abstract schema. Use this pointer to enumerate the class, attribute, and syntax entries in the abstract schema. You can also use the **IADsContainer.GetObject** method to retrieve individual entries.

```
// Bind to the abstract schema.  
IADsContainer *pAbsSchema = NULL;  
hr = ADsGetObject(L"LDAP://schema",  
                  IID_IADsContainer,  
                  (void**)&pAbsSchema);
```

```
' Bind to the abstract schema.  
Dim adschema As IADsContainer  
Set adschema = GetObject("LDAP://schema")
```

Use a similar binding string, "LDAP://schema/<object>", to bind directly to a class or attribute entry in the abstract schema. In this string, "<object>" is the **IDAPDisplayName** of a class or attribute. For classes bind to the **IADsClass** interface; for attributes, bind to **IADsProperty** interface.

```
// Bind to the user class entry in the abstract schema.  
IADsClass *pClass;  
hr = ADsGetObject(L"LDAP://schema/user",  
                  IID_IADsClass,  
                  (void**)&pClass);
```

```
Bind to the user class entry in the abstract schema.  
Dim userclass As IADsClass  
Set userclass = GetObject("LDAP://schema/user")
```

In addition, the **IADs** interface provides the **IADs.Schema** property. This property returns the ADsPath for the object class in abstract schema binding string format. If you have an **IADs** pointer to an object, you can bind to its class in the abstract schema using the ADsPath returned from **IADs.Schema**.

For classes, the following table lists key properties provided by the abstract schema.

PROPERTY	MEANING
IADsClass.Abstract	Indicates whether this is an abstract class.
IADsClass.Auxiliary	Indicates whether this is an auxiliary class.
IADsClass.AuxDerivedFrom	Array of auxiliary classes this class derives from.

PROPERTY	MEANING
IADsClass.Container	Indicates whether objects of this class can contain other objects, which is true if any class includes this class in its list of possibleSuperiors .
IADsClass.DerivedFrom	Array of classes that this class is derived from.
IADsClass.MandatoryProperties	Retrieves an array of the mandatory properties that must be set for an instance of the class. The returned list includes all the mustContain and systemMustContain values for the class and the classes from which it is derived, including superclasses and auxiliary classes.
IADsClass.OID	Retrieves the governsID of the class.
IADsClass.OptionalProperties	Retrieves an array of the optional properties that might be set for an instance of the class. The returned list includes all the mayContain and systemMayContain values for the class and the classes from which it is derived, including superclasses and auxiliary classes.
IADsClass.PossibleSuperiors	Retrieves an array of the possibleSuperiors values for the class, which indicates the object classes that can contain objects of this class.

The abstract schema is stored in the **subSchema** object in the schema container. To get the distinguished name of the **subSchema** object, bind to rootDSE and read the **subSchemaSubEntry** attribute, as described in [Serverless Binding and RootDSE](#). Be aware that it is more efficient to read the abstract schema by binding to "LDAP://schema", than by binding directly to the **subSchema** object.

Example Code for Enumerating Schema Classes, Attributes, and Syntaxes

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ example shows how to enumerate the classes, attributes, and syntaxes in the abstract schema.

```
// Add adsiid.lib to project
// Add activeds.lib to project

#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <activeds.h>

// Entry point to the application
int wmain(int argc, WCHAR* argv[])
{
    HRESULT hr;
    IADsContainer *pAbsSchema = NULL; // For the abstract schema
    IADsClass *pClass = NULL; // For class objects
    IADsProperty *pProp = NULL; // For attribute objects
    IADsSyntax *pSyntax = NULL; // For syntax objects
    IEnumVARIANT *pEnum = NULL;
    ULONG lFetch;
    VARIANT var;
    VARIANT_BOOL bMulti, bAbstract, bAux;
    BSTR bstrPI, bstrClass, bstrName;
    LONG lCount = 0;
    LONG lVarType = 0;
    IADs *pChild = NULL;
    DWORD dwUnknownClass = 0;

    CoInitialize(NULL);

    // Bind to the abstract schema.
    hr = ADsGetObject(L"LDAP://schema",
                      IID_IADsContainer,
                      (void**)&pAbsSchema);
    if (FAILED(hr))
        goto cleanup;

    // Enumerate the attribute and class entries in the abstract schema.
    hr = ADsBuildEnumerator( pAbsSchema, &pEnum );
    if (FAILED(hr))
        goto cleanup;

    VariantInit(&var);
    hr = ADsEnumerateNext( pEnum, 1, &var, &lFetch );
    while( hr == S_OK && lFetch == 1)
    {
        // Identify whether this is a class, attribute, or syntax.
        hr = V_DISPATCH(&var)->QueryInterface( IID_IADs,
                                                (void**) &pChild );
        if ( FAILED(hr) )
            goto cleanuploop;
        hr = pChild->get_Class(&bstrClass);
        if ( FAILED(hr) )
            goto cleanuploop;
        wprintf(L"%s", bstrClass );
        hr = pChild->put_Name(bstrName);
    }
}
```

```

    hr = pChild->get_Name(&wsName);
    if ( FAILED(hr) )
        goto cleanuploop;
    wprintf(L"%s", bstrName );

    // Retrieve data, depending on the type of schema element.
    if (_wcsicmp(L"Class", bstrClass)==0)
    {
        hr = pChild->QueryInterface( IID_IADsClass,
            (void**) &pClass );
        if ( FAILED(hr) )
            goto cleanuploop;
        pClass->get_Abstract(&bAbstract);
        pClass->get_Auxiliary(&bAux);
        if (bAbstract)
            wprintf(L"Abstract");
        else if (bAux)
            wprintf(L"Auxiliary");
        else
            wprintf(L"Structural");

        // Retrieve the primary ADSI
        // interface to use with this class.
        pClass->get_PrimaryInterface(&bstrPI);
        if (_wcsicmp(L"{FD8256D0-FD15-11CE-ABC4-02608C9E7553}",
            bstrPI)==0)
            wprintf(L"%s", IID_IADS,%s", bstrPI);
        if (_wcsicmp(L"{B15160D0-1226-11CF-A985-00AA006BC149}",
            bstrPI)==0)
            wprintf(L"%s", IID_IADsPrintQueue,%s", bstrPI);
        if (_wcsicmp(L"{A2F733B8-EFFE-11CF-8ABC-00C04FD8D503}",
            bstrPI)==0)
            wprintf(L"%s", IID_IADsOU,%s", bstrPI);
        if (_wcsicmp(L"{A05E03A2-EFFE-11CF-8ABC-00C04FD8D503}",
            bstrPI)==0)
            wprintf(L"%s", IID_IADsLocality,%s", bstrPI);
        if (_wcsicmp(L"{3E37E320-17E2-11CF-ABC4-02608C9E7553}",
            bstrPI)==0)
            wprintf(L"%s", IID_IADsUser,%s", bstrPI);
        if (_wcsicmp(L"{27636B00-410F-11CF-B1FF-02608C9E7553}",
            bstrPI)==0)
            wprintf(L"%s", IID_IADsGroup,%s", bstrPI);
        if (_wcsicmp(L"{00E4C220-FD16-11CE-ABC4-02608C9E7553}",
            bstrPI)==0)
            wprintf(L"%s", IID_IADsDomain,%s", bstrPI);
        SysFreeString(bstrPI);
    }
    else if (_wcsicmp(L"Property", bstrClass)==0 )
    {
        hr = pChild->QueryInterface( IID_IADsProperty,
            (void**) &pProp );
        if ( FAILED(hr) )
            goto cleanuploop;
        pProp->get_MultiValued(&bMulti);
        wprintf(L"%s", bMulti ? L"Multi-Valued" : L"Single-Valued");
    }
    else if (_wcsicmp(L"Syntax", bstrClass)==0 )
    {
        hr = pChild->QueryInterface( IID_IADsSyntax,
            (void**) &pSyntax );
        if ( FAILED(hr) )
            goto cleanuploop;
        pSyntax->get_OleAutoDataType ( &lVarType );
        wprintf(L"%u", lVarType);
    }
    else
        dwUnknownClass++;

cleanuploop:
    ...
}

```

```

wprintf(L"\n");
SysFreeString(bstrClass);
SysFreeString(bstrName);
pChild->Release();
VariantClear(&var);
if (SUCCEEDED(hr))
    hr = ADsEnumerateNext( pEnum, 1, &var, &lFetch );
}

wprintf(L"dwUnknownClass: %u\n", dwUnknownClass);

cleanup:
if (pAbsSchema)
    pAbsSchema->Release();
if (pEnum)
    pEnum->Release();
VariantClear(&var);
CoUninitialize();

return hr;
}

```

The following Visual Basic code example reads the abstract schema to enumerate the structural, abstract, and auxiliary object classes. The example then enumerates the single-valued and multi-valued attribute classes.

```

Dim ADSchema As IADsContainer
Dim userclass As IADsClass
Dim aClass As IADsClass
Dim aProp As IADsProperty

' Bind to the abstract schema
Set ADSchema = GetObject("LDAP://schema")

' Enumerate the object classes.
ADSchema.Filter = Array("class")
For Each aClass In ADSchema
If aClass.Abstract Then
AbstractList.AddItem aClass.Name
ElseIf aClass.Auxiliary Then
AuxiliaryList.AddItem aClass.Name
Else
StructuralList.AddItem aClass.Name
End If
Next aClass

' Enumerate the attribute classes.
ADSchema.Filter = Array("property")
For Each aProp In ADSchema
If aProp.MultiValued Then
MultiValProps.AddItem aProp.Name
Else
SingleValProps.AddItem aProp.Name
End If
Next aProp

```

Reading attributeSchema and classSchema Objects

6/3/2022 • 5 minutes to read • [Edit Online](#)

This topic provides code examples and guidelines for reading directly from **attributeSchema** and **classSchema** objects in the schema container. Be aware that, in most programming situations, when you must read data about a class or attribute definition, it is more effective to read data from the abstract schema as described in [Reading the Abstract Schema](#).

The interfaces and techniques used to read from the schema container are those used to read any object in Active Directory Domain Services. The guidelines include:

- To bind to the schema container, obtain its distinguished name which can be retrieved by binding to rootDSE and reading the **schemaNamingContext** property, as described in [Serverless Binding and RootDSE](#).
- Use an **IADsContainer** pointer for the schema container to enumerate the **attributeSchema** and **classSchema** objects.
- Use the **IADs** or **IDirectoryObject** interface to retrieve the properties of an **attributeSchema** and **classSchema** object.
- Use the **ADsOpenObject** or **ADsGetObject** functions to bind directly to an **attributeSchema** or **classSchema** object.
- If you are reading multiple **attributeSchema** or **classSchema** objects, you can increase performance by binding to an **IADsContainer** pointer on the schema container and using the **IADsContainer.GetObject** method to bind to the individual class and attribute objects. This is more efficient than making repeated **ADsOpenObject** or **ADsGetObject** calls to bind to the individual class and attribute objects. Use the **cn** attribute to build the relative path for the **IADsContainer.GetObject** call (for example, "cn=user" for the **classSchema** object for the user class).
- Use an **IDirectorySearch** pointer for the schema container to query the schema for attributes or classes that match a search filter.

For a code example that demonstrates different methods of searching for schema objects, see [Example Code for Searching for Schema Objects](#).

The following C++ code example binds to an **IADsContainer** pointer on the schema container and then uses the **ADsBuildEnumerator** and **ADsEnumerateNext** functions to enumerate its contents. Be aware that the enumeration includes all **attributeSchema** and **classSchema** objects as well as a single **subSchema** object, which is the abstract schema.

For each enumerated object, the code example uses the **IADs.Class** property to determine whether it is an **attributeSchema** or **classSchema** object. The code example shows how to read the properties that are unavailable from the abstract schema.

```
// Add activeds.lib to the project.  
// Add adsiid.lib to the project.  
  
#include "stdafx.h"  
#include <windows.h>  
#include <stdio.h>  
#include <ole2.h>  
#include <activeds.h>  
#include <atldbase.h>  
  
// Forward declarations.  
void ProcessAttribute(IADs *pChild);  
void ProcessClass(IADs *pChild);
```

```

// Entry point for the application.
int wmain(int argc, WCHAR* argv[])
{
    HRESULT hr;

    hr = CoInitialize(NULL);
    if(SUCCEEDED(hr))
    {
        CComBSTR sbstrDSPPath;
        CComVariant svar;
        IADs *pRootDSE = NULL;
        IADsContainer *pSchema = NULL;
        IEnumVARIANT *pEnum = NULL;
        ULONG lFetch;
        CComBSTR sbstrClass;
        DWORD dwClasses = 0, dwAttributes = 0, dwUnknownClass = 0;

        // Bind to rootDSE to get the schemaNamingContext property.
        hr = ADsGetObject(L"LDAP://rootDSE", IID_IADs, (void**)&pRootDSE);
        if (FAILED(hr))
        {
            wprintf(L"ADsGetObject failed: 0x%x\n", hr);
            goto cleanup;
        }

        hr = pRootDSE->Get(CComBSTR("schemaNamingContext"), &svar);
        sbstrDSPPath = "LDAP://";
        sbstrDSPPath += svar.bstrVal;

        // Bind to the actual schema container.
        wprintf(L"Binding to %s\n", sbstrDSPPath);
        hr = ADsGetObject(sbstrDSPPath, IID_IADsContainer, (void**) &pSchema);
        if (FAILED(hr))
        {
            wprintf(L"ADsGetObject to schema failed: 0x%x\n", hr);
            goto cleanup;
        }

        // Enumerate the attribute and class objects in the schema container.
        hr = ADsBuildEnumerator(pSchema, &pEnum);
        if (FAILED(hr))
        {
            wprintf(L"ADsBuildEnumerator failed: 0x%x\n", hr);
            goto cleanup;
        }

        hr = ADsEnumerateNext(pEnum, 1, &svar, &lFetch);
        while(S_OK == hr && 1 == lFetch)
        {
            IADs *pChild = NULL;

            // Get an IADs pointer on the child object.
            hr = V_DISPATCH(&svar)->QueryInterface(IID_IADs, (void**) &pChild);
            if (SUCCEEDED(hr))
            {
                // Verify that this is a class, attribute, or subSchema object.
                hr = pChild->get_Class(&sbstrClass);
                if (SUCCEEDED(hr))
                {
                    // Get data. This depends on type of schema element.
                    if (_wcsicmp(L"classSchema", sbstrClass) == 0)
                    {
                        dwClasses++;
                        wprintf(L"%s\n", sbstrClass);
                        ProcessClass(pChild);
                        wprintf(L"\n");
                    }
                    else if (_wcsicmp(L"attributeSchema", sbstrClass) == 0)
                }
            }
        }
    }
}

```

```

        {
            dwAttributes++;
            wprintf(L"%s\n", sbstrClass);
            ProcessAttribute(pChild);
            wprintf(L"\n");
        }
        else if (_wcsicmp(L"subSchema", sbstrClass) == 0)
        {
            wprintf(L"abstract schema");
            wprintf(L"\n");
        }
        else
        {
            dwUnknownClass++;
        }
    }

    pChild->Release();
}

hr = ADsEnumerateNext(pEnum, 1, &svar, &lFetch);
}

wprintf(L"Classes: %u\nAttributes: %u\nUnknown: %u\n", dwClasses, dwAttributes, dwUnknownClass);

cleanup:
if (pRootDSE)
{
    pRootDSE->Release();
}

if (pEnum)
{
    ADsFreeEnumerator(pEnum);
}

if (pSchema)
{
    pSchema->Release();
}

}

CoUninitialize();

return 0;
}

// PrintGUIDFromVariant
// Prints a GUID in string format.
void PrintGUIDFromVariant(VARIANT varGUID)
{
    HRESULT hr;
    void *pArray;
    WCHAR szGUID[40];
    LPGUID pGUID;

    DWORD dwLen = sizeof(GUID);

    hr = SafeArrayAccessData(V_ARRAY(&varGUID), &pArray);
    if(SUCCEEDED(hr))
    {
        pGUID = (LPGUID)pArray;

        // Convert GUID to string.
        StringFromGUID2(*pGUID, szGUID, 40);

        // Print GUID.
    }
}

```

```

        wprintf(L"%s",szGUID);

        SafeArrayUnaccessData(V_ARRAY(&varGUID));
    }

}

// PrintADSPROPERTYValue
void PrintADSPROPERTYValue(VARIANT var, BSTR bstrPropertyName, HRESULT hr)
{
    if (E_AMS_PROPERTY_NOT_FOUND == hr)
    {
        wprintf(L"-- not set --\n");
    }
    else if (FAILED(hr))
    {
        wprintf(L"get %s failed: 0x%08x\n", bstrPropertyName, hr);
    }
    else
    {
        switch (var.vt)
        {
        case VT_BSTR:
            wprintf(L"%s\n", var.bstrVal);
            break;

        case VT_I4:
            wprintf(L"%u\n", var.iVal);
            break;

        case VT_BOOL:
            wprintf(L"%s\n", var.boolVal ? L"TRUE" : L"FALSE");
            break;

        default:
            wprintf(L"-- ??? --\n");
        }
    }
}

// ProcessAttribute
// Gets information about an attributeClass object.
void ProcessAttribute(IADS *pChild)
{
    HRESULT hr;
    CComVariant svar;
    CComBSTR sbstrPropertyName;

    // Get the attribute Common-Name (cn) property.
    sbstrPropertyName = "cn";
    hr = pChild->Get(sbstrPropertyName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropertyName, hr);

    // Get the attribute LDAPDisplayName.
    sbstrPropertyName = "LDAPDisplayName";
    hr = pChild->Get(sbstrPropertyName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropertyName, hr);

    // Get the class linkID.
    sbstrPropertyName = "linkID";
    hr = pChild->Get(sbstrPropertyName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropertyName, hr);

    // Get the attribute schemaIDGUID.
    sbstrPropertyName = "schemaIDGUID";
    hr = pChild->Get(sbstrPropertyName, &svar);
    if (E_AMS_PROPERTY_NOT_FOUND == hr)
    {
        wprintf(L"-- not set --\n");
    }
}

```

```

        }
    else if(SUCCEEDED(hr))
    {
        PrintGUIDFromVariant(svar);
    }

    // Get the attribute attributeSecurityGUID.
    sbstrPropName = "attributeSecurityGUID";
    hr = pChild->Get(sbstrPropName, &svar);
    if (E_AMS_PROPERTY_NOT_FOUND == hr)
    {
        wprintf(L"-- not set --\n");
    }
    else if (SUCCEEDED(hr))
    {
        PrintGUIDFromVariant(svar);
    }

    // Get the attribute attributeSyntax property.
    sbstrPropName = "attributeSyntax";
    hr = pChild->Get(sbstrPropName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropName, hr);

    // Get the attribute's oMSyntax property.
    sbstrPropName = "oMSyntax";
    hr = pChild->Get(sbstrPropName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropName, hr);
}

// ProcessClass
// Gets information about a schema class.
void ProcessClass(IADS *pChild)
{
    HRESULT hr;
    CComVariant svar;
    CComBSTR sbstrPropName;

    // Get the class cn.
    sbstrPropName = "cn";
    hr = pChild->Get(sbstrPropName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropName, hr);

    // Get the class LDAPDisplayName.
    sbstrPropName = "LDAPDisplayName";
    hr = pChild->Get(sbstrPropName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropName, hr);

    // Get the class schemaIDGUID.
    sbstrPropName = "schemaIDGUID";
    hr = pChild->Get(sbstrPropName, &svar);
    if (FAILED(hr))
    {
        wprintf(L"get schemaIDGUID failed: 0x%08x", hr);
    }
    else
    {
        PrintGUIDFromVariant(svar);
    }

    // Get the class adminDescription property.
    sbstrPropName = "adminDescription";
    hr = pChild->Get(sbstrPropName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropName, hr);

    // Get the class adminDisplayName property.
    sbstrPropName = "adminDisplayName";
    hr = pChild->Get(sbstrPropName, &svar);
    PrintADSPROPERTYValue(svar, sbstrPropName, hr);
}

```

```
PrintADSPROPERTYvalue(svar, sbstrPropertyName, hr);

// Get the class rDNAAttID.
sbstrPropertyName = "rDNAAttID";
hr = pChild->Get(sbstrPropertyName, &svar);
PrintADSPROPERTYvalue(svar, sbstrPropertyName, hr);

// Get the class defaultHidingValue.
sbstrPropertyName = "defaultHidingValue";
hr = pChild->Get(sbstrPropertyName, &svar);
PrintADSPROPERTYvalue(svar, sbstrPropertyName, hr);

// Get the class defaultObjectCategory.
sbstrPropertyName = "defaultObjectCategory";
hr = pChild->Get(sbstrPropertyName, &svar);
PrintADSPROPERTYvalue(svar, sbstrPropertyName, hr);

// Get the class systemOnly value.
sbstrPropertyName = "systemOnly";
hr = pChild->Get(sbstrPropertyName, &svar);
PrintADSPROPERTYvalue(svar, sbstrPropertyName, hr);

// Get the class defaultSecurityDescriptor.
sbstrPropertyName = "defaultSecurityDescriptor";
hr = pChild->Get(sbstrPropertyName, &svar);
PrintADSPROPERTYvalue(svar, sbstrPropertyName, hr);
}
```

Example Code for Searching for Schema Objects

6/3/2022 • 4 minutes to read • [Edit Online](#)

This topic includes a code example used to search for schema objects.

The following C++ code example shows how to search for schema objects that have bits set in the `systemFlags` attribute.

```
#include <stdio.h>
#include <atldbbase.h>
#include <activeds.h>
#include <ntldap.h>

//****************************************************************************

PrintAttributesByType()

Searches for and prints all attributeSchema objects that contain all of
the specified attribute type flags.

Parameters:

pSchemaNC - IDirectorySearch pointer to schema naming context.

dwAttributeType - Bit flags to search for in systemFlags. These are
values such as ADS_SYSTEMFLAG_ATTR_IS_CONSTRUCTED.

bIsExactMatch - TRUE to find attributes that have systemFlags exactly
matching dwAttributeType. FALSE to find attributes that have the
dwAttributeType bit set, and possibly others.

*****/

HRESULT PrintAttributesByType(IDirectorySearch *pSchemaNC,
    DWORD dwAttributeType,
    BOOL bIsExactMatch)
{
    HRESULT hr;

    // Attributes are one-level deep in the Schema container, so only search one level.
    ADS_SEARCHPREF_INFO SearchPrefs[2];
    SearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    SearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
    SearchPrefs[0].vValue.Integer = ADS_SCOPE_ONELEVEL;

    // Use paging if there are more properties than can be returned by one page.
    SearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;
    SearchPrefs[1].vValue.dwType = ADSTYPE_INTEGER;
    SearchPrefs[1].vValue.Integer = 1000;

    // Set the search preferences.
    hr = pSchemaNC->SetSearchPreference(SearchPrefs, sizeof(SearchPrefs)/sizeof(ADS_SEARCHPREF_INFO));
    if(FAILED(hr))
    {
        return hr;
    }

    // Handle that is used for searching.
    ADS_SEARCH_HANDLE hSearch;

    LPWSTR rgpszAttributes[] = {L"cn"};
    DWORD dwAttributes = sizeof(rgpszAttributes)/sizeof(LPWSTR);
```

```

// Create the search filter.
WCHAR wszAttributeType[30]; // Plenty large enough to handle the biggest 32-bit number.
swprintf_s(wszAttributeType, L"%d", dwAttributeType);

CComBSTR sbstrSearchFilter;
if (bIsExactMatch)
{
    sbstrSearchFilter = "(&(objectCategory=attributeSchema)(systemFlags=";
    sbstrSearchFilter += wszAttributeType;
    sbstrSearchFilter += "))";
}
else
{
    sbstrSearchFilter = "(&(objectCategory=attributeSchema)(systemFlags:";
    sbstrSearchFilter += LDAP_MATCHING_RULE_BIT_AND;
    sbstrSearchFilter += ":=";
    sbstrSearchFilter += wszAttributeType;
    sbstrSearchFilter += "))";
}

// Execute the search.
hr = pSchemaNC->ExecuteSearch(sbstrSearchFilter,
                                rgpwszAttributes,
                                dwAttributes,
                                &hSearch);

if(SUCCEEDED(hr))
{
    // Get the first row of results.
    hr = pSchemaNC->GetFirstRow(hSearch);

    while(S_OK == hr)
    {
        ADS_SEARCH_COLUMN col;

        // Print each column.
        for(DWORD i = 0; i < dwAttributes; i++)
        {
            hr = pSchemaNC->GetColumn(hSearch, rgpwszAttributes[i], &col);
            if(SUCCEEDED(hr))
            {
                // Print the data for the column and free the column.
                if (col.dwADsType == ADSTYPE_CASE_IGNORE_STRING)
                {
                    wprintf(L"%s:", rgpwszAttributes[i]);

                    // Print each attribute value.
                    for(DWORD x = 0; x < col.dwNumValues; x++)
                    {
                        wprintf(L"\t%s\n", col.pADsValues[x].CaseIgnoreString);
                    }
                }
                else
                {
                    wprintf(L"<%s property is not a string>", rgpwszAttributes[0]);
                }

                // Free the column.
                pSchemaNC->FreeColumn(&col);
            }
        }

        // Get the next set of results.
        hr = pSchemaNC->GetNextRow(hSearch);
    }

    // Close the search handle to cleanup.
    pSchemaNC->CloseSearchHandle(hSearch);
}

```

```
        }

    return hr;
}
```

The following C and C++ code example shows how to search for schema objects replicated to the global catalog.

```
#include <stdio.h>
#include <atlbase.h>
#include <activeds.h>
#include <ntldap.h>

//****************************************************************************

PrintGCAttributes()

Searches for and prints all attributeSchema objects replicated
to the global catalog.

Parameters:

pSchemaNC - IDirectorySearch pointer to schema naming context.

***** */

HRESULT PrintGCAttributes(IDirectorySearch *pSchemaNC)
{
    HRESULT hr;

    // Attributes are one-level deep in the Schema container, so only search one level.
    ADS_SEARCHPREF_INFO SearchPrefs[2];
    SearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    SearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
    SearchPrefs[0].vValue.Integer = ADS_SCOPE_ONELEVEL;

    // Use paging if there are more properties than can be returned by one page.
    SearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;
    SearchPrefs[1].vValue.dwType = ADSTYPE_INTEGER;
    SearchPrefs[1].vValue.Integer = 1000;

    // Set the search preferences.
    hr = pSchemaNC->SetSearchPreference(SearchPrefs, sizeof(SearchPrefs)/sizeof(ADS_SEARCHPREF_INFO));
    if(FAILED(hr))
    {
        return hr;
    }

    // Handle used for search.
    ADS_SEARCH_HANDLE hSearch;

    LPWSTR rgpwszAttributes[] = {L"cn"};
    DWORD dwAttributes = sizeof(rgpwszAttributes)/sizeof(LPWSTR);

    // Create the search filter.
    LPWSTR pwszSearchFilter = L"(&(objectCategory=attributeSchema)(isMemberOfPartialAttributeSet=TRUE))";

    // Execute the search.
    hr = pSchemaNC->ExecuteSearch(pwszSearchFilter,
                                    rgpwszAttributes,
                                    dwAttributes,
                                    &hSearch);

    if(SUCCEEDED(hr))
    {
        // Get the first row of results.
        hr = pSchemaNC->GetFirstRow(hSearch);
```

```

while(S_OK == hr)
{
    ADS_SEARCH_COLUMN col;

    // Print each column.
    for(DWORD i = 0; i < dwAttributes; i++)
    {
        hr = pSchemaNC->GetColumn(hSearch, rgpszAttributes[i], &col);
        if(SUCCEEDED(hr))
        {
            // Print the data for the column and free the column.
            if (col.dwADsType == ADSTYPE_CASE_IGNORE_STRING)
            {
                wprintf(L"%s:", rgpszAttributes[i]);

                // Print each attribute value.
                for(DWORD x = 0; x < col.dwNumValues; x++)
                {
                    wprintf(L"\t%sn", col.pADsValues[x].CaseIgnoreString);
                }
            }
            else
            {
                wprintf(L"<%s property is not a string>", rgpszAttributes[0]);
            }

            // Free the column.
            pSchemaNC->FreeColumn(&col);
        }
    }

    // Get the next set of results.
    hr = pSchemaNC->GetNextRow(hSearch);
}

// Close the search handle to cleanup.
pSchemaNC->CloseSearchHandle(hSearch);
}

return hr;
}

```

The following C and C++ code example shows how to search for indexed schema objects.

```

#include <stdio.h>
#include <atlbase.h>
#include <activeds.h>
#include <ntldap.h>

*****
PrintIndexedAttributes()

Searches for, and prints, all indexed attributeSchema objects.

Parameters:

pSchemaNC - IDirectorySearch pointer to schema naming context.

*****
HRESULT PrintIndexedAttributes(IDirectorySearch *pSchemaNC)
{
    HRESULT hr;

    // Attributes are one-level deep in the Schema container, so only search one level.
    ADS_SEARCHPREF_INFO SearchPrefs[2];
    SearchPrefs[0].dwSearchFlags = ADS_SEARCHEPINFO_SEARCH_SCOPE;

```

```

SearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
SearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
SearchPrefs[0].vValue.Integer = ADS_SCOPE_ONELEVEL;

// Use paging in the case that there are more properties than can be returned by one page.
SearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;
SearchPrefs[1].vValue.dwType = ADSTYPE_INTEGER;
SearchPrefs[1].vValue.Integer = 1000;

// Set the search preferences.
hr = pSchemaNC->SetSearchPreference(SearchPrefs, sizeof(SearchPrefs)/sizeof(ADS_SEARCHPREF_INFO));
if(FAILED(hr))
{
    return hr;
}

// Handle used for search.
ADS_SEARCH_HANDLE hSearch;

LPWSTR rgpszAttributes[] = {L"cn"};
DWORD dwAttributes = sizeof(rgpszAttributes)/sizeof(LPWSTR);

/*
Create the search filter. Indexed attributes have the least significant bit
of the searchFlags attribute set to 1.
*/
CComBSTR sbstrSearchFilter;
sbstrSearchFilter = "(&(objectCategory=attributeSchema)(searchFlags:";
sbstrSearchFilter += LDAP_MATCHING_RULE_BIT_AND;
sbstrSearchFilter += ":=1))";

// Execute the search.
hr = pSchemaNC->ExecuteSearch(sbstrSearchFilter,
                                rgpszAttributes,
                                dwAttributes,
                                &hSearch);
if(SUCCEEDED(hr))
{
    // Get the first row of results.
    hr = pSchemaNC->GetFirstRow(hSearch);

    while(S_OK == hr)
    {
        ADS_SEARCH_COLUMN col;

        // Print each column.
        for(DWORD i = 0; i < dwAttributes; i++)
        {
            hr = pSchemaNC->GetColumn(hSearch, rgpszAttributes[i], &col);
            if(SUCCEEDED(hr))
            {
                // Print the data for the column and free the column.
                if (col.dwADsType == ADSTYPE_CASE_IGNORE_STRING)
                {
                    wprintf(L"%s:", rgpszAttributes[i]);

                    // Print each attribute value.
                    for(DWORD x = 0; x < col.dwNumValues; x++)
                    {
                        wprintf(L"\t%s\n", col.pADsValues[x].CaseIgnoreString);
                    }
                }
                else
                {
                    wprintf(L"<%s property is not a string>", rgpszAttributes[0]);
                }

                // Free the column.
                pSchemaNC->FreeColumn(&col);
            }
        }
    }
}

```

```
        }

        // Get the next set of results.
        hr = pSchemaNC->GetNextRow(hSearch);
    }

    // Close the search handle to cleanup.
    pSchemaNC->CloseSearchHandle(hSearch);
}

return hr;
}
```

What You Must Know Before Extending the Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

Before you extend the schema, you should take the following items into consideration:

- [Impact of Schema Changes](#)
- [When to Extend the Schema](#)
- [Restrictions on Schema Extension](#)

Impact of Schema Changes

6/3/2022 • 2 minutes to read • [Edit Online](#)

A schema extension impacts a domain forest controlled by Active Directory Domain Services in several ways:

- Schema changes are global. There is a single schema for an entire forest. The schema is globally replicated: a copy of the schema exists on every DC in the forest. When you extend the schema, it is extended for the entire forest.
- Schema object additions cannot be reversed. When a new class or attribute object is added to the schema, it cannot be removed. An existing attribute or class can be disabled, but not removed. For more information, see [Disabling Existing Classes and Attributes](#). Disabling a class or attribute does not affect existing instances of the class or attribute, but prevents the creation of new instances. You cannot disable an attribute if it is included in any class that is not disabled.
- OIDs must be unique. When an attribute or class is added to the schema, no attribute or class with the same OID can be added. This is true even if a class or attribute has been disabled. For this reason it is you must use valid OIDs. Do not synthesize an OID, and do not reuse an existing OID. For more information about obtaining a valid OID, see [Obtaining a Root Object Identifier](#).
- Some changes can be made after creation:
 - For a category 1 or category 2 class, you can add or remove values in the **possSuperiors** attribute. The **possSuperiors** values specify the object classes that can contain the class.
 - For a category 1 or category 2 class, you can add or remove values in the **mayContain** attribute. The **mayContain** values specify the optional attributes, but may be present in an instance of the class.
- The **IDAPDisplayName** for a Category 2 class or attribute can be changed after it has been created. Typically, you should not change the **IDAPDisplayName**. However, there is one legitimate reason for changing the LDAP name and that is if you made a mistake in defining the attribute or class and must create a new one to replace the old one. You do not have to rename the relative distinguished name (RDN) of the attribute or class schema to do this. When the LDAP name is changed this enables you to work around a mistake rather than rebuilding your whole Windows 2000 infrastructure. For more information, see [Disabling Existing Classes and Attributes](#).

The **IDAPDisplayName** for predefined (category 1) classes or attributes cannot be changed. For more information about category 1 and 2 schema objects, see [Restrictions on Schema Extension](#).

When to Extend the Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

Extend the schema only if no existing object class fulfills the requirements of your application. Extending the schema is a complex operation; schema changes are replicated to every domain controller in the enterprise forest. Consider this carefully.

The schema can be extended in three ways:

- Derive a new subclass from an existing class - the subclass has all of the attributes of the superclass and any attributes you specify. Derive from an existing class:
 - When the existing class requires additional attributes, but otherwise is acceptable.
 - When the ability to transform existing objects of the class into a new class is not required. It is not possible to add a subclass to an existing object.
 - To use the existing Directory Manager snap-in to manage the extended attributes of the objects.
- Add attributes to an existing class and/or add parent objects for the class. When adding multiple attributes, perform this operation in a structured manner by defining an auxiliary class and adding that auxiliary class to the existing class.
- Modification of an existing class is required when an application requires the ability to extend existing objects of the class. For example, to add application-specific data to the User object, extend the class User normally, because you must handle existing Users and not just special Users created by your application.
- Create a new class with the required attributes. Create a new class; that is, a class derived from "Top" when no existing class fulfills the operational requirements.

When you subclass an existing class, any user interface items associated to the original class will not be inherited by the subclass. For example, if you subclass a user object, the property pages and menu items associated to user are not inherited. For this reason, it is preferable to extend an existing object or create an auxiliary class rather than create a subclass.

Whether you subclass an existing class or modify an existing class, you will want to extend tools such as the Active Directory Users and Computers snap-in to manage the extended attributes of the objects. For more information, see [Extending the User Interface for Directory Objects](#).

Restrictions on Schema Extension

6/3/2022 • 2 minutes to read • [Edit Online](#)

In order to reduce the possibility of schema changes by one application breaking other applications and to maintain schema consistency, Active Directory Domain Services enforce restrictions on the type of schema changes that an application or user is allowed to make.

The restrictions are imposed only on modification of existing schema objects. The schema is categorized into two categories. The schema objects that ship with Windows 2000 in the base schema belong to Category 1. Any schema objects added later by other applications or users through dynamic schema extension belong to Category 2. The category of a schema object can be determined by the 0x10 bit set in the **systemFlags** attribute on the **classSchema** object. This bit is only set on Category 1 objects, and cannot be altered, nor can it be set on any Category 2 object.

The **systemFlags** attribute is used internally by Active Directory Domain Services to identify special characteristics of "infrastructure" objects in the base schema. In addition to identifying Category 1 objects, **systemFlags** controls whether an object can be moved, deleted, or renamed. These operations are prevented for objects that Windows 2000 depends on to run.

On any schema objects, Category 1 or 2, Active Directory Domain Services impose the following restrictions:

- You cannot add a new **mustContain** to a class (directly or through inheritance by adding an auxiliary class).
- You cannot delete any **mustContain** of the class (directly or through inheritance).

In addition, the following additional restrictions are imposed on Category 1 schema objects:

- You cannot change the following attributes of a Category 1 attribute:
 - **rangeLower** and **rangeUpper** (value range).
 - **attributeSecurityGuid** (determines which property set the attribute belongs in, if any).
- You cannot change the **defaultObjectCategory** of a Category 1 class.
- You cannot change the **objectCategory** of any instance of a Category 1 class.
- You cannot make a Category 1 class or attribute defunct.
- You cannot change the **IDAPDisplayName** of a Category 1 class or attribute.
- You cannot rename a Category 1 class or attribute.

Querying for Category 1 or 2 Schema Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **systemFlags** attribute of **attributeSchema** and **classSchema** objects is an integer bitmask that contains flags that indicate additional system qualities of the attribute or class. The [ADS_SYSTEMFLAG_ENUM](#) enumeration contains values that correspond to the bits you can set in the **systemFlags** attribute. There are additional **systemFlags** bits that you cannot set, such as the 0x10 bit which indicates whether the attribute or class is category 1 or category 2. The 0x10 bit is set for category 1 objects, which are the classes and attributes included in the base schema included with the system. The bit is not set for category 2 attributes and classes, which are extensions to the schema. If no **systemFlags** property exists on an **attributeSchema** or **classSchema** object, it is category 2.

The **LDAP_MATCHING_RULE_BIT_AND** matching rule can be used to search for objects that have the 0x10 flag set in the **systemFlags** attribute. For more information, see [Search Filter Syntax](#).

Querying for Category 1

The following query string searches for category 1 attributes (**attributeSchema** objects with the 0x10 bit set in the **systemFlags** property).

```
(&(objectCategory=attributeSchema)(systemFlags:1.2.840.113556.1.4.803:=16) )
```

Be aware that, in the example above, the LDAP query syntax requires decimal values; therefore, the hex value of the flag must be converted to decimal. In this case, category 1 bit is 0x10 so the filter value must be specified as 16.

Querying for Category 2

The following query string searches for category 2 attributes (**attributeSchema** objects that do not have the 0x10 bit set in the **systemFlags** property).

```
(&(objectCategory=attributeSchema) (!(systemFlags:1.2.840.113556.1.4.803:=16)))
```

Be aware that this query also returns **attributeSchema** objects that do not have a **systemFlags** property, and, therefore, implicitly do not have the specified flag set.

How to Extend the Schema

6/3/2022 • 2 minutes to read • [Edit Online](#)

When the existing classes and/or attributes do not fit with the type of data that you want to store, you might want to extend the schema. For more information on deciding when to extend the schema, see [Extending the Schema](#). When you have decided that schema extension is required, use the following procedure to extend the schema.

Verify Active Directory functionality before you apply any schema extensions

Verify Active Directory functionality before you update the schema to help ensure that the schema extension proceeds without error. At a minimum, ensure that all domain controllers for the forest are online and performing inbound replication.

To verify Active Directory functionality before you apply the schema extension

1. Log on to an administrative workstation that has the Windows Support Tool Repadmin.exe installed.

NOTE

The Support Tools are located on the operating system installation media in the Support\Tools folder.

2. Open a command prompt, and then change directories to the folder in which the Windows Support Tools are installed.
3. At a command prompt, type the following, and then press ENTER:

```
repadmin /replsum /bysrc /bydest /sort:delta
```

All domain controllers should show 0 in the Fails column, and the largest deltas (which indicate the number of changes that have been made to the Active Directory database since the last successful replication) should be less than or roughly equal to the replication frequency of the site link that is used by the domain controller for replication. The default replication frequency is 180 minutes.

For more information about additional steps that you can take to verify Active Directory functionality before you apply the schema extension, see [article 325379 in the Microsoft Knowledge Base](#).

To Extend the Schema

1. Determine the method of extension. Once you have carefully designed your schema changes, the next step is to decide which method to use to extend the schema. You can use one of the following methods:
 - Manually, using import files. See the documentation [Using the LDIFDE Tool](#).

NOTE

Do not use LDIFDE to import Windows Sch*.ldf files. Those files are required to extend the Active Directory schema in order to install domain controllers that run a newer version of Windows Server than the version that is running on the current schema master. When you need to extend the schema in order to install a new domain controller, use Adprep.exe.

- Programmatically, using an installation program. For more information, see [Programmatic Extension](#)
2. Enable Schema Changes. For more information, see [Prerequisites for Installing a Schema Extension](#) and [Enabling Schema Changes at the Schema Master](#).
 3. Obtain an Object Identifier (OID) for your new attributes and/or classes, as described in [Obtaining an Object Identifier](#).
 4. Create new attributes and classes.
 5. Use display specifiers to integrate new attributes and classes with the user interface, if necessary.
 6. Update the schema cache as described in [Updating the Schema Cache](#).
 7. Verify the schema extension using LDPexe.

Related topics

[Obtaining an Object Identifier](#)

[How To Use the Directory Service Command-Line Tools to Manage Active Directory Objects in Windows Server 2003](#)

[Using the LDIFDE Tool](#)

[Extending the Active Directory Schema](#)

[Restrictions on Schema Extension](#)

Naming Attributes and Classes

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes guidelines for naming attributes and classes.

To create a new class or attribute, adhere to the following naming rules:

- Use the same name for both the **cn** and **IDAPDisplayName** properties of a new **attributeSchema** or **classSchema** object.
- Identify the company with a lower-case prefix in the first section of the name. This prefix can be a DNS name, acronym, or other string that uniquely identifies the company. The prefix ensures that all attributes and classes for a specific company are displayed consecutively when browsing the schema.
- If you are developing a schema extension as an independent software vendor, add an abbreviation of the product name of the prefix. This adds distinction between multiple products that contain LDAP schema extensions.
- Use a hyphen as the next character after the prefix.
- Specify an attribute or class name that is unique within the company's attributes after the hyphen. This part of the common-name should be descriptive. Do not use illogical names that are meaningless to developers and viewers of the schema.

For example, if the fictitious Fabrikam company extended the schema by adding an attribute for storing a voice-mail identifier, the **cn** and **IDAPDisplayName** of the new attribute could be "fabrikam-VoiceMailID".

If the **IDAPDisplayName** of an attribute or class is not specified, the system uses the **cn** to generate one. However, the system algorithm for generating the name may result in name collisions or names that are difficult to read. To avoid these problems, it is recommended that an **IDAPDisplayName** be explicitly specified for all attributes and classes.

For development and testing purposes, it may be desirable to append a version suffix to the **cn** and **IDAPDisplayName**, for example, "fabrikam-VoiceMailID-001". In a distributed development/test environment, a version suffix enables developers to run multiple versions of their software simultaneously. After testing is complete, rename the attribute or class to remove the suffix.

It is not possible to delete defunct versions of a schema extensions, but it is possible to disable them and rename them with obscure names. For more information, see [Disabling Existing Classes and Attributes](#).

Disabling Existing Classes and Attributes

6/3/2022 • 3 minutes to read • [Edit Online](#)

Schema additions are permanent. You cannot delete **attributeSchema** and **classSchema** objects. In a distributed system, it is difficult to guarantee that there are no instances of a given class or attribute. Removing the definition of a class or attribute damages existing instances of that class or attribute.

You can disable an existing class or attribute by marking it as "defunct". This does not affect existing instances of the class or attribute so marked, but it prevents the creation of new instances.

The following restrictions apply when disabling schema classes and attributes:

- You cannot disable a Category 1 class or attribute.
- You cannot disable an attribute that is a member of a class that is not also disabled. This is because an attribute might be required for the (not disabled) class, and disabling the attribute prevents new instances of the class from being created.

To disable an attribute, set the **isDefunct** attribute of its **attributeSchema** object to **TRUE**. When an attribute is disabled, new instances of the attribute cannot be created. To re-enable the attribute set the **isDefunct** attribute to **FALSE**.

To disable a class, set the **isDefunct** attribute of its **classSchema** object to **TRUE**. When a class is disabled, new instances of the class cannot be created. To re-enable the class set the **isDefunct** attribute to **FALSE**.

Setting schema objects as defunct can be useful in production environments. When a test version of a schema extension is no longer required, mark it as defunct. You can restore it by removing the **isDefunct** attribute or setting the attribute value to **FALSE**. This also protects against an unintended removal of a schema object by setting it to defunct because the operation can be easily reversed.

Be aware that the Active Directory server does not clean up existing instances of an attribute or class when you make a schema object defunct. If you remove the **isDefunct** property, any instances become valid, normal objects again.

The following list includes other consequences of marking an **attributeSchema** or **classSchema** object defunct:

- Addition or modification of an instance fails.
- Error codes behave as if a defunct class never existed.
- Search and delete behave as if no schema objects have been made defunct.
- Only allow deleting entire attribute from object.

The following list includes additional options in a production environment for reducing the impact of defunct schema extensions:

- Remove all **mayHave** attribute values from a defunct class.
- Reduce the size of all **mustHave** attribute values from a defunct class.
- Remove a defunct attribute from the global catalog.
- Remove a defunct attribute from any index.

Other options for removing unwanted schema changes in a production environment are for developers to use a private domain controller for testing. In this case, you can:

- "Reset" the Active Directory server by using Dcpromo.exe to demote the DC. After the demote completes, use

Dcpromo.exe again to promote the server back to a DC. The developer can then use LDIF scripts to reload any required classes, attributes, and object instances.

- Use Ntbackup.exe to perform a system-state backup to an available disk partition. Reboot to Safe/Directory Service Restore mode to restore.

For Windows Server 2003 operating systems, when you set a class or attribute to defunct, you can immediately reuse the **ldapDisplayName**, **schemaIdGuid**, **OID** and **mapID** values of the defunct schema element when you create a new class or attribute to replace it. The defunct version of the class or attribute is maintained in the Schema container, but it is hidden in the MMC snap-in. To reactivate the old schema element, set **isDefunct** to **FALSE**.

The following LDIF code example shows how to modify the **isDefunct** attribute and change the RDN so that it is not confused with the new class that you create to replace it.

```
dn: CN=MyClass,CN=Schema,CN=Configuration,DC=X
changetype: modify
replace: isDefunct
isDefunct: TRUE
-

dn: CN=MyClass,CN=Schema,CN=Configuration,DC=X
changetype: modrdn
newrdn: cn=MyClassOld
deleteoldrdn: 1

dn:
changetype: modify
add: schemaUpdateNow
schemaUpdateNow: 1
-
```

Use the following command to run the LDIF code example against a forest for a computer running on Windows Server 2003 operating systems.

```
ldifde /i /f rdn.ldf /c "DC=X" "dc=mydomain,dc=com"
```

(Where "DC=X" is a constant)

Obtaining a Link ID

6/3/2022 • 2 minutes to read • [Edit Online](#)

Starting with Windows Server 2003, it is no longer necessary to request a **linkID** from Microsoft; there is a process for automatically generating a **linkID**. The system will automatically generate a link ID for a new linked attribute when the attribute's **linkID** attribute is set to 1.2.840.113556.1.2.50. A back link for this forward link is created by setting the **linkID** to the **attributeID** or **ldapDisplayName** of the forward link. The schema cache must be reloaded after creating the forward link and before creating the back link. Otherwise, the forward link's **attributelD** or **ldapDisplayName** will not be found when the back link is created. The schema cache is reloaded on demand, a few minutes after a schema change is made, or when the DC is restarted. Create all forward links, reload the schema cache and then create all back links.

For example, when you have created the new attributes **myForwardLinkAttr** and **myBackLinkAttr**, and wish to link them:

NOTE

The OIDs in this example are contrived. See [Obtaining an Object Identifier from Microsoft](#) for instructions on obtaining OIDs for new attributes.

1. Set these attributes on the attribute that is to be the forward link:

```
ldapDisplayName: myForwardLinkAttr  
OID: 1.2.840.113556.6.1234  
LinkID: 1.2.840.113556.1.2.50
```

2. Reload the Schema.

3. Set these attributes on the attribute that is to be the back link:

```
ldapDisplayName: myBackLinkAttr  
OID: 1.2.840.113556.6.1235  
LinkID: 1.2.840.113556.6.1234 or myForwardLinkAttr
```

Related topics

[Linked Attributes](#)

[How to Extend the Schema](#)

Obtaining an Object Identifier

6/3/2022 • 2 minutes to read • [Edit Online](#)

Two of the most common ways to get an Object Identifier (OID) are discussed in the following topics:

- [Obtaining a Root OID from an ISO Name Registration Authority](#)
- [Obtaining an Object Identifier from Microsoft](#)

Obtaining a Root OID from an ISO Name Registration Authority

6/3/2022 • 2 minutes to read • [Edit Online](#)

The preferred way to obtain a root object identifier (OID) is to request one from an International Standards Organization (ISO) Name Registration Authority. This is a one-time action; when you have obtained a root OID, the OID space it defines is yours and you can administer it yourself.

There is usually a fee associated with registering an organization name and receiving a root OID. Fees and registration policies vary by country/region. In the United States, the ISO NRA is the American National Standards Institute (ANSI). For more information about the ANSI registration procedure and fee schedule, see <https://ansi.org>. The ISO maintains a list of member organizations at <https://www.iso.org>. If you are outside the United States, you should contact the ISO member organization for your country/region for name registration information.

Regardless of the source used to get the OID, if you intend to extend the Active Directory schema and wish to apply for the Certified for Windows logo, you must register your OID with Microsoft. For more information about how to register your OID with Microsoft, see [Obtaining an Object Identifier from Microsoft](#).

Obtaining an Object Identifier from Microsoft

6/3/2022 • 4 minutes to read • [Edit Online](#)

To extend the Active Directory schema successfully you can obtain a root OID from a script shown below. The OIDs generated from the script are unique; they are mapped from a unique GUID. Please read the best practices carefully as poorly handled OIDs can result in data loss.

NOTE

For instructions on obtaining a link-Id from Microsoft, please visit the [Linked Attributes](#) topic.

After You Have Obtained a Base OID

Once you have a base OID, be careful when deciding how the OIDs should be divided into categories, because these OIDs are contained in the prefix table and are part of the DC replication data. It is recommended that no more than two OID categories be created.

You can create subsequent OIDs for new schema classes and attributes by appending digits to the OID in the form of OID.X, where X may be any number that you choose. A common schema extension generally uses the following structure:

If your assigned base OID was 1.2.840.113556.1.8000.999999, you might create categories as follows.

OID BASE VALUE	DESCRIPTION
1.2.840.113556.1.8000.999999.1	Application Classes The first class would have the OID 1.2.840.113556.1.8000.999999.1.1, the second class would have the OID 1.2.840.113556.1.8000.999999.1.2, and so on.
1.2.840.113556.1.8000.999999.2	Application Attributes The first attribute's OID would be 1.2.840.113556.1.8000.999999.2.1, the second attribute's OID would be 1.2.840.113556.1.8000.999999.2.2, and so on.

Script

```
' oidgen.vbs
'
' THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
' OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR
' FITNESS FOR A PARTICULAR PURPOSE.
'
' Copyright (c) Microsoft Corporation. All rights reserved
' Improvements made by Ryein C. Goddard
'
' This script is not supported under any Microsoft standard support program or service.
' The script is provided AS IS without warranty of any kind. Microsoft further disclaims all
' implied warranties including, without limitation, any implied warranties of merchantability
' or of fitness for a particular purpose. The entire risk arising out of the use or performance
' of the scripts and documentation remains with you. In no event shall Microsoft, its authors,
' or anyone else involved in the creation, production, or delivery of the script be liable for
```

```

' Any damages whatsoever (including, without limitation, damages for loss of business profits,
' business interruption, loss of business information, or other pecuniary loss) arising out of
' the use of or inability to use the script or documentation, even if Microsoft has been advised
' of the possibility of such damages.
' -----
Function GenerateOID()
    'Initializing Variables
    Dim guidString, oidPrefix
    Dim guidPart0, guidPart1, guidPart2, guidPart3, guidPart4, guidPart5, guidPart6
    Dim oidPart0, oidPart1, oidPart2, oidPart3, oidPart4, oidPart5, oidPart6
    On Error Resume Next
    'Generate GUID
    Set TypeLib = CreateObject("Scriptlet.TypeLib")
    guidString = TypeLib.Guid
    'If no network card is available on the machine then generating GUID can result with an error.
    If Err.Number <> 0 Then
        Wscript.Echo "ERROR: Guid could not be generated, please ensure machine has a network card."
        Err.Clear
        WScript.Quit
    End If
    'Stop Error Resume Next
    On Error GoTo 0
    'The Microsoft OID Prefix used for the automated OID Generator
    oidPrefix = "1.2.840.113556.1.8000.2554"
    'Split GUID into 6 hexadecimal numbers
    guidPart0 = Trim(Mid(guidString, 2, 4))
    guidPart1 = Trim(Mid(guidString, 6, 4))
    guidPart2 = Trim(Mid(guidString, 11, 4))
    guidPart3 = Trim(Mid(guidString, 16, 4))
    guidPart4 = Trim(Mid(guidString, 21, 4))
    guidPart5 = Trim(Mid(guidString, 26, 6))
    guidPart6 = Trim(Mid(guidString, 32, 6))
    'Convert the hexadecimal to decimal
    oidPart0 = CLng("&H" & guidPart0)
    oidPart1 = CLng("&H" & guidPart1)
    oidPart2 = CLng("&H" & guidPart2)
    oidPart3 = CLng("&H" & guidPart3)
    oidPart4 = CLng("&H" & guidPart4)
    oidPart5 = CLng("&H" & guidPart5)
    oidPart6 = CLng("&H" & guidPart6)
    'Concatenate all the generated OIDs together with the assigned Microsoft prefix and return
    GenerateOID = oidPrefix & "." & oidPart0 & "." & oidPart1 & "." & oidPart2 & "." & oidPart3 & _
        "." & oidPart4 & "." & oidPart5 & "." & oidPart6
End Function

```

```

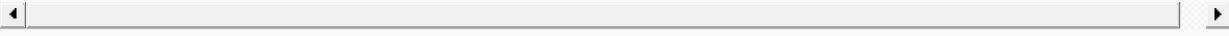
Set oShell = WScript.CreateObject ("WScript.Shell")
oShell.run "cmd /c Regsvr32 Schmmgmt.dll"

Set objFSO=CreateObject("Scripting.FileSystemObject")
outFile="C:\Users\Administrator\Desktop\oidInfo.txt"
Set objFile = objFSO.CreateTextFile(outFile,True)

'Output the resulted OID with best practice info
oidText = "Your root OID is: " & VBCRLF & GenerateOID & VBCRLF & VBCRLF & VBCRLF & _
"This prefix should be used to name your schema attributes and classes. For example: " & _
"if your prefix is ""Microsoft"", you should name schema elements like ""microsoft-Employee-ShoeSize"".
" & _
"For more information on the prefix, view the Schema Naming Rules in the server " & _
"Application Specification (http://www.microsoft.com/windowsserver2003/partners/isvs/appspec.mspx). " & _
VBCRLF & VBCRLF & _
"You can create subsequent OIDs for new schema classes and attributes by appending a .X to the OID where
X may " & _
"be any number that you choose. A common schema extension scheme generally uses the following
structure:" & VBCRLF & _
"If your assigned OID was: 1.2.840.113556.1.8000.2554.999999" & VBCRLF & VBCRLF & _
"then classes could be under: 1.2.840.113556.1.8000.2554.999999.1 " & VBCRLF & _
"which makes the first class OID: 1.2.840.113556.1.8000.2554.999999.1.1" & VBCRLF &

```

```
which makes the first class OID: 1.2.840.113556.1.8000.2554.999999.1.1 & VBCRLF & _  
"the second class OID: 1.2.840.113556.1.8000.2554.999999.1.2 etc..." & VBCRLF & VBCRLF & _  
"Using this example attributes could be under: 1.2.840.113556.1.8000.2554.999999.2 " & VBCRLF & _  
"which makes the first attribute OID: 1.2.840.113556.1.8000.2554.999999.2.1 " & VBCRLF & _  
"the second attribute OID: 1.2.840.113556.1.8000.2554.999999.2.2 etc..." & VBCRLF & VBCRLF & _  
"Here are some other useful links regarding AD schema:" & VBCRLF & _  
"Understanding AD Schema" & VBCRLF & _  
"http://technet2.microsoft.com/WindowsServer/en/Library/b7b5b74f-e6df-42f6-a928-e52979a512011033.mspx "  
& _  
VBCRLF & VBCRLF & _  
"Developer documentation on AD Schema:" & VBCRLF & _  
"http://msdn2.microsoft.com/en-us/library/ms675085.aspx " & VBCRLF & VBCRLF & _  
"Extending the Schema" & VBCRLF & _  
"http://msdn2.microsoft.com/en-us/library/ms676900.aspx " & VBCRLF & VBCRLF & _  
"Step-by-Step Guide to Using Active Directory Schema and Display Specifiers " & VBCRLF & _  
  
"http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/activedirectory/howto/adschema.ms  
px " & _  
VBCRLF & VBCRLF & _  
"Troubleshooting AD Schema " & VBCR & _  
"http://technet2.microsoft.com/WindowsServer/en/Library/6008f7bf-80de-4fc0-ae3e-51eda0d7ab651033.mspx "  
& _  
VBCRLF & VBCRLF  
  
objFile.Write oidText  
objFile.Close
```



Integrating Schema Extensions with the User Interface

6/3/2022 • 2 minutes to read • [Edit Online](#)

The administration tools of Active Directory Domain Services use a common architecture for connecting the administrative user interface to the directory schema. The centerpiece of this architecture is the **displaySpecifier** object class. Display specifiers and their usage are described in detail in [Extending the User Interface for Directory Objects](#).

When you create a new class by subclassing an existing class, you can leverage the existing UI for the superclass by copying the superclass' display specifier. An easy way to copy the display specifier for a class is to export it into an LDIF file using LDIFDE, edit the Distinguished name and CN, then import the modified LDIF file. If the subclass has additional attributes, you will need to provide additional property pages to support editing. If the subclass has additional must-have properties, you will need to provide Creation Dialog extension pages since the UI provided by the superclass has no way to create these new attributes.

Defining a New Attribute

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic shows how to define a new attribute when extending the Active Directory schema.

Consider the following when defining a new attribute:

- Use existing attributes when possible.
- Always use the **cn** ("common name") property as the naming (relative distinguished name) attribute. This is the default for most classes, including those derived directly from **Top**. The **cn** property is an indexed property and will make searching for objects by name more efficient.
- Large multi-valued attributes are costly to store and retrieve and should be avoided. Active Directory Domain Services implement an LDAP extension to enable incremental read of large properties with multiple values, but not all LDAP clients will recognize this extension.
- Remember that attributes are flat; that is there is no implied substructure to an attribute. All attributes in a given class should relate directly to instances of that class.

Creating a New Attribute

To create a new attribute:

- Choose a name for the attribute. The name will be contained in the **cn** and **IDAPDisplayName** attributes. For more information about composing a name for a new attribute, see [Naming Attributes and Classes](#).
- Obtain an object identifier (OID) for the attribute. For more information, see [Obtaining a Root Object Identifier](#).
- Choose a syntax for the attribute. The syntax is determined by the combination of the **oMSyntax** and **oMObjectClass** attributes. For more information, see [Choosing a Syntax](#).
- Decide if the attribute is single or multi-valued. The **isSingleValued** attribute determines if the attribute is single or multi-valued.
- Decide if the attribute should be indexed by default. For more information, see [Indexed Attributes](#).
- Decide if the attribute should be in the global catalog by default. For more information, see [Attributes Included in the Global Catalog](#).
- If the attribute is an integer or string, decide if a range limit is required. The **rangeLower** and **rangeUpper** attributes are used to specify the range limit.
- If the attribute is DN-valued, decide if the attribute should be linked with another attribute. If so, the **linkID** attribute must be set appropriately on each attribute; one attribute must be a forward link, the other a back link. For more information about linked attributes, see [Linked Attributes](#).
- Create a new **attributeSchema** object in the schema container and set the appropriate attributes for the object. There are a large number of attributes that can be set for an **attributeSchema** object, but the attributes listed in the following table below are critical to the definition of a new attribute. The values of these attributes are determined by the previous steps. For more information about these attributes, see [Characteristics of Attributes](#).

ATTRIBUTE	COMMENT
cn	Required.
IDAPDisplayName	Required.
adminDisplayName	Required.
attributeSyntax	Required.
oMSyntax	Required.
oMObjectClass	Required.
schemaIDGUID	Required.
attributeID	Required.
isSingleValued	Required.
searchFlags	Required.
isMemberOfPartialAttributeSet	Required.
rangeLower	Optional.
rangeUpper	Optional.
linkID	Optional. Required for linked attributes.
description	Optional.

- Commit the new **attributeSchema** object to the schema container.
- Update the schema cache, if necessary. For more information, see [Updating the Schema Cache](#).

Example Code for Creating an Attribute

6/3/2022 • 5 minutes to read • [Edit Online](#)

The following code example creates an **attributeSchema** object in the schema container.

The **CreateAttribute** function creates an **attributeSchema** object in the schema container, but does not commit it to the directory. Call the **IADs.SetInfo** method to commit the new **attributeSchema** object to the directory.

The **BytesToVariantArray** function is a utility function that packs an octet string into a variant array.

```
*****  
BytesToVariantArray()  
  
Converts an arrya of BYTES into a VARIANT array.  
  
Parameters:  
  
pValue = Contains an array of BYTES to convert. cValueElements  
contains the number of elements in this array.  
  
cValueElements - Contains the number of elements in the pValue  
array.  
  
pVariant - Receives the VARIANT that contains an octet string  
(VT_UI1 | VT_ARRAY)  
*****/  
  
HRESULT BytesToVariantArray(  
    PBYTE pValue,  
    ULONG cValueElements,  
    VARIANT *pVariant  
)  
{  
    HRESULT hr = E_FAIL;  
    SAFEARRAY *pArrayVal = NULL;  
    SAFEARRAYBOUND arrayBound;  
    CHAR HUGEPEP *pArray = NULL;  
  
    // Check parameters.  
    if((NULL == pValue) || !(cValueElements > 0))  
    {  
        return E_INVALIDARG;  
    }  
  
    // Set bound for array.  
    arrayBound.lLbound = 0;  
    arrayBound.cElements = cValueElements;  
  
    // Create the safe array for the octet string. unsigned char  
    // elements;single dimension;aBound size.  
    pArrayVal = SafeArrayCreate(VT_UI1, 1, &arrayBound);  
  
    if (!(NULL == pArrayVal))  
    {  
        hr = SafeArrayAccessData(pArrayVal,  
            (void HUGEPEP * FAR *) &pArray);  
        if (SUCCEEDED(hr))  
        {  
            // Copy the bytes to the array.  
            for (ULONG i = 0; i < cValueElements; i++)  
            {  
                pArray[i] = pValue[i];  
            }  
        }  
    }  
}
```

```

    // Copy the bytes to the safe array.
    memcpy(pArray, pValue, arrayBound.cElements);
    SafeArrayUnaccessData(pArrayVal);

    // Set type to array of unsigned char.
    V_VT(pVariant) = VT_ARRAY | VT_UI1;

    // Assign the safe array to the array member.
    V_ARRAY(pVariant) = pArrayVal;

    hr = S_OK;
}
else
{
    // Cleanup if the array cannot be accessed.
    if (pArrayVal)
    {
        SafeArrayDestroy(pArrayVal);
    }
}
else
{
    hr = E_OUTOFMEMORY;
}

return hr;
}

```

CreateAttribute()

Creates a new attribute in the schema container
 IADsContainer.Create.
 This function creates the attributeSchema object, but does not commit it to the directory. The caller must call IADs.SetInfo on the new attribute object to commit it to the directory.

Parameters:

pwszAttributeName - Contains a null-terminated string that contains the name of the attribute.

pwszLDAPDisplayName - Contains a null-terminated string that contains the LDAPDisplayName of the attribute.

pwszAttributeOID - Contains a null-terminated string that contains the OID of the attribute.

pSchemaIDGUID -
 pwszAttributeSyntax - Contains a null-terminated string that contains the syntax of the attribute.

i0mSyntax - Contains the value for the oMSyntax attribute.

pbomObjectClass - Pointer to a BYTE array that contains the value for the oMObjectClass attribute. dwSizeomObjectClass contains the number of elements in this array.

dwSizeomObjectClass - Contains the number of elements in the pbomObjectClass array.

pwszDescription - Contains a null-terminated string that contains the description of the attribute.

fIsSingleValued - Contains TRUE if the attribute is single-valued or FALSE if the attribute is multi-valued.

fIsInGC - Contains TRUE if the attribute is contained in the global catalog or FALSE if the attribute should not be contained in the global catalog.

fIsIndexed - Contains TRUE if the attribute is indexed or FALSE if the attribute is not indexed.

iLowerRange - Contains the lower range of the attribute value. Contains zero if the range should not be set.

iUpperRange - Contains the upper range of the attribute value. Contains zero if the range should not be set.

iLinkID - Contains the value for the linkID attribute. If this contains zero, the linkID attribute is not set.

pwszAdminDisplayName - Contains a null-terminated string that contains the admin display name of the attribute.

ppadsNewAttribute - Receives a pointer to the IADS interface of the attribute object created.

******/

```
HRESULT CreateAttribute(
    LPCWSTR pwszAttributeName,
    LPCWSTR pwszLDAPDisplayName,
    LPCWSTR pwszAttributeOID,
    const GUID * pSchemaIDGUID,
    LPCWSTR pwszAttributeSyntax,
    int iOmSyntax,
    PBYTE pbomObjectClass,
    DWORD dwSizeomObjectClass,
    LPCWSTR pwszDescription,
    BOOL fIsSingleValued,
    BOOL fIsInGC,
    BOOL fIsIndexed,
    int iLowerRange,
    int iUpperRange,
    int iLinkID,
    LPCWSTR pwszAdminDisplayName,
    IADS **ppadsNewAttribute
)
{
    if(!ppadsNewAttribute)
    {
        return E_POINTER;
    }

    *ppadsNewAttribute = NULL;

    if( (!pwszAttributeName) ||
        (!pwszAttributeOID) ||
        (!pwszAttributeSyntax) ||
        (!iOmSyntax)
    )
    {
        return E_INVALIDARG;
    }

    if(iLowerRange < iUpperRange)
    {
        return E_INVALIDARG;
    }

    HRESULT hr;
    CComPtr<IADS> spRoot;

    hr = ADsGetObject(L"LDAP://RootDSE",
```

```

        IID_IADs,
        (void**)&spRoot);
if(FAILED(hr))
{
    return hr;
}

CComVariant svarSchema;

// Get the distinguished name of the schema container.
hr = spRoot->Get(CComBSTR("schemaNamingContext"), &svarSchema);
if(FAILED(hr))
{
    return hr;
}

// Bind to the IADsContainer interface of the schema container.
CComPtr<IADsContainer> spSchemaCont;
hr = ADsOpenObject( L"LDAP://RootDSE",
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION,
                    IID_IADs,
                    (void**)&spSchemaCont);
if(FAILED(hr))
{
    return hr;
}

CComPtr<IDispatch> spDisp;

CComBSTR sbstrAttribute = "CN=";
sbstrAttribute += pwszAttributeName;

// Create the object in the schema container.
hr = spSchemaCont->Create(CComBSTR("attributeSchema"),
                            sbstrAttribute,
                            &spDisp);
CComPtr<IADs> spNewAttribute;
hr = spDisp->QueryInterface(IID_IADs, (void**)&spNewAttribute);
if(FAILED(hr))
{
    return hr;
}

CComBSTR sbstrAttributeToSet;
CComVariant svar;

// Put this value so that it can be read from the cache.
sbstrAttributeToSet = "cn";
svar = pwszAttributeName;
hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
if(FAILED(hr))
{
    return hr;
}

// Put LDAPDisplayName.
// If NULL, let it default; that is, do not set it.
if (pwszLDAPDisplayName)
{
    sbstrAttributeToSet = "LDAPDisplayName";
    svar = pwszLDAPDisplayName;
    hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
    if(FAILED(hr))
    {
        return hr;
    }
}

```

```

// Put attributeID.
sbstrAttributeToSet = "attributeID";
svar = pwszAttributeOID;
hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
if(FAILED(hr))
{
    return hr;
}

// Put schemaIDGUID.
// If NULL, let it default; that is, do not set it.
if (pSchemaIDGUID)
{
    hr = BytesToVariantArray(
        (LPBYTE)pSchemaIDGUID,
        sizeof(GUID),
        &svar
    );
    if(FAILED(hr))
    {
        return hr;
    }

    sbstrAttributeToSet = "schemaIDGUID";
    hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
    if(FAILED(hr))
    {
        return hr;
    }
}

// Put attributeSyntax.
sbstrAttributeToSet = "attributeSyntax";
svar = pwszAttributeSyntax;
hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
if(FAILED(hr))
{
    return hr;
}

// Put oMSyntax.
sbstrAttributeToSet = "oMSyntax";
svar = i0mSyntax;
hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
if(FAILED(hr))
{
    return hr;
}

// Put searchFlags.
sbstrAttributeToSet = "searchFlags";
if (fIsIndexed)
{
    svar = 1;
}
else
{
    svar = 0;
}
hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
if(FAILED(hr))
{
    return hr;
}

// Put isSingleValued.
sbstrAttributeToSet = "isSingleValued";

```

```

    if (fIsSingleValued)
    {
        svar = VARIANT_TRUE;
    }
    else
    {
        svar = VARIANT_FALSE;
    }
    hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
    if(FAILED(hr))
    {
        return hr;
    }

    // Put isMemberOfPartialAttributeSet.
    sbstrAttributeToSet = "isMemberOfPartialAttributeSet";
    if (fIsInGC)
    {
        svar = VARIANT_TRUE;
    }
    else
    {
        svar = VARIANT_FALSE;
    }
    svar.vt = VT_BOOL;
    hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
    if(FAILED(hr))
    {
        return hr;
    }

    // Put description.
    sbstrAttributeToSet = "description";
    svar = pwszDescription;
    hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
    if(FAILED(hr))
    {
        return hr;
    }

    // Put rangeLower and rangeUpper.
    // If both are 0, let them default; that is do not set them.
    if ((iLowerRange >= 0) && (iUpperRange > 0))
    {
        // Set rangeUpper.
        sbstrAttributeToSet = "rangeUpper";
        svar = iUpperRange;
        hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
        if(FAILED(hr))
        {
            return hr;
        }

        // Set rangeLower.
        sbstrAttributeToSet = "rangeLower";
        svar = iLowerRange;
        hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
        if(FAILED(hr))
        {
            return hr;
        }
    }

    // Put linkID. If linkID is 0, let it default; that
    // is, do not set it.
    if (iLinkID>0)
    {

```

```

        sbstrAttributeToSet = "linkID";
        svar = iLinkID;
        hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
        if(FAILED(hr))
        {
            return hr;
        }
    }

    // Put adminDisplayName.
    // If NULL, set it to the same string as cn.
    sbstrAttributeToSet = "adminDisplayName";
    if (!pwszAdminDisplayName)
    {
        svar = pwszAttributeName;
    }
    else
    {
        svar = pwszAdminDisplayName;
    }
    hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
    if(FAILED(hr))
    {
        return hr;
    }

    // Put omObjectClass.
    // Only set this if used to delineate omSyntax 127 attributes.
    if (pbomObjectClass && dwSizeomObjectClass)
    {
        hr = BytesToVariantArray(
            pbomObjectClass,
            dwSizeomObjectClass,
            &svar
        );
        if(FAILED(hr))
        {
            return hr;
        }
    }

    sbstrAttributeToSet = "omObjectClass";
    hr = spNewAttribute->Put(sbstrAttributeToSet, svar);
    if(FAILED(hr))
    {
        return hr;
    }
}

// Get a copy of the interface to return.
hr = spNewAttribute->QueryInterface(IID_IADs,
                                    (LPVOID*)ppadsNewAttribute);

return hr;
}

```

Choosing a Syntax

6/3/2022 • 2 minutes to read • [Edit Online](#)

There are 23 syntaxes defined in Active Directory Domain Services. This topic contains a list of recommended syntaxes to use when defining a new attribute. For more information, see [Syntaxes for Attributes in Active Directory Domain Services](#).

The following table provides a list of recommendations.

DATA TO STORE IN ATTRIBUTE	SYNTAX TO USE	COMMENT
Binary data	String(Octet)	Use to store binary data. This is an array of bytes.
Binary data with a DN reference	Object(DN-Binary)	Contains a binary value and a distinguished name (DN). The Active Directory server keeps the DN up-to-date.
Boolean	Boolean	Use for boolean values.
DN Reference	Object(DS-DN)	Use to store distinguished names that you want kept up-to-date by the Active Directory server. When an attribute of DN syntax is created with a valid DN, the server treats the attribute as a reference to the object represented by the DN that was set. If the referenced object is renamed or moved, the server ensures that the attribute reflects the change. If the attribute is reset with a new DN, the attribute is reference to the object represented by the new DN.
Integer	Integer	Use for integers.
Large Integer (64-bit values)	LargeInteger	Use for 64-bit values.
Linked DN	Object(DS-DN)	This string syntax can be used for linked DNs. Back links must be of syntax DN. Forward links can be of syntax DN as well as Object(DN-String) , Object(DN-Binary) , Object(Access-Point) , or Object(OR-Name) . Linked attributes must have a linkID defined. See the description of linkID in Attribute-Schema properties.
Security Descriptor	String(NT-Sec-Desc)	Octet string containing a security descriptor.

DATA TO STORE IN ATTRIBUTE	SYNTAX TO USE	COMMENT
Security Identifier (SID)	String(Sid)	Octet string containing a security identifier (SID). Use this syntax to store SID values only.
String	String(Unicode)	Use for most string attributes. It supports the Unicode character set. When the Active Directory server performs comparisons against attributes of this syntax (such as evaluating a query), it performs case-insensitive comparisons. Use the other string syntaxes (String(IA5) , String(Numeric) , and so on) to store strings that should contain only the specific character sets supported by the syntax.
String data with a DN reference	Object(DN-String)	String containing a string value and a distinguished name (DN). The Active Directory server keeps the DN up-to-date.
Time	String(Generalized-Time)	Use the String(Generalized-Time) syntax to store time values rather than the String(UTC-Time) syntax because String(Generalized-Time) uses four characters for the year and String(UTC-Time) uses only two.

Defining a New Class

6/3/2022 • 2 minutes to read • [Edit Online](#)

When you define a new class, specify the legal parent classes of the new class, that is, the classes that can contain instances of your new class. The legal parent classes are specified in the **possSuperiors** and **systemPossSuperiors** attributes of the new class, as well as in the possible superiors inherited from its superclasses, but not from auxiliary classes.

Be specific when defining the legal parent classes for the new class. Decide where you want users to create instances of your class. For example, specifying "container" as a legal parent will enable the user create instances under any of the standard containers (**container**, **organizationalUnit**, and so on), while specifying "computer" would enable instances to be created only under instances of the **computer** object.

To Create a Class

1. Choose a name for the class. For more information about composing a common-name and an LDAP display name for a new class, see [Naming Attributes and Classes](#).
2. Obtain an object identifier (OID) for the class. For more information, see [Obtaining an Object Identifier](#).
3. Choose a "default object category" for the class. For more information, see [Object Class and Object Category](#).
4. Choose an "object class category" for the class. This indicates whether the class is abstract, structural, or auxiliary. For more information, see [Structural, Abstract, and Auxiliary Classes](#).
5. Create a new **classSchema** object. Many attributes can be set for an **classSchema** object. The following attributes are critical to the definition of a new class:
 - Classes from which the new class inherits: **subClassOf**, **auxiliaryClass**, and **systemAuxiliaryClass**
 - Names and identifiers for the new class: **cn**, **IDAPDisplayName**, **adminDisplayName**, **schemaIDGUID**, **governsID**
 - Possible attributes of the new class: **mustContain**, **systemMustContain**, **mayContain**, **systemMayContain**
 - Possible parents of the new class: **possSuperiors**, **systemPossSuperiors**
 - **objectClassCategory**
 - **defaultObjectCategory**
 - **defaultHidingValue**
 - **rDnAttId**
 - **defaultSecurityDescriptor**
 - **description** (optional)

For more information, and descriptions of these attributes, see [Characteristics of Object Classes](#).

Be aware that the classes specified in **subClassOf**, **possSuperiors**, **systemPossSuperiors**, **auxiliaryClass**, and **systemAuxiliaryClass**, must exist when the new class is written to the directory; otherwise, the **classSchema** object will fail to be added to the directory. Similarly, the attributes specified in **mustContain**, **systemMustContain**, **mayContain**, and **systemMayContain**, must exist or the class creation operation will fail.

6. Write the new **classSchema** object to the directory.

To add an attribute to the **mayContain** property

1. Obtain the **classSchema** object for the class to modify.
2. Add the new attribute to the **mayContain** multi-valued property.
3. Write the changed **classSchema** object back to the directory.

New attributes can be created at the same time as new classes; the order of creating the new attributes and classes is important. For more information, see [What the Installation Must Do](#).

To add new attributes and new classes at the same time

1. Define all of the new attributes first. For more information, see [Defining a New Attribute](#).
2. Update the Schema Cache before defining any new classes. For more information, see [Updating the Schema Cache](#).
3. Create the new classes.
4. Add the desired attributes to the new classes.
5. Update the Schema Cache again.

Example Code for Creating a Class

6/3/2022 • 5 minutes to read • [Edit Online](#)

The following C and C++ code example shows how to create a **classSchema** object in the ADSI cache.

The **CreateClass** example function shows how to create the **classSchema** object and returns an **IADs** pointer to the new object. Be aware that **CreateClass** does not call **IADs::SetInfo** to commit the new **classSchema** object to the directory. The caller must make the call using the returned pointer.

The following **BytesToVariantArray** code example shows a utility function that packs an octet string into a variant array.

```
HRESULT BytesToVariantArray(PBYTE pValue,
                            ULONG cValueElements,
                            VARIANT *pVariant);

///////////////////////////////
// CreateClass routine
// Calls IADsContainer::Create to create a new classSchema object in
// the schema container. Sets key properties of the new class.
// Be aware that the CreateClass routine does not commit the new class
// to the directory. The CreateClass caller must commit
// the new class by calling IADs::SetInfo on the returned
// pointer to the new class object.
/////////////////////////////
HRESULT CreateClass(
    IADsContainer *pSchema,
    LPOLESTR szClassName,
    LPOLESTR szLDAPDisplayName,
    LPOLESTR szClassOID,
    const GUID * pSchemaIDGUID,
    LPOLESTR szSubClassOf,
    LPOLESTR *arrayAuxiliaryClasses,
    DWORD dwSizearrayAuxiliaryClasses,
    LPOLESTR szDefaultObjectCategory,
    LPOLESTR szDescription,
    BOOL bDefaultHidingValue,
    LPOLESTR szDefaultSecurityDescriptor,
    LPOLESTR szRDNAttribute,
    int iObjectClassCategory,
    LPOLESTR *arrayPossibleSuperiors,
    DWORD dwSizearrayPossibleSuperiors,
    LPOLESTR *arrayMustContain,
    DWORD dwSizearrayMustContain,
    LPOLESTR *arrayMayContain,
    DWORD dwSizearrayMayContain,
    LPOLESTR szAdminDisplayName,
    IADs **ppNewClass // Return an IADs pointer to the new class.
)
{
    if ((!szClassName)
        || (!szClassOID)
        || (!szSubClassOf)
        || (!iObjectClassCategory)
        || (!arrayPossibleSuperiors)
    )
    {
        return E_INVALIDARG;
    }

    if (!ppNewClass)
```

```

{
    return E_POINTER;
}

HRESULT hr = E_FAIL;
IDispatch *pDisp = NULL;
LPOLESTR szStart = L"cn=";
DWORD dwLength = wcslen(szStart) + wcslen(szClassName) + 1;
LPOLESTR szBuffer = new OLECHAR[dwLength];
wcscpy_s(szBuffer,szStart);
wcscat_s(szBuffer,szClassName);
*ppNewClass = NULL;
VARIANT var;

VariantInit(&var);

if (!pSchema)
{
    return E_POINTER;
}

hr = pSchema->Create(CComBSTR("classSchema"),
                      CComBSTR(szBuffer),
                      &pDisp);
delete [] szBuffer;

if (SUCCEEDED(hr))
{
    hr = pDisp->QueryInterface(IID_IADS, (void**)ppNewClass);

    while (SUCCEEDED(hr))
    {
        // Put this value so that it can be read from the cache.
        var.vt = VT_BSTR;
        var.bstrVal = SysAllocString(szClassName);
        hr = (*ppNewClass)->Put(CComBSTR("cn"), var);
        VariantClear(&var);

        // Put LDAPDisplayName.
        // If NULL, let it default; that is do not set it.
        if (szLDAPDisplayName)
        {
            var.vt = VT_BSTR;
            var.bstrVal = SysAllocString(szLDAPDisplayName);
            hr = (*ppNewClass)->Put(CComBSTR("LDAPDisplayName"),
                                      var);
            VariantClear(&var);
        }

        // Put attributeID.
        var.vt = VT_BSTR;
        var.bstrVal = SysAllocString(szClassOID);
        hr = (*ppNewClass)->Put(CComBSTR("governSID"), var);
        VariantClear(&var);

        // Put schemaIDGUID.
        // If NULL, let it default; that is do not set it.
        if (pSchemaIDGUID)
        {
            hr = BytesToVariantArray(
                (LPBYTE)pSchemaIDGUID, // Pointer to bytes to
                                         // put in a variant array.
                sizeof(GUID),          // Size, in bytes, of pValue.
                &var // Return variant containing
                     // octet string (VT_UI1|VT_ARRAY).
            );
        }
    }
}

```



```

}

if (arrayPossibleSuperiors &&
    dwSizearrayPossibleSuperiors)
{
    // Build a Variant of array type, using the
    // specified string array.
    hr = ADsBuildVarArrayStr(arrayPossibleSuperiors,
                            dwSizearrayPossibleSuperiors,
                            &var);
    if (SUCCEEDED(hr))
    {
        // Verify that all the specified classes
        // are valid.
        hr = (*ppNewClass)->Put(CComBSTR("possSuperiors"),
                                var);
        VariantClear(&var);
    }
}

if (arrayMustContain && dwSizearrayMustContain)
{
    // Build a Variant of array type,
    // using the specified string array.
    hr = ADsBuildVarArrayStr(arrayMustContain,
                            dwSizearrayMustContain,
                            &var);
    if (SUCCEEDED(hr))
    {
        // Verify that all the specified
        // attributes are valid.
        hr = (*ppNewClass)->Put(CComBSTR("mustContain"),
                                var);
        VariantClear(&var);
    }
}

if (arrayMayContain && dwSizearrayMayContain)
{
    // Build a Variant of array type,
    // using the specified string array.
    hr = ADsBuildVarArrayStr(arrayMayContain,
                            dwSizearrayMayContain,
                            &var);
    if (SUCCEEDED(hr))
    {
        // Verify that all the specified
        // attributes are valid.
        hr = (*ppNewClass)->Put(CComBSTR("mayContain"),
                                var);
        VariantClear(&var);
    }
}

if (arrayAuxiliaryClasses &&
    dwSizearrayAuxiliaryClasses)
{
    // Build a Variant of array type,
    // using the specified string array.
    hr = ADsBuildVarArrayStr(arrayAuxiliaryClasses,
                            dwSizearrayAuxiliaryClasses,
                            &var);
    if (SUCCEEDED(hr))
    {
        // Verify that all the
        // specified classes are valid.
        hr = (*ppNewClass)->Put(CComBSTR("auxiliaryClass"),
                                var);
        VariantClear(&var);
    }
}

```

```

        var.vt = VT_VARIANT;
    }

    // Put description.
    var.vt = VT_BSTR;
    var.bstrVal = SysAllocString(szDescription);
    hr = (*ppNewClass)->Put(CComBSTR("description"),
        var);
    VariantClear(&var);

    // Put adminDisplayName.
    // If NULL, set it to the same string as cn.
    var.vt = VT_BSTR;
    if (!szAdminDisplayName)
    {
        var.bstrVal = SysAllocString(szClassName);
    }
    else
    {
        var.bstrVal = SysAllocString(szAdminDisplayName);
    }
    hr = (*ppNewClass)->Put(CComBSTR("adminDisplayName"),
        var);
    VariantClear(&var);

    // End of properties to set.
    break;
}
}

if (pDisp)
{
    pDisp->Release();
}

if (FAILED(hr))
{
    // Cleanup, if failed.
    if (*ppNewClass)
    {
        (*ppNewClass)->Release();
        (*ppNewClass) = NULL;
    }
}

return hr;
}

///////////////////////////////
// BytesToVariantArray
// Packs an octet string into a variant array.
/////////////////////////////
HRESULT BytesToVariantArray(
    PBYTE pValue, // Pointer to bytes to put in a
                 // variant array.
    ULONG cValueElements,// Size of pValue in bytes.
    VARIANT *pVariant // Return variant that contains
                      // octet string (VT_UI1|VT_ARRAY).
)
{
    HRESULT hr = E_FAIL;
    SAFEARRAY *pArrayVal = NULL;
    SAFEARRAYBOUND arrayBound;
    CHAR HUGEP *pArray = NULL;

    // Set bound for array.
    arrayBound.lLbound = 0;
    arrayBound.cElements = cValueElements;
}

```

```
arrayBound.cElements = cValueElements;

// Create the safe array for the octet string.
// unsigned char elements;single dimension;aBound size.
pArrayVal = SafeArrayCreate(VT_UI1, 1, &arrayBound);

if (!(pArrayVal == NULL) )
{
    hr = SafeArrayAccessData(pArrayVal,
                           (void HUGE * FAR *) &pArray );
    if (SUCCEEDED(hr))
    {
        // Copy bytes to the safe array.
        memcpy( pArray, pValue, arrayBound.cElements );
        SafeArrayUnaccessData( pArrayVal );
        // Set type to array of unsigned char.
        V_VT(pVariant) = VT_ARRAY | VT_UI1;
        // Assign the safe array to the array member.
        V_ARRAY(pVariant) = pArrayVal;
        hr = S_OK;
    }
    else
    {
        // Cleanup if the array cannot be accessed.
        if ( pArrayVal )
        {
            SafeArrayDestroy( pArrayVal );
        }
    }
}
else
{
    hr = E_OUTOFMEMORY;
}

return hr;
}
```

Installing Schema Extensions

6/3/2022 • 2 minutes to read • [Edit Online](#)

Schema updates will be strictly controlled at most customer sites - this is the recommended practice. If a service requires schema extensions the schema extensions must be separately installable. This allows customers to perform the schema update separately and in advance of the rest of the installation.

Documenting Schema Extensions

6/3/2022 • 2 minutes to read • [Edit Online](#)

In addition to providing a separate installation procedure for schema extensions, the nature of the schema extensions must be clearly documented. The documentation should contain:

- A list of any new classes and attributes with an explanation of each one
- A list of any standard classes that are extended with new attributes and a list of the attributes
- A list of any new property pages with an explanation of how to use each one
- An explanation of how the service uses the schema extensions

What the Installation Must Do

6/3/2022 • 2 minutes to read • [Edit Online](#)

Applications that extend the schema, must apply updates as described in the following procedure.

To apply updates when extending the schema

1. Add the new attributes.
2. Add the new classes.
3. Add the new attributes to classes.
4. Trigger a cache reload.

New attributes referenced in Step 3 must be referred to by its OID because the new attribute name is not present in the schema cache at this point.

Step 4 is unnecessary if the extensions will not be used immediately; the extensions will appear in the schema cache in approximately 5 minutes, depending on system load. For more information about the schema cache and how to trigger a cache reload, see [Updating the Schema Cache](#).

Updating the Schema Cache

6/3/2022 • 2 minutes to read • [Edit Online](#)

All information that is written to an Active Directory server is validated against the schema. The schema is held in memory on directory servers (domain controllers) for performance reasons. The in-memory version is updated automatically after the on-disk version has been updated. The automatic update occurs five minutes after the last change was applied; applying another change to the schema in the 5-minute window resets the timer for another 5 minutes. This behavior keeps the cache consistent, but can be confusing, because changes do not appear in the schema until the cache is updated, even though they were applied on disk.

To update the Active Directory schema cache after a schema update, or if you want to use the schema update for non-schema operations immediately, add the **schemaUpdateNow** attribute (it is an operational attribute) to the root DSE (blank DN) with value 1. A schema cache update will start immediately. The call is blocking. If the call returns with no error, the cache is updated and all schema updates are ready to be used. An error return indicates the cache update was unsuccessful. Applications that must use this feature should be designed to accommodate the blocking write, particularly in giving the user feedback, if the program or script executes interactively.

The following code example is a sample LDIFDE script that shows how to trigger a cache reload.

```
dn:  
changetype: modify  
add: schemaUpdateNow  
schemaUpdateNow: 1  
-
```

For more information about how to update the schema cache programmatically, see [Example Code for Updating the Schema Cache](#).

Prerequisites for Installing a Schema Extension

6/3/2022 • 2 minutes to read • [Edit Online](#)

To modify the schema, ensure you have the following rights and are aware of the following information:

- You must have sufficient rights to modify the schema. The schema contains all of the data definitions for Active Directory Domain Services for the entire enterprise. To update the schema, a user must be a member of the Schema Administrators group, or have been delegated the rights to update the schema.
- You can only make schema changes at the schema master. For more information, see [Detecting the Schema Master](#).
- The schema master must be writable; that is, the safety interlock in the registry must be removed. For more information, see [Enabling Schema Changes at the Schema Master](#).

Detecting the Schema Master

6/3/2022 • 4 minutes to read • [Edit Online](#)

Active Directory Domain Services have a multi-master update system; every domain controller holds a writable copy of the directory. Schema updates are propagated to all domains that belong to the same tree or forest. Because it is difficult to reconcile conflicting updates to the schema, schema updates can only be performed at a single server. The server with the right to perform updates can change, but only one server will have that right at any given time. This server is called the schema master. The first DC installed in an enterprise is, by default, the schema master.

Schema changes can only be made at the schema master level. To detect which DC is the schema master, perform the following steps.

Detecting the DC Schema Master

1. Read the **fsmoRoleOwner** attribute of the schema container on any DC. The **fsmoRoleOwner** attribute returns the distinguished name (DN) of the **nTDSDSA** object for the schema master.
2. Bind to the **nTDSDSA** object whose DN you just retrieved. The parent of this object is the server object for the DC containing the schema master.
3. Get the **ADsPath** for the parent of the **nTDSDSA** object. The parent is a server object.
4. Bind to the server object.
5. Get the **dNSHostName** attribute of the server object. This is the DNS name of the DC that contains the schema master.
6. Specify the DNS name of the schema master as the server and the DN as the DN of the schema container to bind to the schema master. For example, if the server was "dc1.fabrikam.com" and the schema container DN was "cn=schema,cn=configuration,dc=fabrikam,dc=com", the **ADsPath** would be as follows.

```
LDAP://dc1.fabrikam.com/cn=schema,cn=configuration,dc=fabrikam,dc=com
```

It is recommended that you find the schema master and bind to it to make schema changes. However, you can move the schema master to another server.

To make another server the schema master, a suitably privileged user can:

- Use the Schema Manager MMC snap-in.

NOTE

The Active Directory Schema MMC snap-in must be registered manually. To register the Schema snap-in, you must run the following command from the command prompt in the Windows System32 directory.

```
regsvr32.exe schmmgmt.dll
```

- Use the NTDSUTIL command-line utility.
- Use a third-party application (an application that issues the appropriate LDAP write).

To become the schema master programmatically, an application running in the context of a suitably privileged user can issue an LDAP write of the operational attribute **becomeSchemaMaster** to the rootDSE on that DC.

This initiates an atomic transfer of the schema master right from the current holder to the local DC.

The following code example finds the schema master. The following function binds to the schema container on the computer that is the schema master.

```
HRESULT BindToSchemaMaster(IADsContainer **ppSchemaMaster)
{
    HRESULT hr = E_FAIL;
    // Get rootDSE and the schema container DN.
    IADs *pObject = NULL;
    IADs *pTempSchema = NULL;
    IADs *pNTDS = NULL;
    IADs *pServer = NULL;
    BSTR bstrParent;
    LPOLESTR szPath = new OLECHAR[MAX_PATH];
    VARIANT var, varRole, varComputer;
    hr = ADsOpenObject(L"LDAP://rootDSE",
        NULL,
        NULL,
        ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
        IID_IADs,
        (void**)&pObject);
    if (hr == S_OK)
    {
        hr = pObject->Get(CComBSTR("schemaNamingContext"), &var);
        if (hr == S_OK)
        {
            wcscpy_s(szPath,L"LDAP://");
            wcscat_s(szPath,var.bstrVal);
            hr = ADsOpenObject(szPath,
                NULL,
                NULL,
                ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                IID_IADs,
                (void**)&pTempSchema);

            if (hr == S_OK)
            {
                /*
                Read the fsmoRoleOwner attribute to identify which server is
                the schema master.
                */
                hr = pTempSchema->Get(CComBSTR("fsmoRoleOwner"), &varRole);
                if (hr == S_OK)
                {
                    // The fsmoRoleOwner attribute returns the nTDSSDA object.
                    // The parent is the server object.
                    // Bind to NTDSDSA object and get parent.
                    wcscpy_s(szPath,L"LDAP://");
                    wcscat_s(szPath,varRole.bstrVal);
                    hr = ADsOpenObject(szPath,
                        NULL,
                        NULL,
                        ADS_SECURE_AUTHENTICATION,
                        IID_IADs,
                        (void**)&pNTDS);
                    if (hr == S_OK)
                    {
                        hr = pNTDS->get_Parent(&bstrParent);
                        if (hr == S_OK)
                        {
                            /*
                            Bind to server object and get the DNS name of
                            the server.
                            */
                            wcscpy_s(szPath,bstrParent);
                            hr = ADsOpenObject(szPath,
```

```

        NULL,
        NULL,
        ADS_SECURE_AUTHENTICATION,
        IID_IADS,
        (void**)&pServer);
    if (hr == S_OK)
    {
        // Get the dns name of the server.
        hr = pServer->Get(CComBSTR("dNSHostName"),
            &varComputer);
        if (hr == S_OK)
        {
            wcscpy_s(szPath,L"LDAP://");
            wcscat_s(szPath,varComputer.bstrVal);
            wcscat_s(szPath,L"/");
            wcscat_s(szPath,var.bstrVal);
            hr = ADsOpenObject(szPath,
                NULL,
                NULL,
                ADS_SECURE_AUTHENTICATION,
                IID_IADS,
                (void**)ppSchemaMaster);
            if (FAILED(hr))
            {
                if (*ppSchemaMaster)
                {
                    (*ppSchemaMaster)->Release();
                    (*ppSchemaMaster) = NULL;
                }
            }
            VariantClear(&varComputer);
        }
        if (pServer)
            pServer->Release();
    }
    SysFreeString(bstrParent);
}
if (pNTDS)
    pNTDS->Release();
}
VariantClear(&varRole);
}
if (pTempSchema)
    pTempSchema->Release();
}
VariantClear(&var);
}
if (pObject)
    pObject->Release();

return hr;
}

```

The following code example displays the DNS name for the computer that is the schema master.

```

On Error Resume Next

.....
' Bind to the rootDSE
.....
sPrefix = "LDAP://"
Set root= GetObject(sPrefix & "rootDSE")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If
.....
```

```

' Get the DN for the schema
.....
sSchema = root.Get("schemaNamingContext")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get method"
End If

.....
' Bind to the schema container
.....
Set Schema= GetObject(sPrefix & sSchema )
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method to bind to schema"
End If
.....
' Read the fsmoRoleOwner attribute to see which server is the
' schema master.
.....
sMaster = Schema.Get("fsmoRoleOwner")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::Get method for fsmoRoleOwner"
End If
.....
' The fsmoRoleOwner attribute returns the nTDSDSA object.
' The parent is the server object.
' Bind to NTDSDSA object and get the parent object.
.....
Set NTDS = GetObject(sPrefix & sMaster)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method for NTDS"
End If
sServer = NTDS.Parent
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::get_Parent method"
End If
.....
' Bind to server object
' and get the reference to the computer object.
.....
Set Server = GetObject(sServer)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method for " & sServer
End If
sComputer = Server.Get("dNSHostName")
.....
' Display the DNS name for the computer.
.....
strText = "Schema master has the following DNS name: "& sComputer
WScript.echo strText

.....
' Display subroutines
.....

```

```

Sub BailOnFailure(ErrNum, ErrText)
    strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub

```

Enabling Schema Changes at the Schema Master

6/3/2022 • 2 minutes to read • [Edit Online](#)

By default, schema modification is disabled on all Windows 2000 domain controllers. The ability to update the schema is controlled by the following registry value on the schema master domain controller:

```
HKEY_LOCAL_MACHINE
  System
    CurrentControlSet
      Services
        NTDS
          Parameters
            Schema Update Allowed
```

This registry value is a **REG_DWORD** value. If this value is not present or contains zero (0), then schema modification is disabled. If this value is present and contains a value other than zero, then schema modification is enabled.

The Schema Manager MMC snap-in provides the user with the ability to manually enable or disable schema modification. Schema modification can be enabled or disabled programmatically by modifying this registry value on the schema master domain controller.

The following C++ function shows how to determine if the schema can be modified on a specified schema master.

```
HRESULT IsSchemaUpdateEnabled(
    LPTSTR pszSchemaMasterComputerName,
    BOOL *pfEnabled)
{
    *pfEnabled = FALSE;

    LPTSTR szPrefix = "\\\\";
    LPTSTR pszPath = new TCHAR[1strlen(szPrefix) +
        1strlen(pszSchemaMasterComputerName) + 1];
    if(!pszPath)
    {
        return E_OUTOFMEMORY;
    }

    HRESULT hr = E_FAIL;
    LONG lReturn;
    HKEY hKeyMachine;

    tcscpy_s(pszPath, szPrefix);
    tcscat_s(pszPath, pszSchemaMasterComputerName);
    lReturn = RegConnectRegistry(
        pszPath,
        HKEY_LOCAL_MACHINE,
        &hKeyMachine);

    delete [] pszPath;

    if (ERROR_SUCCESS == lReturn)
    {
        HKEY hKeyParameters;
        LPTSTR szKeyPath =
            TEXT("System\\CurrentControlSet\\Services\\NTDS\\Parameters");
        LPTSTR szValueName = TEXT("Schema Update Allowed");
```

```

lReturn = RegOpenKeyEx(
    hKeyMachine,
    szKeyPath,
    0,
    KEY_READ,
    &hKeyParameters);
if (ERROR_SUCCESS == lReturn)
{
    DWORD dwType;
    DWORD dwValue;
    DWORD dwSize;

    dwSize = sizeof(dwValue);
    lReturn = RegQueryValueEx(
        hKeyParameters,
        szValueName,
        0,
        &dwType,
        (LPBYTE)&dwValue,
        &dwSize);
    if (ERROR_SUCCESS == lReturn)
    {
        *pfEnabled = (0 != dwValue);

        hr = S_OK;
    }
}

RegCloseKey(hKeyParameters);
}

RegCloseKey(hKeyMachine);
}

return hr;
}

```

The following C++ function shows how to enable or disable schema modification on a specified schema master.

```

HRESULT EnableSchemaUpdate(
    LPTSTR pszSchemaMasterComputerName,
    BOOL fEnabled)
{
    LPTSTR szPrefix = "\\\\";
    LPTSTR pszPath = new TCHAR[lstrlen(szPrefix) +
        lstrlen(pszSchemaMasterComputerName) + 1];
    if(!pszPath)
    {
        return E_OUTOFMEMORY;
    }

    HRESULT hr = E_FAIL;
    LONG lReturn;
    HKEY hKeyMachine;

    strcpy_s(pszPath, szPrefix);
    strcat_s(pszPath, pszSchemaMasterComputerName);
    lReturn = RegConnectRegistry(
        pszPath,
        HKEY_LOCAL_MACHINE,
        &hKeyMachine);

    delete [] pszPath;

    if (ERROR_SUCCESS == lReturn)
    {

```

```
HKEY hKeyParameters;
LPTSTR szRelKeyPath =
    TEXT("System\CurrentControlSet\Services\NTDS\Parameters");
LPTSTR szValueName = TEXT("Schema Update Allowed");

lReturn = RegOpenKeyEx(
    hKeyMachine,
    szRelKeyPath,
    0,
    KEY_SET_VALUE,
    &hKeyParameters);
if (ERROR_SUCCESS == lReturn)
{
    DWORD dwValue;
    DWORD dwSize;

    if(fEnabled)
    {
        dwValue = 1;
    }
    else
    {
        dwValue = 0;
    }

    dwSize = sizeof(dwValue);
    lReturn = RegSetValueEx(
        hKeyParameters,
        szValueName,
        0L,
        REG_DWORD,
        (LPBYTE)&dwValue,
        dwSize);
    if (ERROR_SUCCESS == lReturn)
    {
        hr = S_OK;
    }

    RegCloseKey(hKeyParameters);
}

RegCloseKey(hKeyMachine);
}

return hr;
}
```

Recommendations for Schema Extension Applications

6/3/2022 • 2 minutes to read • [Edit Online](#)

In addition to the prerequisites, the following best practices are recommended for schema extension applications:

- Find the schema master. Bind to the schema on the DC that is the schema master. Avoid unnecessarily changing the schema master role between DCs. To bind to the schema container on the schema master. For more information, see [Prerequisites for Installing a Schema Extension](#).
- Before performing any action, check the **allowedChildClassesEffective** property of the schema container to verify that you can create attributes and/or classes. If **attributeSchema** and **classSchema** are not values in that property, you do not have sufficient rights to add attributes or classes to the schema. For more information, see [Example Code for Checking for Rights to Create Schema Objects](#).
- Ensure that the Schema Update Allowed is set appropriately in the registry of the schema master. To create or set this value, restore it to its original state as part of your application clean-up routine. For more information about checking and setting this value, see [Enabling Schema Changes at the Schema Master](#).
- Before adding attributes or classes, be sure that they do not already exist. If they do exist, verify that they are the same attributes or classes you are adding and not an attribute or class created by someone with different syntax and properties that are incompatible with your attributes or classes.

For attributes, query for **cn**, **attributeID**, **governsID**, **IDAPDisplayName**, and **schemaIDGUID** to ensure they are not already used. If you add a set of linked attributes (one forward link, one back link), ensure that the **linkIDs** are not already used. Query for **governsID** because the object identifier (OID) must be unique among attributes and classes.

For classes, query for **cn**, **governsID**, **attributeID**, **IDAPDisplayName**, and **schemaIDGUID** to ensure they are not already used. Query for **attributeID** because the OID must be unique among classes and attributes.

For more information about checking for naming collisions, see [Example Code for Detecting Schema Naming Collisions](#).

If attributes or classes exist that conflict with your new attributes or classes, your application should not apply your schema changes.

- If such a collision exists, your application should not apply your schema changes. The schema administrator may need to resolve the collision and then run your application again. Alternatively, a different **IDAPDisplayName** could be used; however, any applications using the attribute or object must be aware of that change. To help avoid OID collisions, obtain an OID from an ISO name registration authority.
- If your application is dependent on attributes or classes that it has added, update the schema cache before adding the new attributes or classes that are dependent on those attributes or classes. Be aware that the **schemaUpdateNow** operational attribute is synchronous. That is, the **IADs::Put** method call will block until the schema cache is updated. When the call returns, the schema cache has been updated and the new attributes and/or classes are accessible.

For more information about how to update the schema cache, see [Example Code for Updating the Schema Cache](#).

Example Code for Checking for Rights to Create Schema Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C/C++ code example shows a function that checks the **allowedChildClassesEffective** attribute on the schema container (IADs pointer to schema container is passed as a parameter) for the **attributeSchema** and **classSchema** classes. It returns **S_OK** if both classes are listed in **allowedChildClassesEffective**. If both are not, it returns **S_FALSE**.

```
// Function takes an IADs pointer to schema container as a parameter.

HRESULT HasSchemaAdditionRights(IADs *pSchema)
{
    if (!pSchema)
        return E_POINTER;
    HRESULT hr = E_FAIL;
    VARIANT var;
    VARIANT *pVar;
    // Check access rights.
    BOOL bIsAddAttributeAllowed = FALSE;
    BOOL bIsAddClassAllowed = FALSE;
    // Get AllowedChildClassesEffective. It is constructed;
    // therefore, you must use GetInfoEx to explicitly
    // get it into the cache.
    LPOLESTR pwszArray[] = {L"allowedChildClassesEffective"};
    DWORD dwArrayItems = sizeof(pwszArray)/sizeof(LPOLESTR);
    VARIANT vArray;
    VariantInit(&vArray);
    // Build a Variant of array type, using the specified string array.
    hr = ADsBuildVarArrayStr(pwszArray, dwArrayItems, &vArray);
    if (SUCCEEDED(hr))
    {
        hr = pSchema->GetInfoEx(vArray,0L);
        if (SUCCEEDED(hr))
        {
            hr = pSchema->GetEx(CComBSTR("allowedChildClassesEffective"),
                                  &var);
            if (SUCCEEDED(hr))
            {
                hr = SafeArrayAccessData((SAFEARRAY*)(var.parray),
                                         (void HUGE* FAR*)&pVar);
                long lLBound, lUBound;
                // One-dimensional array. Get the bounds for the array.
                hr = SafeArrayGetLBound((SAFEARRAY*)(var.parray),
                                       1,
                                       &lLBound);
                hr = SafeArrayGetUBound((SAFEARRAY*)(var.parray),
                                       1,
                                       &lUBound);
                // Get the count of elements
                long cElements = lUBound-lLBound + 1;
                // Get the elements of the array.
                if (SUCCEEDED(hr))
                {
                    for (int i = 0; i < cElements; i++ )
                    {
                        // Check each element to verify that attributeSchema
                        // or classSchema is there.
                        if (VT_BSTR == pVar[i].vt)
```

```
    {
        if (0==_wcsicmp(L"attributeSchema", pVar[i].bstrVal))
            bIsAddAttributeAllowed = TRUE;
        else if (0==_wcsicmp(L"classSchema", pVar[i].bstrVal))
            bIsAddClassAllowed = TRUE;
    }
}

if (bIsAddAttributeAllowed && bIsAddClassAllowed)
    hr = S_OK;
else
    hr = S_FALSE;
}
VariantClear(&var);
}

return hr;
};


```

Example Code for Updating the Schema Cache

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C/C++ functions get the DNS name of the schema master (GetSchemaMasterDNSName) and update the schema cache at the schema master (UpdateSchemaMasterCache). To use these functions, call GetSchemaMasterDNSName to get the schema master DNS name, pass the schema master DNS name to UpdateSchemaMasterCache, call CoTaskMemFree on string containing the schema master DNS name.

For more information, see [Updating the Schema Cache](#).

```
HRESULT UpdateSchemaMasterCache(LPOLESTR szSchemaMasterDNSName)
{
    if (!szSchemaMasterDNSName)
        return E_POINTER;
    HRESULT hr = E_FAIL;
    IADS *pObject = NULL;
    VARIANT var;
    LPOLESTR szPath = new OLECHAR[MAX_PATH];
    wcscpy_s(szPath,L"LDAP://");
    wcscat_s(szPath,szSchemaMasterDNSName);
    wcscat_s(szPath,L"/rootDSE");
    hr = ADsOpenObject(szPath,
                       NULL,
                       NULL,
                       ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                       IID_IADS,
                       (void**)&pObject);
    if (SUCCEEDED(hr))
    {
        VariantInit(&var);
        var.vt = VT_I4;
        var.lVal = 1L;
        hr = pObject->Put(CComBSTR("schemaUpdateNow"), var);
        if (SUCCEEDED(hr))
        {
            hr = pObject->SetInfo();
            if (SUCCEEDED(hr))
                MessageBox(NULL,L"Updated",L"Extend Schema Wizard",MB_OK|MB_ICONEXCLAMATION);
            else
                MessageBox(NULL,L"Update Failed",L"Extend Schema Wizard",MB_OK|MB_ICONEXCLAMATION);
        }
    }
    if (pObject)
        pObject->Release();
}

return hr;
}

HRESULT GetSchemaMasterDNSName(LPOLESTR *pszSchemaMasterDNSName)
{
    if (!pszSchemaMasterDNSName)
        return E_POINTER;
    HRESULT hr = E_FAIL;
    IADS *pObject = NULL;
    IADS *pTempSchema = NULL;
    IADS *pNTDS = NULL;
    IADS *pServer = NULL;
    BSTR bstrParent;
    LPOLESTR szPath = new OLECHAR[MAX_PATH];
    VARIANT var, varRole,varComputer;
    hr = ADsOpenObject(L"LDAP://rootDSE",
```

```

        NULL,
        NULL,
        ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
        IID_IADS,
        (void**)&pObject);
if (SUCCEEDED(hr))
{
    hr = pObject->Get(CComBSTR("schemaNamingContext"), &var);
    if (SUCCEEDED(hr))
    {
        wcscpy_s(szPath,L"LDAP://");
        wcscat_s(szPath,var.bstrVal);
        hr = ADsOpenObject(szPath,
                           NULL,
                           NULL,
                           ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                           IID_IADS,
                           (void**)&pTempSchema);

        if (SUCCEEDED(hr))
        {
            // Read the fsmoRoleOwner attribute to see which server is the schema master.
            hr = pTempSchema->Get(CComBSTR("fsmoRoleOwner"), &varRole);
            if (SUCCEEDED(hr))
            {
                // fsmoRoleOwner attribute returns the nTDSDSA object.
                // The parent is the server object.
                // Bind to NTDSDSA object and get the parent object.
                wcscpy_s(szPath,L"LDAP://");
                wcscat_s(szPath,varRole.bstrVal);
                hr = ADsOpenObject(szPath,
                                   NULL,
                                   NULL,
                                   ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                                   IID_IADS,
                                   (void**)&pNTDS);
                if (SUCCEEDED(hr))
                {
                    hr = pNTDS->get_Parent(&bstrParent);
                    if (SUCCEEDED(hr))
                    {
                        // Bind to server object
                        // and get the reference to the computer object.
                        wcscpy_s(szPath,bstrParent);
                        hr = ADsOpenObject(szPath,
                                           NULL,
                                           NULL,
                                           ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                                           IID_IADS,
                                           (void**)&pServer);
                        if (SUCCEEDED(hr))
                        {
                            // Get the dns name of the server.
                            hr = pServer->Get(CComBSTR("dNSHostName"), &varComputer);
                            if (SUCCEEDED(hr))
                            {
                                *pszSchemaMasterDNSName = (OLECHAR *)CoTaskMemAlloc (sizeof(OLECHAR)*
(wcslen(varComputer.bstrVal)+1));
                                if (*pszSchemaMasterDNSName)
                                {
                                    wcscpy_s(*pszSchemaMasterDNSName, varComputer.bstrVal);
                                }
                                else
                                {
                                    hr = E_OUTOFMEMORY;
                                }
                            }
                            VariantClear(&varComputer);
                        }
                    }
                }
            }
        }
    }
}

```

```
    if (pServer)
        pServer->Release();
    }
    SysFreeString(bstrParent);
}
if (pNTDS)
    pNTDS->Release();
}
VariantClear(&varRole);
}
if (pTempSchema)
    pTempSchema->Release();
}
VariantClear(&var);
}

if (pObject)
    pObject->Release();

return hr;
}
```

Example Code for Detecting Schema Naming Collisions

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes a code example that detects schema naming collisions.

The following C/C++ code example queries the schema for the key naming attributes on a **classSchema** or **attributeSchema** object.

It returns **TRUE** if conflicting attributes or classes are found. It returns **FALSE** if the attribute or class with the specified **cn**, **LDAPDisplayName**, **OID**, **schemaIDGUID**, or **linkID** does not conflict with the schema and, therefore, is safe to add to the schema.

```
/*****************************************************************************  
  
FindCollidingAttributesOrClasses()  
  
*****  
  
HRESULT FindCollidingAttributesOrClasses(  
    IDirectorySearch *pSchemaNC, // IDirectorySearch pointer to  
                                // schema naming context.  
    LPWSTR szName,  
    LPWSTR szLDAPDisplayName,  
    LPWSTR szOID,  
    const GUID * pSchemaIDGUID,  
    DWORD dwLinkID, // Value is zero (0) if the attribute is not linked,  
                    // or if checking for class.  
    BOOL *pfIsColliding)  
{  
    if (!pSchemaNC ||  
        !szName ||  
        !szLDAPDisplayName ||  
        !szOID ||  
        !pSchemaIDGUID)  
    {  
        return E_POINTER;  
    }  
  
    HRESULT hr = E_FAIL;  
  
    LPWSTR szBuffer = NULL;  
  
    hr = ADsEncodeBinaryData((LPBYTE)pSchemaIDGUID,  
                            sizeof(GUID),  
                            &szBuffer);  
  
    if (SUCCEEDED(hr))  
    {  
        LPWSTR pwszFormatLinkID = L"(|(cn=%s)(LDAPDisplayName=%s)(attributeID=%s)(governnsID=%s)  
(schemaIDGUID=%s)(linkID=%d))";  
        LPWSTR pwszFormatNoLinkID = L"(|(cn=%s)(LDAPDisplayName=%s)(attributeID=%s)(governnsID=%s)  
(schemaIDGUID=%s))";  
        LPWSTR szFilter = new WCHAR[lstrlenW(pwszFormatLinkID) +  
                                lstrlenW(szName) +  
                                lstrlenW(szLDAPDisplayName) +  
                                lstrlenW(szOID) +  
                                lstrlenW(szOID) +  
                                lstrlenW(szBuffer) +  
                                20 +  
                                1];
```

```

if(szFilter)
{
    if (dwLinkID > 0L)
    {
        swprintf_s(szFilter,
                   pwszFormatLinkID,
                   szName,
                   szLDAPDisplayName,
                   szOID,
                   szOID,
                   szBuffer,
                   dwLinkID);
    }
    else
    {
        swprintf_s(szFilter,
                   pwszFormatNoLinkID,
                   szName,
                   szLDAPDisplayName,
                   szOID,
                   szOID,
                   szBuffer);
    }

    // Attributes are one-level deep in the Schema
    // container so only search one level.
    ADS_SEARCHPREF_INFO SearchPrefs[2];
    SearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    SearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
    SearchPrefs[0].vValue.Integer = ADS_SCOPE_ONELEVEL;

    SearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;
    SearchPrefs[1].vValue.dwType = ADSTYPE_INTEGER;
    SearchPrefs[1].vValue.Integer = 1000;

    int iCount = 0;

    // Set the search preference
    hr = pSchemaNC->SetSearchPreference(
        SearchPrefs,
        sizeof(SearchPrefs)/sizeof(ADS_SEARCHPREF_INFO)
    );
    if (SUCCEEDED(hr))
    {
        LPWSTR pszAttribs[] = {
            L"LDAPDisplayName",
            L"cn",
            L"attributeID",
            L"governsID",
            L"schemaIDGUID",
            L"linkID",
            L"objectCategory"
        };
    }

    // Handle used for searching
    ADS_SEARCH_HANDLE hSearch = NULL;
    DWORD x = 0L;

    hr = pSchemaNC->ExecuteSearch(szFilter,
                                    pszAttribs,
                                    sizeof(pszAttribs)/sizeof(LPWSTR),
                                    &hSearch);
    if (SUCCEEDED(hr))
    {
        // Call IDirectorySearch::GetNextRow()
        // to retrieve the next row of data.
        hr = pSchemaNC->GetFirstRow(hSearch);
        while (hr != S_NO_MORE_ROWS)

```

```
while (m_iCount < iAddNonCollRows)
{
    // Keep track of count.
    iCount++;

    // Get the next row
    hr = pSchemaNC->GetNextRow(hSearch);
}

// Close the search handle to cleanup
pSchemaNC->CloseSearchHandle(hSearch);

if(SUCCEEDED(hr))
{
    if (0 == iCount)
    {
        *pfIsColliding = FALSE;
    }
    else
    {
        *pfIsColliding = TRUE;
    }

    hr = S_OK;
}
}

delete szFilter;
}

FreeADsMem(szBuffer);
}

return hr;
}
```

Supported Installation Mechanisms

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services support four mechanisms for extending the Active Directory schema. You can extend the Active Directory schema by using

- [LDIF Scripts](#) (preferred)
- [Comma-separated Value \(CSV\) Scripts](#)
- [Programmatic Extension](#)
- and by [Extending the User Interface for Directory Objects](#).

LDIF Scripts

6/3/2022 • 9 minutes to read • [Edit Online](#)

The LDAP Data Interchange Format (LDIF) is an Internet Engineering Task Force (IETF) standard that defines how to import and export directory data between directory servers that use LDAP service providers. Windows 2000 and Windows Server 2003 include a command-line utility, LDIFDE, which can be used to import directory objects into Active Directory Domain Services using LDIF files. LDIFDE enables you to set a filter to a specific string in order to search for and list directory objects in Active Directory Domain Services as LDIF files which can be easily read by schema administrators.

When importing a Unicode file, LDIFDE imports the file as Unicode if it contains the Unicode identifier at the beginning of the file. If you wish to import a file as Unicode when it does not contain the Unicode identifier at the beginning of the file, you can use the -u switch in order to force it to be imported as Unicode.

The default mode for exporting files is ANSI. If there are Unicode entries, they will be converted into base 64 format. To export a file into Unicode format, use the -u switch.

An LDIF file must apply schema changes when there are dependencies between the attributes that are added. For example, forward link attributes should be added before the corresponding back link attribute. You must also update the schema cache before adding classes that depend on attributes or classes added earlier in the LDIF script. For more information, see the following code example.

Be aware that for binary values, you must encode the values as base64. Base64 encoding is defined in IETF RFC 2045, Section 6.8.

For more information about the format of LDIF files, see [The LDAP Data Interchange Format \(LDIF\) - Technical Specification](#) (RFC 2849) on the Internet Engineering Task Force website.

NTDS-specific LDIF changetypes

It is better to use ntdsSchema* changetypes rather than calling ldifde -k. The -k option of ldifde ignores a larger set of LDAP errors. The complete list of ignored errors is as follows:

- The object is already a member of the group.
- An object class violation (meaning the specified object class does not exist), if the object being imported has no other attributes.
- object already exists (LDAP_ALREADY_EXISTS)
- constraint violation (LDAP_CONSTRAINT_VIOLATION)
- attribute or value already exists (LDAP_ATTRIBUTE_OR_VALUE_EXISTS)
- no such object (LDAP_NO_SUCH_OBJECT)

The following changetypes are designed specifically for schema upgrade operations.

CHANGETYPE	DESCRIPTION
ntdsSchemaAdd	ntdsSchemaAdd corresponds to add in an LDIF file. The only difference is that ntdsSchemaAdd would cause ldifde to skip an add operation if the object already exists in the schema. (LDAP_ALREADY_EXISTS is ignored.)

CHANGETYPE	DESCRIPTION
ntdsSchemaModify	ntdsSchemaModify corresponds to modify in an LDIF file. The only difference is that ntdsSchemaModify would cause ldifde to skip an modify operation if the object is not found in the schema. (LDAP_NO SUCH OBJECT is ignored.)
ntdsSchemaDelete	ntdsSchemaDelete corresponds to delete in an LDIF file. The only difference is that ntdsSchemaDelete would cause ldifde to skip an delete operation if the object is not found in the schema. (LDAP_NO SUCH OBJECT is ignored.)
ntdsSchemaModRdn	ntdsSchemaModRdn corresponds to modrdn in an LDIF file. The only difference is that ntdsSchemaModRdn would cause ldifde to skip a modify-relative-distinguished-name operation if the object is not found in the schema. (LDAP_NO SUCH OBJECT is ignored.)

Example

The following code example includes:

- Myschemaext.ldf is an LDIF script that contains new attributes and classes. Be aware that this file is a modified version of the file generated from Lgetattcls.vbs. Also be aware that the **My-Test-Attribute-DN-FL** attribute was moved ahead of **My-Test-Attribute-DN-BL** because the back link (**My-Test-Attribute-DN-BL**) is dependent on the forward link (**My-Test-Attribute-DN-FL**). Furthermore, the **schemaUpdateNow** operational attribute is set in two places to trigger updates of the schema cache so that dependent attributes and classes will be available for adding the two classes in the script.

NOTE

See the topic [Obtaining a Link ID](#) for information about the source of the ID in the linkID: statements.

- Lgetattcls.vbs is a VBScript file that generates the LDIF script used as the starting point for the Myschemaext.ldf. Be aware that the current schema path is replaced by CN=Schema,CN=Configuration,DC=myorg,DC=com. You can replace DC=myorg,DC=com to reflect the distinguished name (DN) to publish in the LDIF script ensure that LSETATTCLS.VBS reflects the change in its **sFromDN** so that the correct DN is replaced when the LDIF script is applied. Also be aware that the script uses a prefix to find the classes and attributes you should also define and use a prefix for all your classes and attributes. For more information, see [Naming Attributes and Classes](#). In addition, the script outputs only the necessary attributes for the **attributeSchema** and **classSchema** objects to the LDIF file.
- Lsetattcls.vbs is a VBScript file that uses the Myschemaext.ldf script to add the new attributes and classes in the script. Ensure that the schema master is able to be written to before running the script.

MYSCHEMAEXT.LDF

```

dn: CN=My-Test-Attribute-CaseExactString,CN=Schema,CN=Configuration,DC=myorg,DC=com
changetype: add
adminDisplayName: My-Test-Attribute-CaseExactString
attributeID: 1.2.840.113556.1.4.7000.159.24.10.65
attributeSyntax: 2.5.5.3
cn: My-Test-Attribute-CaseExactString
description: Test attribute of syntax CaseExactString used to show how to add a CaseExactString attribute.

```

```
isMemberOfPartialAttributeSet: FALSE
isSingleValued: TRUE
LDAPDisplayName: myTestAttributeCaseExactString
distinguishedName: CN=My-Test-Attribute-CaseExactString,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectCategory: CN=Attribute-Schema,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectClass: attributeSchema
oMSyntax: 27
name: My-Test-Attribute-CaseExactString
schemaIDGUID:: 6ASznA3W0hGBpwDAT7mMGg==
searchFlags: 0

dn: CN=My-Test-Attribute-DN-FL,CN=Schema,CN=Configuration,DC=myorg,DC=com
changetype: add
adminDisplayName: My-Test-Attribute-DN-FL
attributeID: 1.2.840.113556.1.4.7000.159.24.10.614
attributeSyntax: 2.5.5.1
cn: My-Test-Attribute-DN-FL
description: Test forward link attribute of syntax DN used to show how to add a forward link attribute. Back link is My-Test-Attribute-DN-BL.
isMemberOfPartialAttributeSet: FALSE
isSingleValued: TRUE
LDAPDisplayName: myTestAttributeDNFL
linkID: 1.2.840.113556.1.2.50
distinguishedName: CN=My-Test-Attribute-DN-FL,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectCategory: CN=Attribute-Schema,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectClass: attributeSchema
oMObjectClass:: KwwCh3McAIVK
oMSyntax: 127
rangeLower: 0
rangeUpper: 257
name: My-Test-Attribute-DN-FL
schemaIDGUID:: YGLudffa0hGLEwDAT7mMGg==
searchFlags: 0

dn: CN=My-Test-Attribute-DN-BL,CN=Schema,CN=Configuration,DC=myorg,DC=com
changetype: add
adminDisplayName: My-Test-Attribute-DN-BL
attributeID: 1.2.840.113556.1.4.7000.159.24.10.615
attributeSyntax: 2.5.5.1
cn: My-Test-Attribute-DN-BL
description: Test back link attribute of syntax DN used to show how to add a back link attribute. Forward link is My-Test-Attribute-DN-FL.
isMemberOfPartialAttributeSet: FALSE
isSingleValued: TRUE
LDAPDisplayName: myTestAttributeDNBL
linkID: 1.2.840.113556.6.1234
distinguishedName: CN=My-Test-Attribute-DN-BL,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectCategory: CN=Attribute-Schema,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectClass: attributeSchema
oMObjectClass:: KwwCh3McAIVK
oMSyntax: 127
rangeLower: 0
rangeUpper: 257
name: My-Test-Attribute-DN-BL
schemaIDGUID:: jFfbhffa0hGLEwDAT7mMGg==
searchFlags: 0

dn: CN=My-Test-Attribute-DN-Regular,CN=Schema,CN=Configuration,DC=myorg,DC=com
changetype: add
adminDisplayName: My-Test-Attribute-DN-Regular
attributeID: 1.2.840.113556.1.4.7000.159.24.10.613
attributeSyntax: 2.5.5.12
cn: My-Test-Attribute-DN-Regular
description: Test attribute of syntax DN used to show how to add a DN attribute.
isMemberOfPartialAttributeSet: FALSE
isSingleValued: TRUE
LDAPDisplayName: myTestAttributeDNRegular
distinguishedName: CN=My-Test-Attribute-DN-Regular,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectCategory: CN=Attribute-Schema,CN=Schema,CN=Configuration,DC=myorg,DC=com
```

```
objectClass: attributeSchema
oMObjectClass:: KwwCh3McAIVK
oMSyntax: 64
rangeLower: 0
rangeUpper: 257
name: My-Test-Attribute-DN-Regular
schemaIDGUID:: 5QSznA3W0hGBpwDAT7mMGg==
searchFlags: 0

dn: CN=My-Test-Attribute-DNString,CN=Schema,CN=Configuration,DC=myorg,DC=com
changetype: add
adminDisplayName: My-Test-Attribute-DNString
attributeID: 1.2.840.113556.1.4.7000.159.24.10.611
attributeSyntax: 2.5.5.14
cn: My-Test-Attribute-DNString
description: Test attribute of syntax DNString used to show how to add a DNString attribute.
isMemberOfPartialAttributeSet: FALSE
isSingleValued: TRUE
1LDAPDisplayName: myTestAttributeDNString
distinguishedName: CN=My-Test-Attribute-DNString,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectCategory: CN=Attribute-Schema,CN=Schema,CN=Configuration,DC=myorg,DC=com
objectClass: attributeSchema
oMObjectClass:: KoZIHvcUAQEBCDA==
oMSyntax: 127
rangeLower: 1
rangeUpper: 64
name: My-Test-Attribute-DNString
schemaIDGUID:: 5ASznA3W0hGBpwDAT7mMGg==
searchFlags: 0

DN:
changetype: modify
add: schemaUpdateNow
schemaUpdateNow: 1
-
dn: CN=My-Test-Auxiliary-Class1,CN=Schema,CN=Configuration,DC=Fabrikam,DC=com
changetype: add
adminDisplayName: My-Test-Auxiliary-Class1
description: Test class used to show how to add an auxiliary class.
objectCategory: CN=Class-Schema,CN=Schema,CN=Configuration,DC=Fabrikam,DC=com
objectClass: classSchema
1LDAPDisplayName: myTestAuxiliaryClass1
governsID: 1.2.840.113556.1.4.7000.159.24.10.611.11
instanceType: 4
objectClassCategory: 3
schemaIDGUID:: mmsxdsXb0hGL0AAA+HW2YA==
subClassOf: Top
mayContain: my-Test-Attribute-DNString
mustContain: description

DN:
changetype: modify
add: schemaUpdateNow
schemaUpdateNow: 1
-
dn: CN=My-Test-Structural-Class1,CN=Schema,CN=Configuration,DC=Fabrikam,DC=com
changetype: add
adminDisplayName: My-Test-Structural-Class1
auxiliaryClass: myTestAuxiliaryClass1
defaultHidingValue: FALSE
defaultObjectCategory: CN=Organizational-Unit,CN=Schema,CN=Configuration,DC=Fabrikam,DC=com
defaultSecurityDescriptor: D:(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;DA)(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;SY)(A;;RPLCLORC;;;AU)
adminDescription: Test class used to show how to add a structure class.
objectCategory: CN=Class-Schema,CN=Schema,CN=Configuration,DC=Fabrikam,DC=com
objectClass: classSchema
1LDAPDisplayName: myTestStructuralClass1
```

```

IDAPNtspPrimaryName: myTestAttributeDN
governedID: 1.2.840.113556.1.4.7000.159.24.10.611.12
mayContain: myTestAttributeDNFL
mayContain: wWWHomePage
mustContain: url
instanceType: 4
objectClassCategory: 1
possSuperiors: organizationalUnit
rDNAttID: ou
schemaIDGUID:: 1HsnsL7b0hGL0AAA+HW2YA==
subClassOf: organizationalUnit

DN:
changetype: modify
add: schemaUpdateNow
schemaUpdateNow: 1
-

```

LGETATTCLS.VBS

```

On Error Resume Next

.....
' Bind to the rootDSE
.....
sPrefix = "LDAP://"
Set root= GetObject(sPrefix & "rootDSE")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If

.....
' Get the DN for the Schema
.....
sSchema = root.Get("schemaNamingContext")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get method"
End If

.....
' Bind to the Schema container
.....
Set Schema= GetObject(sPrefix & sSchema )
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method to bind to Schema"
End If

.....
' Read the fsmoRoleOwner attribute to see which server is the schema master.
.....
sMaster = Schema.Get("fsmoRoleOwner")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::Get method for fsmoRoleOwner"
End If

.....
' fsmoRoleOwner attribute returns the nTDSDSA object.
' The parent is the server object.
' Bind to NTDSDSA object and get parent
.....
Set NTDS = GetObject(sPrefix & sMaster)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method for NTDS"
End If
sServer = NTDS.Parent
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::get_Parent method"
End If
.....

```

```

' Bind to server object and get the
' reference to the computer object.
.....
Set Server = GetObject(sServer)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method for " & sServer
End If
.....
' Display the DN for the computer object.
.....
sComputerDNSName = Server.Get("DNSHostName")
strText = "Schema Master has the following DNS Name: "& sComputerDNSName
WScript.echo strText

sFile = "myschemaext1.ldf"
sFromDN = sSchema
sToDN = "CN=Schema,CN=Configuration,DC=myorg,DC=com"
sAttrPrefix = "My-Test"
sFilter = "((&(cn=" & sAttrPrefix & "*)(|(objectCategory=classSchema)_
(objectCategory=attributeSchema))))"
sRetAttr = "dn,adminDescription,adminDisplayName,governsID,cn,mayContain,_
mustContain,systemMayContain,systemMustContain,LDAPDisplayName,_
objectClassCategory,distinguishedName,objectCategory,objectClass,_
possSuperiors,systemPossSuperiors,subClassOf,defaultObjectCategory,_
name,schemaIDGUID,auxiliaryClass,auxiliaryClass,systemAuxiliaryClass,_
description,defaultHidingValue,rDNAttId,defaultSecurityDescriptor,_
attributeID,attributeSecurityGUID,attributeSyntax,_
isMemberOfPartialAttributeSet,isSingleValued,mAPIID,oMSyntax,rangeLower,_
rangeUpper,searchFlags,oMObjectClass,linkID"

' Add flag rootDN.
sCommand = "ldifde -d " & sSchema
sCommand = sCommand & " -c " & sFromDN & " " & sToDN
' Add flag schema master.
sCommand = sCommand & " -s " & sComputerDNSName
' Add flag filename.
sCommand = sCommand & " -f " & sFile
' Add flag filter to search for attributes.
sCommand = sCommand & " -r " & sFilter
' Add flag for attributes to return.
sCommand = sCommand & " -l " & sRetAttr

WScript.echo sCommand
Set WshShell = Wscript.CreateObject("Wscript.Shell")
WshShell.Run (sCommand)

.....
' Display subroutines
.....
```

Sub BailOnFailure(ErrNum, ErrText) strText = "Error 0x"_
& Hex(ErrNum) & " " & ErrText
 MsgBox strText, vbInformation, "ADSI Error"
 WScript.Quit
End Sub

LSETATTCLS.VBS

```

On Error Resume Next
.....
' Bind to the rootDSE
.....
sPrefix = "LDAP://"
Set root= GetObject(sPrefix & "rootDSE")
```

```

Set root = GetObject("LDAP://<yourDC>")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If

.....
' Get the DN for the Schema
.....
sSchema = root.Get("schemaNamingContext")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get method"
End If

.....
' Bind to the Schema container
.....
Set Schema= GetObject(sPrefix & sSchema )
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method to bind to Schema"
End If

.....
' Read the fsmoRoleOwner attribute to see which server is the schema master.
.....
sMaster = Schema.Get("fsmoRoleOwner")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::Get method for fsmoRoleOwner"
End If

.....
' fsmoRoleOwner attribute returns the nTDSDSA object.
' The parent is the server object.
' Bind to NTDSDSA object and get parent
.....
Set NTDS = GetObject(sPrefix & sMaster)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method for NTDS"
End If
sServer = NTDS.Parent
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::get_Parent method"
End If

.....
' Bind to server object
' and get the reference to the computer object.
.....
Set Server = GetObject(sServer)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method for " & sServer
End If
sComputer = Server.Get("serverReference")

.....
' Display the DN for the computer object.
.....
sComputerDNSName = Server.Get("DNSHostName")
' strText = "Schema Master has the following DN: "& sComputer
strText = "Schema Master has the following DNS Name: "& sComputerDNSName
WScript.echo strText

sFile = "myschemaext.ldf"
sFromDN = "CN=Schema,CN=Configuration,DC=myorg,DC=com"
sToDN = sSchema
' Add flag replace fromDN with ToDN.
sCommand = "ldifde -i -k -c " & sFromDN & " " & sToDN
' Add flag schema master.
sCommand = sCommand & " -s " & sComputerDNSName
'Add flag filename.
sCommand = sCommand & " -f " & sFile
' Add flag filter to search for my attributes.

WScript.echo sCommand
Set WshShell = WScript.CreateObject("WScript.Shell")

```

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run (sCommand)

.....
' Display subroutines
.....

Sub BailOnFailure(ErrNum, ErrText)    strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub
```

Comma-separated Value (CSV) Scripts

6/3/2022 • 2 minutes to read • [Edit Online](#)

Windows 2000 includes a command-line utility, CSVDE, to import directory objects using .csv files and export directory objects as .csv files. CSV scripts are created for ease-of-use. The first line in the script identifies the attributes in the lines that follow. Columns are separated by commas. The file format is compatible with the Microsoft Excel .csv format, so that files are easily created. Use Excel or any other tool that can read and write .csv files. CSVDE supports Unicode.

Example CSV File

The following code example is a CSV file that adds an auxiliary class.

```
DN,adminDisplayName,cn,defaultHidingValue,defaultObjectCategory,description,governsID,lDAPDisplayName,mayContain,mustContain,distinguishedName,objectCategory,objectClass,objectClassCategory,possSuperiors,name,rDNAttID,schemaIDGUID,subClassOf,auxiliaryClass,defaultSecurityDescriptor
"CN=My-Test-Auxiliary-Class1,CN=Schema,CN=Configuration,DC=myorg,DC=mytest,DC=Fabrikam,DC=com",My-Test-Auxiliary-Class1,My-Test-Auxiliary-Class1,TRUE,"CN=My-Test-Auxiliary-Class1,CN=Schema,CN=Configuration,DC=myorg,DC=mytest,DC=Fabrikam,DC=com",Test class used to show how to add an auxiliary
class.,1.2.840.113556.1.4.7000.159.24.10.611.11,myTestAuxiliaryClass1,myTestAttributeDNString,description;myTestAttributeUnicodeString,"CN=My-Test-Auxiliary-Class1,CN=Schema,CN=Configuration,DC=myorg,DC=mytest,DC=Fabrikam,DC=com", "CN=Class-Schema,CN=Schema,CN=Configuration,DC=myorg,DC=mytest,DC=Fabrikam,DC=com",classSchema,3,container,My-Test-Auxiliary-Class1,cn,X'9a6b3176c5dbd2118bd00000f875b660',top,,
```

Programmatic Extension

6/3/2022 • 2 minutes to read • [Edit Online](#)

The benefit of an LDIF file is that the administrator can examine its function. However, the schema can also be extended programmatically. Programmatic extensions have several merits, but an application is opaque; the administrator must trust the documentation included with the setup application. Use of programmatic schema extensions may be a barrier to deployment for applications that use them. A programmatic extension is invariant; it is a Windows NT executable. The binary cannot be tampered with, unlike an LDIF or .csv file, which can be edited inadvertently or maliciously.

Benefits of an LDIF file include:

- Applications can detect and recover from errors and provide user feedback.
- Applications can handle Unicode without resorting to Base64 encoding.
- Applications can leverage Windows Installer setup APIs.
- Applications can be signed to establish authenticity.

Example Code for Extending the Schema Programmatically

6/3/2022 • 13 minutes to read • [Edit Online](#)

This topic includes a C++ code example that programmatically extends the Active Directory Schema.

```
/*
** schema.hxx: The following C++ code example installs an Active
**                 Directory Schema extension.
**
** This example creates six new attributes, one new class, and
** modifies two existing classes by adding new MAY HAVE attributes.
**
** Attributes:    courseTugboat      integer
**                  speedTugboat       integer
**                  maxPayloadTugboat   integer
**                  allowedTugboat     integer
**                  consistencyGUID    octet string
**                  consistencyChildCount integer
**
** Class:        policyParametersTugboat
**
** Classes Modified: container, groupPolicyContainer
**
** The modified classes have the two consistency
** attributes added.
**
** Notes:        Must run on a DC which is the current Schema Master
**                 and has schema updates enabled using the registry key
**                 and value:
**
** KEY:  HKEY_LOCAL_MACHINE\CurrentControlSet\Services\NTDS\Parameters
** Value: Schema Update Allowed, REG_DWORD, 1
**
** Libraries:  activeds.lib, adsiid.lib
**
*/
#include <windows.h>
#include <ole2.h>
#include <iads.h>
#include <activeds.h>
#include <stdio.h>
#include <atldbase.h>

#define COURSE_ATTR_NAME L"Course-Tugboat"
#define COURSE_ATTR_RDN L"CN=Course-Tugboat"
#define COURSE_ATTR_LDAP_NAME L"courseTugboat"

#define SPEED_ATTR_NAME L"Speed-Tugboat"
#define SPEED_ATTR_RDN L"CN=Speed-Tugboat"
#define SPEED_ATTR_LDAP_NAME L"speedTugboat"

#define MAX_PAYLOAD_ATTR_NAME L"Max-Payload-Tugboat"
#define MAX_PAYLOAD_ATTR_RDN L"CN=Max-Payload-Tugboat"
#define MAX_PAYLOAD_ATTR_LDAP_NAME L"maxPayloadTugboat"

#define ALLOWED_ATTR_NAME L"Allowed-Tugboat"
#define ALLOWED_ATTR_RDN L"CN=Allowed-Tugboat"
#define ALLOWED_ATTR_LDAP_NAME L"allowedTugboat"
```

```

#define CONSISTENCY_GUID_ATTR_NAME L"Consistency-GUID"
#define CONSISTENCY_GUID_ATTR_RDN L"CN=Consistency-GUID"
#define CONSISTENCY_GUID_ATTR_LDAP_NAME L"consistencyGUID"

#define CONSISTENCY_CHILD_ATTR_NAME L"Consistency-Child-Count"
#define CONSISTENCY_CHILD_ATTR_RDN L"CN=Consistency-Child-Count"
#define CONSISTENCY_CHILD_ATTR_LDAP_NAME L"consistencyChildCount"

#define TUGBOAT_CLASS_NAME L"Policy-Parameters-Tugboat"
#define TUGBOAT_CLASS_RDN L"CN=Policy-Parameters-Tugboat"

// Forward declaration of the error report.

void ReportErrorW(LPCWSTR pwszDefaultMsg, DWORD dwErr);

// GUIDs for schema extensions. Provide your own GUID for each
// extension so they are the same in every installation. If the
// DS assigns them your GUIDs will be different in every installation;
// this will affect system performance when moving to an
// identity-based schema.

// GUIDS for the policy attributes

// {C45F05B2-4D16-11d2-800E-0080C76670C0}
static const GUID attrGuid1 =
{ 0xc45f05b2, 0x4d16, 0x11d2, { 0x80, 0xe, 0x0, 0x80, 0xc7, 0x66, 0x70, 0xc0 } };

// {C45F05B3-4D16-11d2-800E-0080C76670C0}
static const GUID attrGuid2 =
{ 0xc45f05b3, 0x4d16, 0x11d2, { 0x80, 0xe, 0x0, 0x80, 0xc7, 0x66, 0x70, 0xc0 } };

// {C45F05B4-4D16-11d2-800E-0080C76670C0}
static const GUID attrGuid3 =
{ 0xc45f05b4, 0x4d16, 0x11d2, { 0x80, 0xe, 0x0, 0x80, 0xc7, 0x66, 0x70, 0xc0 } };

// {C45F05B6-4D16-11d2-800E-0080C76670C0}
static const GUID attrGuid4 =
{ 0xc45f05b6, 0x4d16, 0x11d2, { 0x80, 0xe, 0x0, 0x80, 0xc7, 0x66, 0x70, 0xc0 } };

// GUIDs for Consistency checking attributes

// {7707464B-4D9D-11d2-AF2D-00C04FB9624E}
static const GUID attrGuid5 =
{ 0x7707464b, 0x4d9d, 0x11d2, { 0xaf, 0x2d, 0x0, 0xc0, 0x4f, 0xb9, 0x62, 0x4e } };

// {7707464C-4D9D-11d2-AF2D-00C04FB9624E}
static const GUID attrGuid6 =
{ 0x7707464c, 0x4d9d, 0x11d2, { 0xaf, 0x2d, 0x0, 0xc0, 0x4f, 0xb9, 0x62, 0x4e } };

// {C45F05B5-4D16-11d2-800E-0080C76670C0}
static const GUID classGuid1 =
{ 0xc45f05b5, 0x4d16, 0x11d2, { 0x80, 0xe, 0x0, 0x80, 0xc7, 0x66, 0x70, 0xc0 } };

void main(void)
{
    HRESULT hr;
    IADs *pRoot = NULL;
    VARIANT varDSRoot,
        varSchemaUpdate;
    LPWSTR pwszDSPath = NULL;
    LPWSTR pwszGPCPath = NULL;
    LPWSTR pwszContPath = NULL;

    // Returned by CreateDSObject
    IDispatch *pDisp = NULL;

    // Pointers to schema objects
    IDirectoryObject *pSchema = NULL,
        *pGpc = NULL;

    // Data structures for creating schema objects
}

```

```

// Data structures for creating schema objects
//
// attribute values: These are unions and cannot be
// statically initialized.
//
ADSVVALUE      cn,
                singleValued,
                oid,
                syntax,
                omSyntax,
                ldapname,
                idGuid,
                objectClass,
                objectClassCategory,
                subClassOf,
                defaultSecurityDesc,
                defaultHidingValue,
                mayContain[6];

// ATTR_INFO for creating an attributeSchema object
// Each ADS_ATTR_INFO describes one attribute of an object to be
// stored in the DS.

ADSV_ATTR_INFO attrArray[] = {
    {L"cn", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &cn, 1},
    {L"isSingleValued", ADS_ATTR_UPDATE, ADSTYPE_BOOLEAN, &singleValued, 1},
    {L"objectClass", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &objectClass, 1},
    {L"attributeID", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &oid, 1},
    {L"attributeSyntax", ADS_ATTR_UPDATE, ADSTYPE_INTEGER, &syntax, 1},
    {L"oMSyntax", ADS_ATTR_UPDATE, ADSTYPE_INTEGER, &omSyntax, 1},
    {L"LDAPDisplayName", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &ldapname, 1},
    {L"schemaIdGUID", ADS_ATTR_UPDATE, ADSTYPE_OCTET_STRING, &idGuid, 1},
};

// ATTR_INFO for creating a classSchema object

ADSV_ATTR_INFO classArray[] = {
    {L"cn", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &cn, 1},
    {L"objectClass", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &objectClass, 1},
    {L"governsID", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &oid, 1},
    {L"objectClassCategory", ADS_ATTR_UPDATE, ADSTYPE_INTEGER, &objectClassCategory, 1},
    {L"schemaIdGUID", ADS_ATTR_UPDATE, ADSTYPE_OCTET_STRING, &idGuid, 1},
    {L"defaultSecurityDescriptor", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &defaultSecurityDesc,
1},
    {L"defaultHidingValue", ADS_ATTR_UPDATE, ADSTYPE_BOOLEAN, &defaultHidingValue, 1},
    {L"subClassOf", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &subClassOf, 1},
    {L"mayContain", ADS_ATTR_UPDATE, ADSTYPE_CASE_IGNORE_STRING, &mayContain[0], 6},
};

// ATTR_INFO for adding attributes to Group Policy Container
ADSV_ATTR_INFO gpcUpdate[] = {{L"mayContain", ADS_ATTR_APPEND, ADSTYPE_CASE_IGNORE_STRING,
&mayContain[0], 2},};

DWORD          dwAttrs;
ULONG          iAttrsMod;

hr = CoInitializeEx(NULL, COINIT_MULTITHREADED | COINIT_DISABLE_OLE1DDE);

// Get the name of the schema container for this domain.
// Read the Root DSE from the default DS, which will be
// the DS for the local domain. This will get the name of the
// schema container, stored in the "schemaNamingContext"
// operational attribute.

hr = ADsGetObject(L"LDAP://RootDSE",
                  IID_IADs,
                  (void**)&pRoot);
if(FAILED(hr))
{

```

```

    {
        goto cleanup;
    }

    // Get IDirectoryObject on the root DSE as well; use this for
    // forcing a schema update later.

    VariantInit(&varDSRoot);
    hr = pRoot->Get(CComBSTR("schemaNamingContext"), &varDSRoot);
    if(FAILED(hr))
    {
        goto cleanup;
    }
    wprintf(L"\nDS Root:%S\n", varDSRoot.bstrVal);

    // ADsPath of the schema container
    pwszDSPath = new WCHAR[7 + lstrlenW(varDSRoot.bstrVal) + 1];
    if(!pwszDSPath)
    {
        goto cleanup;
    }
    wcscpy_s(pwszDSPath, L"LDAP://");
    wcscat_s(pwszDSPath, varDSRoot.bstrVal);

    // ADsPath of the Group Policy Container class-schema object
    pwszGPCPath = new WCHAR[33 + lstrlenW(varDSRoot.bstrVal) + 1];
    if(!pwszGPCPath)
    {
        goto cleanup;
    }
    wcscpy_s(pwszGPCPath, L"LDAP://CN=Group-Policy-Container,");
    wcscat_s(pwszGPCPath, varDSRoot.bstrVal);

    // ADsPath of the Container class-schema object
    pwszContPath = new WCHAR[20 + lstrlenW(varDSRoot.bstrVal) + 1];
    if(!pwszContPath)
    {
        goto cleanup;
    }
    wcscpy_s(pwszContPath, L"LDAP://CN=Container,");
    wcscat_s(pwszContPath, varDSRoot.bstrVal);

    // Bind to the schema container and get the IDirectoryObject
    // interface for it.
    hr = ADsGetObject(pwszDSPath,
                      IID_IDirectoryObject,
                      (void**)&pSchema);
    if(FAILED(hr))
    {
        goto cleanup;
    }

//*****
// Consistency-Child-Count
//*****


    // Create a new attribute object. Set the values into the
    // attribute unions, then call the method to create the object.
    //

    // Calculate attribute count:
    dwAttrs = sizeof(attrArray)/sizeof(ADS_ATTR_INFO);

    cn.dwType = ADSTYPE_CASE_IGNORE_STRING;
    cn.CaseIgnoreString = CONSISTENCY_CHILD_ATTR_NAME;
    singleValued.dwType = ADSTYPE_BOOLEAN;
    singleValued.Boolean = VARIANT_TRUE;
    oid.dwType = ADSTYPE_CASE_IGNORE_STRING;

```

```

// Reserved. Test OID:
oid.CaseIgnoreString = L"1.2.840.113556.1.4.7000.161";
objectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
objectClass.CaseIgnoreString = L"attributeSchema";
syntax.dwType = ADSTYPE_CASE_IGNORE_STRING;
syntax.CaseIgnoreString = L"2.5.5.9"; // 2.5.5.9 = Integer
omSyntax.dwType = ADSTYPE_INTEGER;
omSyntax.Integer = 2;
//
// The LDAP display name is defaulted by the server and
// should not be provided unless different than the name
// computed from the CN attribute - the LDAP name is the CN
// with the hyphens removed and case delimiting substituted.
// The initial character is always lowercase. For this example,
// an explicit LDAP display name is provided to show
// how it is done.
ldapname.dwType = ADSTYPE_CASE_IGNORE_STRING;
ldapname.CaseIgnoreString = CONSISTENCY_CHILD_ATTR_LDAP_NAME;
//
// Schema-ID-Guid is provided by the server is the client
// does not provide it. This is a good example of how to
// write an Octet String to the DS.
idGuid.dwType = ADSTYPE_OCTET_STRING;
idGuid.OctetString.dwLength = sizeof(attrGuid6);
idGuid.OctetString.lpValue = (LPBYTE)&attrGuid6;

hr = pSchema->CreateDSObject(CONSISTENCY_CHILD_ATTR_RDN,
                               attrArray,
                               dwAttrs,
                               &pDisp);

if (FAILED(hr))
{
    ReportErrorW(L"CreateDSObject failed.", hr);
}
else
{
    wprintf(L"\n");
    wprintf(CONSISTENCY_CHILD_ATTR_NAME);
    wprintf(L" Attribute defined.\n");

    // The IDispatch interface returned by the CreateDSObject
    // call is not used. Release it now.
    pDisp->Release();
    pDisp = NULL;
}

//*****
// Consistency-GUID
//*****


// Create a new attribute object. Set the values into the
// attribute unions, then call the method to create the object.
//


cn.dwType = ADSTYPE_CASE_IGNORE_STRING;
cn.CaseIgnoreString = L"Consistency-GUID";
singleValued.dwType = ADSTYPE_BOOLEAN;
singleValued.Boolean = VARIANT_TRUE;
oid.dwType = ADSTYPE_CASE_IGNORE_STRING;
// Reserved. Test OID:
oid.CaseIgnoreString = L"1.2.840.113556.1.4.7000.160";
objectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
objectClass.CaseIgnoreString = L"attributeSchema";
syntax.dwType = ADSTYPE_CASE_IGNORE_STRING;
syntax.CaseIgnoreString = L"2.5.5.10"; // 2.5.5.10 = Octet String
omSyntax.dwType = ADSTYPE_INTEGER;
omSyntax.Integer = 4;

```

```

// The LDAP display name will be defaulted by the server and
// should not be provided unless different than the name
// computed from the CN attribute - the LDAP name is the CN
// with the hyphens removed and case delimiting substituted.
// The initial character is always lowercase. For this example,
// an explicit LDAP display name is provided to show
// how it is done.
ldapname.dwType = ADSTYPE_CASE_IGNORE_STRING;
ldapname.CaseIgnoreString = CONSISTENCY_GUID_ATTR_LDAP_NAME;
//
// Schema-ID-Guid is provided by the server is the client does
// not provide it. This is a good example of how to write an
// Octet String to the DS.
idGuid.dwType = ADSTYPE_OCTET_STRING;
idGuid.OctetString.dwLength = sizeof(attrGuid5);
idGuid.OctetString.lpValue = (LPBYTE)&attrGuid5;

hr = pSchema->CreateDSObject(CONSISTENCY_GUID_ATTR_RDN,
                               attrArray,
                               dwAttrs,
                               &pDisp);

if (FAILED(hr))
{
    ReportErrorW(L"CreateDSObject failed.", hr);
}
else
{
    wprintf(L"\n");
    wprintf(CONSISTENCY_GUID_ATTR_NAME);
    wprintf(L" Attribute defined.\n");

    // The IDispatch interface, returned by the CreateDSObject
    // call, is not used. Release it now.
    pDisp->Release();
    pDisp = NULL;
}

//*****
// Course-Tugboat
//*****


// Create a new attribute object. Set the values into the
// attribute unions, then call the method to create the object.

// Calculate attribute count:
dwAttrs = sizeof(attrArray)/sizeof(ADS_ATTR_INFO);

cn.dwType = ADSTYPE_CASE_IGNORE_STRING;
cn.CaseIgnoreString = COURSE_ATTR_NAME;
singleValued.dwType = ADSTYPE_BOOLEAN;
singleValued.Boolean = VARIANT_TRUE;
oid.dwType = ADSTYPE_CASE_IGNORE_STRING;

// Reserved. Test OID:
oid.CaseIgnoreString = L"1.2.840.113556.1.4.7000.155";
objectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
objectClass.CaseIgnoreString = L"attributeSchema";
syntax.dwType = ADSTYPE_CASE_IGNORE_STRING;
syntax.CaseIgnoreString = L"2.5.5.9";    // 2.5.5.9 = Integer
omSyntax.dwType = ADSTYPE_INTEGER;
omSyntax.Integer = 2;
//
// The LDAP display name is defaulted by the server and should
// not be provided unless different than the name computed from
// the CN attribute - the LDAP name is the CN with the hyphens
// removed and case delimiting substituted. The initial character
// is always lowercase. For this example, an explicit LDAP display
// name is provided to show how it is done.

```

```

ldapname.dwType = ADSTYPE_CASE_IGNORE_STRING;
ldapname.CaseIgnoreString = COURSE_ATTR_LDAP_NAME;
//
// Schema-ID-Guid is provided by the server if the client does
// not provide it. This is a good example of how to write an
// Octet String to the DS.
idGuid.dwType = ADSTYPE_OCTET_STRING;
idGuid.OctetString.dwLength = sizeof(attrGuid1);
idGuid.OctetString.lpValue = (LPBYTE)&attrGuid1;

hr = pSchema->CreateDSObject(COURSE_ATTR_RDN,
                               attrArray,
                               dwAttrs,
                               &pDisp);

if (FAILED(hr))
{
    ReportErrorW(L"CreateDSObject failed.", hr);
}
else
{
    wprintf(L"\n");
    wprintf(COURSE_ATTR_NAME);
    wprintf(L" Attribute defined.\n");

    // Do not use the IDispatch interface returned by the
    // CreateDSObject call. Release it now.
    pDisp->Release();
    pDisp = NULL;
}

//*****
// Speed-Tugboat
//*****

// Create a new attribute object. Set the values into the
// attribute unions, then call the method to create the object.

// Calculate attribute count:
dwAttrs = sizeof(attrArray)/sizeof(ADS_ATTR_INFO);

cn.dwType = ADSTYPE_CASE_IGNORE_STRING;
cn.CaseIgnoreString = SPEED_ATTR_NAME;
singleValued.dwType = ADSTYPE_BOOLEAN;
singleValued.Boolean = VARIANT_TRUE;
oid.dwType = ADSTYPE_CASE_IGNORE_STRING;

// Reserved. Test OID:
oid.CaseIgnoreString = L"1.2.840.113556.1.4.7000.156";
objectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
objectClass.CaseIgnoreString = L"attributeSchema";
syntax.dwType = ADSTYPE_CASE_IGNORE_STRING;
syntax.CaseIgnoreString = L"2.5.5.9"; // 2.5.5.9 = Integer
omSyntax.dwType = ADSTYPE_INTEGER;
omSyntax.Integer = 2;
//
// The LDAP display name is defaulted by the server and should
// not be provided unless different than the name computed
// from the CN attribute - the LDAP name is the CN with the
// hyphens removed and case delimiting substituted. The initial
// character is always lowercase. For this example, an explicit
// LDAP display name is provided to show how it is done.
ldapname.dwType = ADSTYPE_CASE_IGNORE_STRING;
ldapname.CaseIgnoreString = SPEED_ATTR_LDAP_NAME;
//
// Schema-ID-Guid is provided by the server if the client does
// not provide it. This is a good example of how to write an
// Octet String to the DS.
idGuid.dwType = ADSTYPE_OCTET_STRING;

```

```

idGuid.OctetString.dwLength = sizeof(attrGuid2);
idGuid.OctetString.lpValue = (LPBYTE)&attrGuid2;

hr = pSchema->CreateDSObject(SPEED_ATTR_RDN,
                               attrArray,
                               dwAttrs,
                               &pDisp);

if (FAILED(hr))
{
    ReportErrorW(L"CreateDSObject failed.", hr);
}
else
{
    wprintf(L"\n");
    wprintf(SPEED_ATTR_NAME);
    wprintf(L" Attribute defined.\n");

    // Do not use the IDispatch interface returned by the
    // CreateDSObject call. Release it now.
    pDisp->Release();
    pDisp = NULL;
}

//*****
// Max-Payload-Tugboat
//*****

// Create a new attribute object. Set the values into the
// attribute unions, then call the method to create the object.
//

// Calculate attribute count:
dwAttrs = sizeof(attrArray)/sizeof(ADS_ATTR_INFO);

cn.dwType = ADSTYPE_CASE_IGNORE_STRING;
cn.CaseIgnoreString = MAX_PAYLOAD_ATTR_NAME;
singleValued.dwType = ADSTYPE_BOOLEAN;
singleValued.Boolean = VARIANT_TRUE;
oid.dwType = ADSTYPE_CASE_IGNORE_STRING;
oid.CaseIgnoreString = L"1.2.840.113556.1.4.7000.157";
objectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
objectClass.CaseIgnoreString = L"attributeSchema";
syntax.dwType = ADSTYPE_CASE_IGNORE_STRING;
syntax.CaseIgnoreString = L"2.5.5.9"; // 2.5.5.9 = Integer
omSyntax.dwType = ADSTYPE_INTEGER;
omSyntax.Integer = 2;
//
// The LDAP display name is defaulted by the server and should
// not be provided unless different than the name computed from
// the CN attribute - the LDAP name is the CN with the hyphens
// removed and case delimiting substituted. The initial character
// is always lowercase. For this example, an explicit LDAP
// display name is provided to show how it is done.
ldapname.dwType = ADSTYPE_CASE_IGNORE_STRING;
ldapname.CaseIgnoreString = MAX_PAYLOAD_ATTR_LDAP_NAME;
//
// Schema-ID-Guid is provided by the server is the client
// does not provide it. This is a good example of how to write
// an Octet String to the DS.
idGuid.dwType = ADSTYPE_OCTET_STRING;
idGuid.OctetString.dwLength = sizeof(attrGuid3);
idGuid.OctetString.lpValue = (LPBYTE)&attrGuid3;

hr = pSchema->CreateDSObject(MAX_PAYLOAD_ATTR_RDN,
                               attrArray,
                               dwAttrs,
                               &pDisp);

if (FAILED(hr))
{
    ReportErrorW(L"CreateDSObject failed.", hr);
}

```

```

    {
        ReportErrorW(L"CreateDSObject failed.", hr);
    }
    else
    {
        wprintf(L"\n");
        wprintf(MAX_PAYLOAD_ATTR_NAME);
        wprintf(L" Attribute defined.\n");

        // Do not use the IDispatch interface returned by the
        // CreateDSObject call. Release it now.
        pDisp->Release();
        pDisp = NULL;
    }

    //*****
    // Allowed-Tugboat
    //*****

    // Create a new attribute object. Set the values into the
    // attribute unions, then call the method to create the object.
    //

    // Calculate attribute count:
    dwAttrs = sizeof(attrArray)/sizeof(ADS_ATTR_INFO);

    cn.dwType = ADSTYPE_CASE_IGNORE_STRING;
    cn.CaseIgnoreString = ALLOWED_ATTR_NAME;
    singleValued.dwType = ADSTYPE_BOOLEAN;
    singleValued.Boolean = VARIANT_TRUE;
    oid.dwType = ADSTYPE_CASE_IGNORE_STRING;
    oid.CaseIgnoreString = L"1.2.840.113556.1.4.7000.159";
    objectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
    objectClass.CaseIgnoreString = L"attributeSchema";
    syntax.dwType = ADSTYPE_CASE_IGNORE_STRING;
    syntax.CaseIgnoreString = L"2.5.5.8";      // 2.5.5.8 = Boolean
    omSyntax.dwType = ADSTYPE_INTEGER;
    omSyntax.Integer = 1;
    //
    // The LDAP display name is defaulted by the server and should
    // not be provided unless different than the name computed
    // from the CN attribute - the LDAP name is the CN with the
    // hyphens removed and case delimiting substituted. The initial
    // character is always lowercase. For this example, an explicit
    // LDAP display name is provided to show how it is done.
    ldapname.dwType = ADSTYPE_CASE_IGNORE_STRING;
    ldapname.CaseIgnoreString = ALLOWED_ATTR_LDAP_NAME;
    //
    // Schema-ID-Guid is provided by the server is the client does not
    // provide it. This is a good example of how to write an
    // Octet String to the DS.

    idGuid.dwType = ADSTYPE_OCTET_STRING;
    idGuid.OctetString.dwLength = sizeof(attrGuid4);
    idGuid.OctetString.lpValue = (LPBYTE)&attrGuid4;

    hr = pSchema->CreateDSObject(ALLOWED_ATTR_RDN,
                                  attrArray,
                                  dwAttrs,
                                  &pDisp);
    if (FAILED(hr))
    {
        ReportErrorW(L"CreateDSObject failed.", hr);
    }
    else
    {
        wprintf(L"\n");
        wprintf(ALLOWED_ATTR_NAME);
    }
}

```

```

wprintf(allowed_attr_name),
wprintf(L" Attribute defined.\n");

// Do not use the IDispatch interface returned by the
// CreateDSObject call. Release it now.
pDisp->Release();
pDisp = NULL;
}

// Force an update of the schema cache to create the class that
// includes these attributes. Force a synchronous schema update
// by writing the operational attribute "schemaUpdateNow" to
// the Root DSE.
varSchemaUpdate.vt = VT_I4;
varSchemaUpdate.intValue = 1;
hr = pRoot->Put(CComBSTR("schemaUpdateNow"), varSchemaUpdate);
hr = pRoot->SetInfo();
if (FAILED(hr))
{
    ReportErrorW(L"Force Schema Recalc failed.", hr);
}

//*****
// Policy-Parameters-Tugboat
//*****
// Create a new class object and add attributes
// (including the new ones) to the class.
//
cn.dwType = ADSTYPE_CASE_IGNORE_STRING;
cn.IgnoreString = TUGBOAT_CLASS_NAME;
objectClass.dwType = ADSTYPE_CASE_IGNORE_STRING;
objectClass.IgnoreString = L"classSchema";
oid.dwType = ADSTYPE_CASE_IGNORE_STRING;
oid.IgnoreString = L"1.2.840.113556.1.5.7000.92";
subClassOf.dwType = ADSTYPE_CASE_IGNORE_STRING;
subClassOf.IgnoreString = L"serviceConnectionPoint";
defaultSecurityDesc.dwType = ADSTYPE_CASE_IGNORE_STRING;
defaultSecurityDesc.IgnoreString = L"D:(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;DA)
(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;SY)(A;;RPLCLORC;;AU)(A;SAFA;WDWOSDDTWPCCRCCDCSW;;;WD)";
defaultHidingValue.dwType = ADSTYPE_BOOLEAN;
defaultHidingValue.Boolean = -1;
mayContain[0].dwType = ADSTYPE_CASE_IGNORE_STRING;
mayContain[0].IgnoreString = COURSE_ATTR_LDAP_NAME;
mayContain[1].dwType = ADSTYPE_CASE_IGNORE_STRING;
mayContain[1].IgnoreString = SPEED_ATTR_LDAP_NAME;
mayContain[2].dwType = ADSTYPE_CASE_IGNORE_STRING;
mayContain[2].IgnoreString = MAX_PAYLOAD_ATTR_LDAP_NAME;
mayContain[3].dwType = ADSTYPE_CASE_IGNORE_STRING;
mayContain[3].IgnoreString = ALLOWED_ATTR_LDAP_NAME;
mayContain[4].dwType = ADSTYPE_CASE_IGNORE_STRING;
mayContain[4].IgnoreString = CONSISTENCY_GUID_ATTR_LDAP_NAME;
mayContain[5].dwType = ADSTYPE_CASE_IGNORE_STRING;
mayContain[5].IgnoreString = CONSISTENCY_CHILD_ATTR_LDAP_NAME;

// Object-Class-Category=
// 88_CLASS      0
// STRUCTURAL_CLASS  1
// ABSTRACT_CLASS   2
// AUXILIARY_CLASS  3
objectClassCategory.dwType = ADSTYPE_INTEGER;
objectClassCategory.Integer = 1;

// Calculate attribute count:
dwAttrs = sizeof(classArray)/sizeof(ADS_ATTR_INFO);

hr = pSchema->CreateDSObject(TUGBOAT_CLASS_RDN,

```

```

        classArray,
        dwAttrs,
        &pDisp);

    if (FAILED(hr))
    {
        ReportErrorW(L"CreateDSObject failed.", hr);
    }
    else
    {
        wprintf(L"\nClass defined.\n");

        // Do not use the IDispatch interface returned by the
        // CreateDSObject call. Release it now.
        pDisp->Release();
        pDisp = NULL;
    }

    //*****
    // Add the consistency attributes to Group-Policy-Container
    // and Container
    //*****

    hr = ADsGetObject(pwszGPPPath,
                      IID_IDirectoryObject,
                      (void**)&pGpc);

    if (FAILED(hr))
    {
        ReportErrorW(L"Read GPC class object failed.", hr);
        goto cleanup;
    }
    else
    {
        wprintf(L"\nRetrieved GPC class object.\n");

        mayContain[0].dwType = ADSTYPE_CASE_IGNORE_STRING;
        mayContain[0].CaseIgnoreString =
            CONSISTENCY_GUID_ATTR_LDAP_NAME;
        mayContain[1].dwType = ADSTYPE_CASE_IGNORE_STRING;
        mayContain[1].CaseIgnoreString =
            CONSISTENCY_CHILD_ATTR_LDAP_NAME;
        hr = pGpc->SetObjectAttributes(gpcUpdate,
                                         1,
                                         &iAttrsMod);

        if (FAILED(hr))
        {
            ReportErrorW(L"Update GPC Class object failed.",
                         hr);
        }
        else
        {
            wprintf(L"\nUpdated GPC Class object.\n");
        }
    }

    // Done with class object.
    pGpc->Release();
    pGpc = NULL;

    //
    // Apply the consistency attributes to Container. The
    // ATTR_INFO required is filled in. Apply it.
    //
    hr = ADsGetObject(pwszContPath,
                      IID_IDirectoryObject,
                      (void**)&pGpc);

    if (FAILED(hr))
    {

```

```

        ReportErrorW(L"Read Container class object failed.",
                     hr);
        goto cleanup;
    }
    else
    {
        wprintf(L"\nRetrieved Container class object.\n");

        hr = pGpc->SetObjectAttributes(gpcUpdate,1,&iAttrsMod);
        if (FAILED(hr))
        {
            ReportErrorW(L"Update Container Class object failed.",
                         hr);
        }
        else
        {
            wprintf(L"\nUpdated Container Class object.\n");
        }
    }

    // Force an update of the schema cache to use the changes
    // immediately. Force a synchronous schema update by writing
    // the operational attribute "schemaUpdateNow" to
    // the Root DSE.
    varSchemaUpdate.vt = VT_I4;
    varSchemaUpdate.intVal = 1;
    hr = pRoot->Put(CComBSTR("schemaUpdateNow"),
                      varSchemaUpdate);
    hr = pRoot->SetInfo();
    if (FAILED(hr))
    {
        ReportErrorW(L"Force Schema Recalc failed.", hr);
    }
}

cleanup:
    VariantClear(&varDSRoot);
    VariantClear(&varSchemaUpdate);

    if(pwszDSPath)
    {
        delete pwszDSPath;
    }

    if(pwszGPCPath)
    {
        delete pwszGPCPath;
    }

    if(pwszContPath)
    {
        delete pwszContPath;
    }

    if(pDisp)
    {
        pDisp->Release();
    }

    if(pSchema)
    {
        pSchema->Release();
    }

    if(pRoot)
    {
        pRoot->Release();
    }

```

```
CoUninitialize();  
}  
  
// Simple error message reporter  
  
void ReportErrorW(LPCWSTR pwszDefaultMsg, DWORD dwErr)  
{  
    DWORD    dwStatus;  
    LPWSTR   pwszMsg;  
  
    dwStatus = FormatMessageW(FORMAT_MESSAGE_FROM_SYSTEM |  
                             FORMAT_MESSAGE_ALLOCATE_BUFFER |  
                             FORMAT_MESSAGE_IGNORE_INSERTS,  
                             NULL,  
                             dwErr,  
                             LANG_NEUTRAL,  
                             (LPWSTR)&pwszMsg,  
                             64,  
                             NULL);  
  
    if (dwStatus != 0)  
    {  
        wprintf(L"%s %s", pwszDefaultMsg, pwszMsg);  
        LocalFree(pwszMsg);  
    }  
    else  
    {  
        wprintf(L"%s %X\n", pwszDefaultMsg, dwErr);  
    }  
}
```

Extending the User Interface for Directory Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

With Microsoft Windows 2000 and later, Microsoft Management Console (MMC) snap-ins, such as the Active Directory Users and Computers snap-in, for administration of Active Directory Domain Services, are available. In addition, the Windows 2000 shell includes user interfaces (UIs) for finding objects that reside in the directory and reading and writing properties. This section explains how to extend the UI for viewing and managing Active Directory Domain Services objects in the Windows shell and Active Directory administrative snap-ins. This section also covers what is required to deploy the UI extensions to the user's desktops.

Specifically, this section discusses the following:

- [About Active Directory User Interfaces](#)
- [Display Specifiers](#)
- [Class and Attribute Display Names](#)
- [Class Icons](#)
- [Viewing Containers as Leaf Nodes](#)
- [Modifying Existing User Interfaces](#)
- [User Interface Extension for New Object Classes](#)
- [Property Pages for Use with Display Specifiers](#)
- [Context Menus for Use with Display Specifiers](#)
- [Object Creation Wizards](#)
- [Using Standard Dialog Boxes for Handling Objects in Active Directory Domain Services](#)
- [Administrative Notification Handlers](#)
- [Distributing User Interface Components](#)
- [How Applications Should Use Display Specifiers](#)
- [Debugging an Active Directory Extension](#)

About the User Interfaces of Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Administrators and users must be able to view and modify directory service objects in Microsoft Active Directory Domain Services from a user interface. Administrators manage the Active Directory server using different Microsoft Management Console (MMC) snap-ins, specifically, the Active Directory Domain Services Users and Computers, Active Directory Domain Services Sites and Services, Active Directory Domain Services Domains and Trusts, and Active Directory Domain Services Schema Manager snap-ins. The end user, if granted appropriate permissions, can also use the Active Directory MMC snap-ins to view directory objects. The Windows shell can also be used to view directory objects. The Windows shell enables users to browse objects stored in the directory from either the directory item in My Network Places on the desktop or through the search dialogs available in the Start menu.

Active Directory Domain Services support a user interface that adapts to meet the needs of administrators and end users. Active Directory Domain Services enable you to extend the user interface that represents existing object classes as well as new classes added to the schema. The following elements can be used to control or extend the UI for each class defined in the schema:

- [Property Pages for Use with Display Specifiers](#)
- [Context Menus for Use with Display Specifiers](#)
- [Class and Attribute Display Names](#)
- [Object Creation Wizards](#)
- [Class Icons](#)
- [Viewing Containers as Leaf Nodes](#)

Beginning with Windows 2000, the system provides COM objects that implement common dialog boxes for working with directory objects. These dialog boxes provide a common UI without having to reimplement these dialog boxes.

- [Directory Object Picker](#)
- [Domain Browser](#)
- [Container Browser](#)

Active Directory administrative snap-ins, context menus, and property pages can be extended using MMC extension snap-ins. Other MMC extensions, such as taskpads, namespace items, control bars, and toolbars can also be implemented. For more information, see [Extending the Active Directory Administrative Snap-ins](#).

Display Specifiers

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Active Directory Domain Services, an object class, or class, defines a type of object that can be created in Active Directory Domain Services. The definition of each object class is stored as a **classSchema** object in the [Active Directory Schema](#). In addition to the **classSchema** definition, each object class can have one or more display specifiers that specify the user interface data for objects of that class.

A display specifier is an object of the **displaySpecifier** class. The attributes of a **displaySpecifier** object specify localized user interface data that describes the various UI elements for a particular object class. Display specifiers store data for property sheets, context menus, icons, creation wizards, and localized class and attribute names. For property pages and context menus, the Windows shell and Active Directory administrative snap-ins use this data to form different user interfaces for administrators and end users—one set of property pages and/or context menus can be associated with administrative applications while a different set of elements can be associated with end user applications.

The **displaySpecifier** objects are stored in locale-specific containers in the **DisplaySpecifiers** container in the Configuration container, which is replicated to every domain controller in the enterprise forest. The **DisplaySpecifiers** container has subcontainers that correspond to the various locales supported by the enterprise installation. These subcontainers are named using language identifiers. For example, the name of the locale container for US-English is 409, which corresponds to the hexadecimal language identifier, 0x0409. Thus, an object class can have multiple display specifiers: one in each locale subcontainer. For more information, and a list of possible language identifiers, see [Language Identifier Constants and Strings](#). For more information about locales, see [Locales and Languages](#).

The name of a **displaySpecifier** object is formed by appending the string "-Display" to the **IDAPDisplayname** of the object class. For example, the name of a **displaySpecifier** object for the **user** class is "user-Display".

You can add, delete, or modify properties of a class's **displaySpecifier** objects to specify the UI elements (class name, attribute names, property sheets, context menus, icon, and so on) that appear for each instance of an object of that class.

For more information about display specifiers, see [DisplaySpecifiers Container](#).

DisplaySpecifiers Container

6/3/2022 • 2 minutes to read • [Edit Online](#)

Display specifiers are stored, by locale, in the DisplaySpecifiers container of the Configuration container. Because the Configuration container is replicated across the entire forest, display specifiers are propagated across all domains in a forest.

The Configuration container stores the DisplaySpecifiers container, which then stores containers that correspond to each locale. These locale containers are named using the hexadecimal representation of the locale identifier. For example, the US/English locale container is named 409, the German locale's container is named 407, and the Japanese locale's container is named 411. For more information, and a list of possible language identifiers, see [Language Identifier Constants and Strings](#).

Each locale container stores objects of the [displaySpecifier](#) class.

To list all display specifiers for a locale, enumerate all the [displaySpecifier](#) objects in the specified locale container within the DisplaySpecifiers container.

The following code example contains a function that binds to the display specifier container for the specified locale:

```
*****
This function returns a pointer to the display specifier container
for the specified locale.

If locale is NULL, use default system locale and then return the
locale in the locale parameter.
*****
HRESULT BindToDisplaySpecifiersContainerByLocale(
    LCID *locale,
    IADS **ppDispSpecCont)
{
    HRESULT hr = E_FAIL;

    if ((!ppDispSpecCont)||(!locale))
        return E_POINTER;

    // If no locale is specified, use the default system locale.
    if (!(*locale))
    {
        *locale = GetSystemDefaultLCID();
        if (!(*locale))
            return E_FAIL;
    }

    // Be sure that the locale is valid.
    if (!IsValidLocale(*locale, LCID_SUPPORTED))
        return E_INVALIDARG;

    LPOLESTR szPath = new OLECHAR[MAX_PATH*2];
    IADS *pObj = NULL;
    VARIANT var;

    hr = ADsOpenObject(L"LDAP://rootDSE",
                       NULL,
                       NULL,
                       ADS_SECURE_AUTHENTICATION,
                       IID_IADS,
                       (void**)&pObj);
```

```
if (SUCCEEDED(hr))
{
    // Get the DN to the configuration container.
    hr = pObj->Get(CComBSTR("configurationNamingContext"), &var);
    if (SUCCEEDED(hr))
    {
        // Build the string to bind to the container for the
        // specified locale in the DisplaySpecifiers container.
        swprintf_s(
            szPath,
            L"LDAP://cn=%x,cn=DisplaySpecifiers,%s",
            *locale,
            var.bstrVal);

        // Bind to the container.
        *ppDispSpecCont = NULL;
        hr = ADsOpenObject(szPath,
                           NULL,
                           NULL,
                           ADS_SECURE_AUTHENTICATION,
                           IID_IADs,
                           (void**)ppDispSpecCont);

        if(FAILED(hr))
        {
            if ((*ppDispSpecCont))
            {
                (*ppDispSpecCont)->Release();
                (*ppDispSpecCont) = NULL;
            }
        }
    }

    // Cleanup.
    VariantClear(&var);
    if (pObj)
        pObj->Release();

    return hr;
}
```

Class and Attribute Display Names

6/3/2022 • 2 minutes to read • [Edit Online](#)

The display specifier for an object class contains the following attributes that can be used to specify the localized display names used in the UI for objects of that class:

- The **classDisplayName** attribute is a single-value Unicode string that specifies the class display name.
- The **attributeDisplayNames** attribute is a multi-value property that specifies the names to use in the UI for attributes of the object class.

The **attributeDisplayNames** values are Unicode strings; each element consists of a comma-delimited name pair:

```
<attribute name>,<display text>
```

In this example, "<attribute name>" is the **IDAPDisplayName** of the attribute and "<display text>" is the text to display as the name of that attribute in the user interface.

Guidelines for Class and Attribute Display Names

Because many vendors may extend classes with new attributes or creating entirely new classes, it is important that the class and attribute display names are unambiguous and do not result in conflicts.

Each vendor should prefix the class display name with a unique friendly identifier based on the vendor name. For example, if the fictitious company, Fabrikam Inc., creates a new class derived from the "contact" class, they can have a unique class display name "Fabrikam Contact."

If a vendor extends an existing class with new attributes, they should again uniquely identify the attribute display name so that no conflicts occur with other attribute display names. Again, prefixing the attribute display name with unique friendly identifier based on the vendor name is good practice. For example, if the Fabrikam company extends the user class with a new HR attribute, they could uniquely display the attribute as "Fabrikam HR Information."

In addition, from a localization perspective, each vendor should localize the class and attribute display names into each language supported by Windows 2000.

Adding a Value to the attributeDisplayNames Attribute

To add a name mapping value to the attributeDisplayNames attribute

1. Determine if the name mapping value for the attribute exists. If a name mapping value is to be replaced, first deleted the existing value, using the **IADs::PutEx** method, with the *InControlCode* parameter set to **ADS_PROPERTY_DELETE** and the *vProp* parameter set to the value to be removed. Do not use **ADS_PROPERTY_CLEAR** or **ADS_PROPERTY_UPDATE** for *InControlCode*.
2. Create the string that represents the attribute display name. For an example, see the format above.
3. Use the **IADs::PutEx** method with the *InControlCode* parameter set to **ADS_PROPERTY_APPEND** to add the new value.
4. Call **IADs::SetInfo** to commit the changes to the directory.

For more information about naming new classes and attributes, see [Naming Attributes and Classes](#).

Class Icons

6/3/2022 • 2 minutes to read • [Edit Online](#)

The icon used to represent a class object can be specified in the **iconPath** attribute in the **DisplaySpecifiers** container. Moreover, each class can store multiple icon states. For example, a folder class can have icons for the open, closed, and disabled states. The current implementation accepts a maximum of sixteen different icon states per class.

The **iconPath** attribute can be specified in one of two ways.

```
<state>,<icon file name>
```

or

```
<state>,<module file name>,<resource ID>
```

In these examples, the "<state>" is an integer with a value between 0 and 15. The value 0 is defined to be the default or closed state of the icon. The value 1 is defined to be the open state of the icon. The value 2 is the disabled state. All other values are application-defined.

The "<icon file name>" is the path and file name of an icon file that contains the icon image.

The "<module file name>" is the path and file name of a module, such as an EXE or DLL, that contains the icon image in a resource. The "<resource ID>" is an integer that specifies the resource identifier of the icon resource within the module.

Adding a Value to the **iconPath** Attribute

To add a value to the **iconPath** attribute, perform the following steps.

1. Determine if the value for the attribute exists. If a value is to be replaced, first, delete the existing value using the [IADs::PutEx](#) method with the *InControlCode* parameter set to **ADS_PROPERTY_DELETE** and the *vProp* parameter set to the value to be removed. Do not use **ADS_PROPERTY_CLEAR** or **ADS_PROPERTY_UPDATE** for *InControlCode*.
2. Create the string that represents the attribute icon data. For an example, see the format above.
3. To add the new value, use the [IADs::PutEx](#) method with the *InControlCode* parameter set to **ADS_PROPERTY_APPEND**.
4. To commit changes to the directory, call [IADs::SetInfo](#).

Viewing Containers as Leaf Nodes

6/3/2022 • 2 minutes to read • [Edit Online](#)

Any object in Active Directory Domain Services can be a container of other objects. This can clutter the user interface, so it is possible to declare that an object of a specific class be only be displayed as a leaf in the user interface. The **treatAsLeaf** attribute is a single-valued display specifier attribute that determines if objects of that class should be only be displayed as leaf objects. This attribute is a Boolean value that, if **TRUE**, indicates that objects of the class should only be displayed as leaf elements. If **FALSE**, indicates that objects of the class can be displayed as a container or a leaf. Like all display specifier attributes, the **treatAsLeaf** attribute is set on a per-locale basis, so this attribute can be localized as required. For example, the **User-Display** for the English locale (0409) display specifier has the **treatAsLeaf** attribute set to **TRUE** by default. This causes the user interface to display all **User** objects as leaf objects.

To set the value of the **treatAsLeaf** attribute

1. Bind to the desired display attribute in the desired locale. For more information and a code example, see [DisplaySpecifiers Container](#).
2. Use the **IADs::Put** method to set the **treatAsLeaf** attribute to either **TRUE** or **FALSE**.
3. To commit changes to the directory, call the **IADs::SetInfo** method.

Modifying Existing User Interfaces

6/3/2022 • 2 minutes to read • [Edit Online](#)

The results pane of the Active Directory Users and Computers MMC snap-in displays several columns of attribute data for objects within a container, such as the **Name** and **Description** attributes. The snap-in enables the user to add and remove the columns displayed in the results pane of the snap-in.

To change the display, the user uses the **View** pull-down menu and selects **Add/Remove Columns**. In the **Add/Remove Columns** dialog box, there is a list of columns that the user can choose from to display in the results pane.

The Active Directory Users and Computers MMC snap-in that is included with Windows Server 2003, Standard Edition, Windows Server 2003, Enterprise Edition, and Windows Server 2003, Datacenter Edition, provides the ability to modify the list of columns that can be displayed in the results pane of the snap-in for a container. This feature only exists if the snap-in is targeted at a forest with Windows Server 2003 schema.

To add a column to the list, add a value to the **extraColumns** attribute of the display specifier for the object type that the attribute is associated with. The **extraColumns** attribute is a multi-valued string attribute where each string is in the following format.

```
<ldapdisplayname>,<column header>,<default visibility>,<width>,<unused>
```

The following table lists the contents of these values.

VALUE	DESCRIPTION
"<ldapdisplayname>"	Contains a string that represents the ldapDisplayName of the attribute.
"<column header>"	Contains a string that represents the text displayed in the header for the column.
"<default visibility>"	Contains a numeric value that is 0 if the attribute is hidden by default or 1 if the attribute is visible by default.
"<width>"	Contains the width of the column, in pixels. If this value is -1, the width of the column is set to the width of the column header.
"<unused>"	Unused. Must be zero.

For example, to add a column that will display the canonical name for objects in an organizational unit, a value for the **canonicalName** attribute is added to the **extraColumns** attribute of the **organizationalUnit-Display** object in the display specifiers container. The string added to the **extraColumns** attribute of the **organizationalUnit-Display** object will look like the following.

```
canonicalName,Canonical Name,0,150,0
```

The **Add/Remove Columns** dialog box displays only the columns that are contained in the **extraColumns** attribute of the **displaySpecifier** object of the container type that is being displayed. If the **extraColumns** attribute does not contain any values, the **Add/Remove Columns** dialog box will display a fixed set of columns. A copy of the fixed set of columns is contained in the **extraColumns** attribute of the **default-Display** object.

To add one or more columns to the list of columns for a specific object, you must copy all of the **extraColumns** values from the **default-Display** object to the target object and then add the custom columns. If you specify the **extraColumns** attribute on a given class, then that class will use those columns and will not merge them with the columns that are specified in the **default-Display** class. Therefore, further changes to the **default-Display** class will have no effect on that object.

To display a custom column for all container types that do not have any custom columns registered, add a value for the column to the **extraColumns** attribute of the **default-Display** object.

User Interface Extension for New Object Classes

6/3/2022 • 3 minutes to read • [Edit Online](#)

Active Directory Domain Services and its administrative MMC snap-in user interface can be customized to adapt to the requirements of administrators and users. Active Directory Domain Services enable the schema to be modified by creating new classes and attributes, or modifying existing classes. Display specifiers for the classes can be modified to reflect the new user interface elements that schema modifications require.

The following table lists attributes that can be used to modify how the administrative snap-ins will display objects of a particular class.

ATTRIBUTE	DESCRIPTION
<code>defaultHidingValue</code>	<p>The <code>defaultHidingValue</code> attribute is an attribute of a <code>classSchema</code> object. This attribute contains a Boolean value that, if TRUE, causes instances of the object class to be hidden in the administrative snap-ins and the Windows shell. This also means that a menu item for the new object class does not appear in the New context menu of the administrative snap-ins, even if the appropriate creation wizard properties are set on the new object class's <code>displaySpecifier</code> object. If this attribute is FALSE, instances of the class will be visible in the administrative snap-ins and the shell. This also causes a menu item to create a new object instance to be added to the New menu of the administrative snap-ins.</p> <p>If no value is set for this attribute, the default value is TRUE. This means that, by default, instances of the object are hidden.</p>
<code>showInAdvancedViewOnly</code>	<p>The <code>showInAdvancedViewOnly</code> attribute contains a Boolean value that, if TRUE, causes instances of the object class to appear in the Users and Computers snap-in in Advanced View only and do not appear in the Windows shell. If this property is FALSE, instances of the class will be visible in Normal view in the Users and Computers snap-in and the Windows shell.</p> <p>If no value is set for this attribute, the default value is TRUE. This attribute can be set on an individual object to override the value set on the object class. For example, the <code>Container</code> class has this attribute set to TRUE but the <code>User</code> container has this value set to FALSE. Because of this, the <code>User</code> container appears in the shell and in Normal view in the Users and Computers snap-in but other containers that do not have <code>showInAdvancedViewOnly</code> set to FALSE appear only in Advanced view in the Users and Computers snap-in.</p>

Creating Display Specifiers for New Classes

To customize the user interface for a new class, create a display specifier object for the new class for each supported locale, then set the desired attributes for the display specifier.

Inheriting Display Specifiers for Derived Classes

A new class that inherits from an existing class does not inherit the parent class display specifier. If the new class is to use some or all of the parent class display specifier properties, create a new display specifier for the new class and copy the properties from the parent class display specifier to the new object display specifier. This must be done for all locales for which the parent class display specifier properties apply.

Certain parts of the UI feature set, such as the menu items and creation wizard for the user class, are implemented internally and are not available for use by a derived object. In these instances it is better to extend an existing class than to use a derived class.

Modifying Existing Classes

New attributes can be added to an existing class. New UI components (property pages, menu items, and attribute display names) can be added or the existing UI replaced. It is also possible to design new property pages that expose fewer attributes of a class and to create context menus with fewer actions. For more information, see [Property Pages for Use with Display Specifiers](#), [Context Menus for Use with Display Specifiers](#), and [Class and Attribute Display Names](#).

Property Pages for Use with Display Specifiers

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services provide a mechanism for adding pages to the property sheet displayed for a directory object from the Active Directory administrative snap-ins or the Windows shell. To add a page to the property sheet, implement a property sheet extension.

Developer Audience

This documentation assumes that the reader is familiar with COM operation and component development using C++. It is not currently possible to create an Active Directory property sheet extension using Visual Basic.

Creating an Active Directory Property Sheet Extension

A *property sheet extension* is a COM in-proc server that implements certain interfaces and is registered with Active Directory Domain Services. To create a property sheet extension, perform the following steps.

To create an Active Directory property sheet extension

1. Create the property sheet extension DLL. A property sheet extension is a COM in-proc server that, at a minimum, implements the [IShellExtInit](#) and [IShellPropSheetExt](#) interfaces. For more information, see [Implementing the Property Page COM Object](#).
2. Install the property sheet extension on the computers where the property sheet extension is to be used. To do this, create a Microsoft Windows Installer package for the property sheet extension DLL and deploy the package appropriately using the group policy. For more information, see [Distributing User Interface Components](#).
3. Register the property sheet extension in the Windows registry and with Active Directory Domain Services. For more information, see [Registering the Property Page COM Object in a Display Specifier](#).

Implementing the Property Page COM Object

6/3/2022 • 7 minutes to read • [Edit Online](#)

A property sheet extension is a COM object implemented as an in-proc server. The property sheet extension must implement the [IShellExtInit](#) and [IShellPropSheetExt](#) interfaces. A property sheet extension is instantiated when the user displays the property sheet for an object of a class for which the property sheet extension has been registered in the display specifier of the class.

- [Implementing IShellExtInit](#)
- [Implementing IShellPropSheetExt](#)
- [Passing the Extension Object to the Property Page](#)
- [Working With the Notification Object](#)
- [Miscellaneous](#)
- [Multiple-Selection Property Sheets](#)
- [New with Windows Server 2003](#)
- [Related topics](#)

Implementing IShellExtInit

After the property sheet extension COM object is instantiated, the [IShellExtInit::Initialize](#) method is called. [IShellExtInit::Initialize](#) supplies the property sheet extension with an [IDataObject](#) object that contains data that pertains to the directory object that the property sheet applies.

The [IDataObject](#) contains data in the [CFSTR_DSOBJECTNAMES](#) format. The [CFSTR_DSOBJECTNAMES](#) data format is an HGLOBAL that contains a [DSOBJECTNAMES](#) structure. The [DSOBJECTNAMES](#) structure contains directory object data that the property sheet extension applies.

The [IDataObject](#) also contains data in the [CFSTR_DS_DISPLAY_SPEC_OPTIONS](#) format. The [CFSTR_DS_DISPLAY_SPEC_OPTIONS](#) data format is an HGLOBAL that contains a [DSDISPLAYSPECOPTIONS](#) structure. The [DSDISPLAYSPECOPTIONS](#) contains configuration data for use by the extension.

If any value other than [S_OK](#) is returned from [IShellExtInit::Initialize](#), the property sheet is not displayed.

The *pidlFolder* and *hkeyProgID* parameters of the [IShellExtInit::Initialize](#) method are not used.

The [IDataObject](#) pointer can be saved by the extension by incrementing the reference count of the [IDataObject](#). This interface must be released when no longer required.

Implementing IShellPropSheetExt

After [IShellExtInit::Initialize](#) returns, the [IShellPropSheetExt::AddPages](#) method is called. The property sheet extension must add the page or pages during this method. A property page is created by filling a [PROPSHEETPAGE](#) structure and then passing this structure to the [CreatePropertySheetPage](#) function. The property page is then added to the property sheet by calling the callback function passed to [IShellPropSheetExt::AddPages](#) in the *lpfnAddPage* parameter.

If any value other than [S_OK](#) is returned from [IShellPropSheetExt::AddPages](#), the property sheet is not displayed.

If the property sheet extension is not required to add any pages to the property sheet, it should not call the

callback function passed to [IShellPropSheetExt::AddPages](#) in the *lpfnAddPage* parameter.

The [IShellPropSheetExt::ReplacePage](#) method is not used.

Passing the Extension Object to the Property Page

The property sheet extension object is independent from the property page. In many cases, it is desirable to be able to use the extension object, or some other object, from the property page. To do this, set the **IParam** member of [PROPSHEETPAGE](#) structure to the object pointer. The property page can then retrieve this value when it processes the [WM_INITDIALOG](#) message. For a property page, the *lParam* parameter of the [WM_INITDIALOG](#) message is a pointer to the [PROPSHEETPAGE](#) structure. Retrieve the object pointer by casting the *lParam* of the [WM_INITDIALOG](#) message to a [PROPSHEETPAGE](#) pointer and then retrieving the **IParam** member of the [PROPSHEETPAGE](#) structure.

The following C++ code example shows how to pass an object to a property page.

```
case WM_INITDIALOG:
{
    LPPROPSHEETPAGE pPage = (LPPROPSHEETPAGE)lParam;

    if(NULL != pPage)
    {
        CPropSheetExt *pPropSheetExt;
        pPropSheetExt = (CPropSheetExt*)pPage->lParam;

        if(pPropSheetExt)
        {
            return pPropSheetExt->OnInitDialog(wParam, lParam);
        }
    }
}
break;
```

Be aware that after [IShellPropSheetExt::AddPages](#) is called, the property sheet will release the property sheet extension object and never use it again. This means that the extension object would be deleted before the property page is displayed. When the page attempts to access the object pointer, the memory will have been freed and the pointer will not be valid. To correct this, increment the reference count for the extension object when the page is added and then release the object when the property page dialog is destroyed. This creates another issue because the property page dialog box is not created until the first time the page is displayed. If the user never selects the extension page, the page never gets created and destroyed. This results in the extension object never getting released, so a memory leak occurs. To avoid this, implement a property page callback function. To do this, add the **PSP_USECALLBACK** flag to the **dwFlags** member of the [PROPSHEETPAGE](#) structure and set the **pfnCallback** member of the [PROPSHEETPAGE](#) structure to the address of the [PropSheetPageProc](#) function implemented. When the [PropSheetPageProc](#) function receives the **PSPCB_RELEASE** notification, the *ppsp* parameter of the [PropSheetPageProc](#) contains a pointer to the [PROPSHEETPAGE](#) structure. The **IParam** member of the [PROPSHEETPAGE](#) structure contains the extension pointer which can be used to release the object.

The following C++ code example shows how to release an extension object.

```

UINT CALLBACK CPropSheetExt::PageCallbackProc( HWND hWnd,
                                              UINT uMsg,
                                              LPPROPSHEETPAGE ppsp)
{
    switch(uMsg)
    {
        case PSPCB_CREATE:
            // Must return TRUE to enable the page to be created.
            return TRUE;

        case PSPCB_RELEASE:
        {
            /*
            Release the object. This is called even if the page dialog box was
            never actually created.
            */
            CPropSheetExt *pPropSheetExt = (CPropSheetExt*)ppsp->lParam;

            if(pPropSheetExt)
            {
                pPropSheetExt->Release();
            }
        }
        break;
    }

    return FALSE;
}

```

Working With the Notification Object

Because the property sheet extension pages are displayed within a property sheet created by a component unknown to the extension, it is necessary to use a "manager" to handle the data transfer between the extension pages and the property sheet. This "manager" is called the notification object. The notification object operates as a moderator between the individual pages and the property sheet.

When the property sheet extension object is initialized, the extension must create a notification object by calling [ADsPropCreateNotifyObj](#), passing the [IDataObject](#) obtained from [IShellExtInit::Initialize](#) and the directory object name. It is not necessary to increment the reference count of the [IDataObject](#) interface, because the notification object created by [ADsPropCreateNotifyObj](#) function will do this. The notification object handle provided by [ADsPropCreateNotifyObj](#) should be saved for later use.

[ADsPropCreateNotifyObj](#) can be either called during [IShellExtInit::Initialize](#) or [IShellPropSheetExt::AddPages](#). When the property sheet extension is shut down, it must send a [WM_ADSPROP_NOTIFY_EXIT](#) message to the notification object. This causes the notification object to destroy itself. This is best done when the [PropSheetPageProc](#) function receives the [PSPCB_RELEASE](#) notification.

The property sheet extension can obtain data in addition to that provided by the [CFSTR_DSOBJECTNAMES](#) clipboard format by calling [ADsPropGetInitInfo](#). One of the advantages of using [ADsPropGetInitInfo](#) is that it provides an [IDirectoryObject](#) object used to programmatically work with the directory object.

NOTE

Unlike most COM methods and functions, [ADsPropGetInitInfo](#) does not increment the reference count for the [IDirectoryObject](#) object. The [IDirectoryObject](#) must not be released unless the reference count is manually incremented first.

When the property page is first created, the extension should register the page with the notification object by

calling [ADsPropSetHwnd](#) with the window handle of the page.

[ADsPropCheckIfWritable](#) is a utility function that the property sheet extension can use to determine if a property can be written.

Miscellaneous

The handle of the property page is passed to the page dialog box procedure. The property sheet is the direct parent of the property page, so the handle of the property sheet can be obtained by calling the [GetParent](#) function with the property page handle.

When the contents of the extension page changes, the extension should use the [PropSheet_Changed](#) macro to notify the property sheet of changes. The property sheet will then enable the Apply button.

Multiple-Selection Property Sheets

With Windows Server 2003 and later operating systems, the Active Directory administrative MMC snap-ins support property sheet extensions for multiple directory objects. These property sheets are displayed when the properties are viewed for more than one item at a time. The primary difference between a single-selection property sheet extension and a multiple-selection property sheet extension is that the [DSOBJECTNAMES](#) structure supplied by the [CFSTR_DSOBJECTNAMES](#) clipboard format in [IShellExtInit::Initialize](#) will contain more than one [DSOBJECT](#) structure.

When the notification object is created, a multi-selection property sheet extension must pass a unique name that is provided by the snap-in rather than a name created by the extension. To obtain the unique name, request the [CFSTR_DS_MULTISELECTPROPPAGE](#) clipboard format from the [IDataObject](#) obtained from [IShellExtInit::Initialize](#). This data is an HGLOCAL that contains a null-terminated Unicode string that is the unique name. This unique name is then passed to the [ADsPropCreateNotifyObj](#) function to create the notification object. The [CreateADsNotificationObject](#) example function in [Example Code for Implementation of the Property Sheet COM Object](#) demonstrates how to do this correctly, as well as being compatible with earlier versions of the snap-in that do not support multi-selection property sheets.

For multiple-selection property sheets, the system only binds to the first object in the [DSOBJECT](#) array. Because of this, [ADsPropGetInitInfo](#) only supplies the [IDirectoryObject](#) and writeable attributes for the first object in the array. The other objects in the array are not bound to.

A multiple-selection property sheet extension is registered under the [adminMultiselectPropertyPages](#) attribute.

New with Windows Server 2003

The following features are new with Windows Server 2003.

If the property page encounters an error, [ADsPropSendErrorMessage](#) can be called with the appropriate error data. [ADsPropSendErrorMessage](#) will store all error messages in a queue. These messages will be displayed the next time [ADsPropShowErrorDialog](#) is called. When [ADsPropShowErrorDialog](#) returns, the queued messages are deleted.

Windows Server 2003 introduces the [ADsPropSetHwndWithTitle](#) function. This function is similar to [ADsPropSetHwnd](#), but includes the page title. This enables the error dialog box displayed by [ADsPropShowErrorDialog](#) to provide more useful data to the user. If the property sheet extension uses the [ADsPropShowErrorDialog](#) function, the extension should use [ADsPropSetHwndWithTitle](#) rather than [ADsPropSetHwnd](#).

Related topics

Example Code for Implementation of the Property Sheet COM Object

Example Code for Implementation of the Property Sheet COM Object

6/3/2022 • 3 minutes to read • [Edit Online](#)

The following code example can be used to implement an Active Directory property sheet extension.

IShellExtInit Implementation

The following C++ code example can be used to implement the **IShellExtInit** methods.

```
*****  
CPropSheetExt::Initialize()  
*****  
  
STDMETHODIMP CPropSheetExt::Initialize( LPCITEMIDLIST pidlFolder,  
                                      IDataObject *pDataObj,  
                                      HKEY hKeyProgId)  
{  
    STGMEDIUM    stm;  
    FORMATETC    fe;  
    HRESULT      hr = S_OK;  
  
    if(NULL == pDataObj)  
    {  
        return E_INVALIDARG;  
    }  
  
    hr = pDataObj->QueryInterface(IID_IDataObject, (LPVOID*)&m_pdo);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    fe.cfFormat = RegisterClipboardFormat(CFSTR_DSOBJECTNAMES);  
    fe.ptd = NULL;  
    fe.dwAspect = DVASPECT_CONTENT;  
    fe.lindex = -1;  
    fe.tymed = TYMED_HGLOBAL;  
    hr = m_pdo->GetData(&fe, &stm);  
    if(SUCCEEDED(hr))  
    {  
        LPDSOBJECTNAMES pdson = (LPDSOBJECTNAMES)GlobalLock(stm.hGlobal);  
  
        if(pdson)  
        {  
            LPWSTR pwszName = (LPWSTR)((LPBYTE)pdson + pdson->aObjects[0].offsetName);  
  
            m_pwszADsPath = (LPWSTR)GlobalAlloc(GPTR, (lstrlenW(pwszName) + 1) * sizeof(WCHAR));  
            if(m_pwszADsPath)  
            {  
                wcscpy_s(m_pwszADsPath, pwszName);  
            }  
  
            LPWSTR pwszClass = (LPWSTR)((LPBYTE)pdson + pdson->aObjects[0].offsetClass);  
            pwszClass = NULL;  
  
            GlobalUnlock(stm.hGlobal);  
        }  
    }  
}
```

```

        ReleaseStgMedium(&stm);
    }

    m_hwndNotifyObj = CreateADsNotificationObject(pDataObj);

    if(m_hwndNotifyObj)
    {
        ADSPROPINITPARAMS    InitParams;

        hr = GetADsPageInfo(m_hwndNotifyObj, &InitParams);
        if(SUCCEEDED(hr))
        {
            if(InitParams.pDsObj)
            {
                hr = InitParams.pDsObj->QueryInterface( IID_IDirectoryObject,
                                                (LPVOID* ) & m_pDirObj);
            }
        }
    }

    fe.cfFormat = RegisterClipboardFormat(CFSTR_DS_DISPLAY_SPEC_OPTIONS);
    fe.ptd = NULL;
    fe.dwAspect = DVASPECT_CONTENT;
    fe.lindex = -1;
    fe.tymed = TYMED_HGLOBAL;
    hr = pDataObj->GetData(&fe, &stm);
    if(SUCCEEDED(hr))
    {
        PDSDISPLAYSPECOPTIONS    pdso;

        pdso = (PDSDISPLAYSPECOPTIONS)GlobalLock(stm.hGlobal);
        if(pdso)
        {
            DWORD    dwBytes = GlobalSize(stm.hGlobal);

            m_pDispSpecOpts = (PDSDISPLAYSPECOPTIONS)GlobalAlloc(GPTR, dwBytes);
            if(m_pDispSpecOpts)
            {
                CopyMemory(m_pDispSpecOpts, pdso, dwBytes);
            }

            GlobalUnlock(stm.hGlobal);
        }

        ReleaseStgMedium(&stm);
    }

    return hr;
}

```

IShellPropSheetExt Implementation

The following C++ code example can be used to implement the [IShellPropSheetExt](#) methods.

```

*****CPropSheetExt::AddPages()

*****STDMETHODIMP CPropSheetExt::AddPages( LPFNADDPROPSHEETPAGE pfnAddPage,
                                         LPARAM lParam)

{
    PROPSHEETPAGE psp;
    HPROPSHEETPAGE hPage;

    psp.dwSize      = sizeof(psp);
    psp.dwFlags     = PSP_USETITLE | PSP_USECALLBACK;
    psp.hInstance   = g_hInst;
    psp.pszTemplate = MAKEINTRESOURCE(IDD_PAGEDLG);
    psp.hIcon       = 0;
    psp.pszTitle    = g_szPageTitle;
    psp.pfnDlgProc  = (DLGPROC)PageDlgProc;
    psp.pcRefParent = NULL;
    psp.pfnCallback = PageCallbackProc;
    //pass the object pointer to the dialog box
    psp.lParam      = (LPARAM)this;

    hPage = CreatePropertySheetPage(&psp);

    if(hPage)
    {
        if(pfnAddPage(hPage, lParam))
        {
            /*
            Maintain this object until the page is released in
            PageCallbackProc.
            */
            this->AddRef();
            return S_OK;
        }
        else
        {
            DestroyPropertySheetPage(hPage);
        }
    }
    else
    {
        return E_OUTOFMEMORY;
    }

    return E_FAIL;
}

*****CPropSheetExt::ReplacePage()

*****STDMETHODIMP CPropSheetExt::ReplacePage( UINT uPageID,
                                              LPFNADDPROPSHEETPAGE lpfnAddPage,
                                              LPARAM lParam)

{
    return E_NOTIMPL;
}

```

Miscellaneous Implementation

The following C++ code example can be used to implement the utility methods and functions used in the previous code example.

```
HWND CreateADsNotificationObject(IDataObject *pDataObject)
{
    STGMEDIUM    stm;
    FORMATETC    fe;
    HRESULT      hr;
    HWND         hwndNotifyObject = NULL;

    /*
    The "DsAdminMultiSelectClipFormat" clipboard format is supported by the
    data object if the Active Directory Users and Computers snap-in supports
    multi-selection property sheets.
    */
    fe.cfFormat = RegisterClipboardFormat(TEXT("DsAdminMultiSelectClipFormat"));
    fe.ptd = NULL;
    fe.dwAspect = DVASPECT_CONTENT;
    fe.lindex = -1;
    fe.tymed = TYMED_HGLOBAL;
    hr = pDataObject->GetData(&fe, &stm);
    if (SUCCEEDED(hr))
    {
        PWSTR pwszUniqueID = (LPWSTR)GlobalLock(stm.hGlobal);

        if (pwszUniqueID)
        {
            hr = ADsPropCreateNotifyObj(pDataObject, pwszUniqueID, &hwndNotifyObject);

            GlobalUnlock(stm.hGlobal);
        }

        ReleaseStgMedium(&stm);
    }
    else
    {
        fe.cfFormat = RegisterClipboardFormat(CFSTR_DSOBJECTNAMES);
        fe.ptd = NULL;
        fe.dwAspect = DVASPECT_CONTENT;
        fe.lindex = -1;
        fe.tymed = TYMED_HGLOBAL;
        hr = pDataObject->GetData(&fe, &stm);
        if(SUCCEEDED(hr))
        {
            LPDSOBJECTNAMES pdson = (LPDSOBJECTNAMES)GlobalLock(stm.hGlobal);

            if(pdson)
            {
                LPWSTR pwszName = ((LPBYTE)pdson + pdson->aObjects[0].offsetName);

                hr = ADsPropCreateNotifyObj(pDataObject, pwszName, &hwndNotifyObject);

                GlobalUnlock(stm.hGlobal);
            }

            ReleaseStgMedium(&stm);
        }
    }

    return hwndNotifyObject;
}

*****  

GetADsPageInfo()  

*****/
```

```

HRESULT GetADSPageInfo(HWND hwndNotifyObject, ADSPROPINITPARAMS *pip)
{
    if(!IsWindow(hwndNotifyObject))
    {
        return E_INVALIDARG;
    }

    ADSPROPINITPARAMS InitParams;

    InitParams.dwSize = sizeof(ADSPROPINITPARAMS);
    if(ADsPropGetInitInfo(hwndNotifyObject, &InitParams))
    {
        *pip = InitParams;

        return InitParams.hr;
    }

    return E_FAIL;
}

//****************************************************************************

CPropSheetExt::PageDlgProc

***** */

#define THIS_POINTER_PROP TEXT("ThisPointerProperty")

BOOL CALLBACK CPropSheetExt::PageDlgProc(    HWND hWnd,
                                             UINT uMsg,
                                             WPARAM wParam,
                                             LPARAM lParam)
{
    // Get the object pointer.
    CPropSheetExt *pExt = (CPropSheetExt*)GetProp(hWnd, THIS_POINTER_PROP);

    switch(uMsg)
    {
    case WM_INITDIALOG:
        {
            /*
            Get the pointer to the object. This is contained in the LPARAM of
            the PROPSHEETPAGE structure.
            */
            LPPROPSHEETPAGE pPage = (LPPROPSHEETPAGE)lParam;

            if(pPage)
            {
                pExt = (CPropSheetExt*)pPage->lParam;

                if(pExt)
                {
                    // Set the page window handle in the object.
                    pExt->m_hwndPage = hWnd;

                    // Store the object pointer with this particular page dialog.
                    SetProp(hWnd, THIS_POINTER_PROP, (HANDLE)pExt);

                    ADsPropSetHwnd(pExt->m_hwndNotifyObj, hWnd, g_szPageTitle);

                    // Forward the message to the message handler.
                    return pExt->OnInitDialog(wParam, lParam);
                }
            }
        }
        break;

    case WM_NOTIFY:
        ...
    }
}

```

```

    if(pExt)
    {
        // Forward the message to the message handler.
        return pExt->OnNotify(wParam, lParam);
    }
    break;

case WM_COMMAND:
    if(pExt)
    {
        // Forward the message to the message handler.
        return pExt->OnCommand(wParam, lParam);
    }
    return FALSE;

case WM_DESTROY:
    if(pExt)
    {
        // Forward the message to the message handler.
        pExt->OnDestroy();
    }

    // Remove the property from the page.
    RemoveProp(hWnd, THIS_POINTER_PROP);
    break;
}

return FALSE;
}

```

CPropSheetExt::PageCallbackProc()

This function is called when the page is created and released. This is even called if the page is never actually displayed.

```

*****  

UINT CALLBACK CPropSheetExt::PageCallbackProc(  HWND hWnd,
                                                UINT uMsg,
                                                LPPROPSHEETPAGE ppsp)
{
    switch(uMsg)
    {
        case PSPCB_CREATE:
            // Must return TRUE to create the page.
            return TRUE;

        case PSPCB_RELEASE:
            {
                /*
                Release the object. This is called even if the page dialog was
                never actually created.
                */
                CPropSheetExt *pPropSheetExt = (CPropSheetExt*)ppsp->lParam;

                if(pPropSheetExt)
                {
                    if(IsWindow(pPropSheetExt->m_hwndNotifyObj))
                    {
                        SendMessage(pPropSheetExt->m_hwndNotifyObj,
                                   WM_ADSPROP_NOTIFY_EXIT,
                                   0,
                                   0);

                        pPropSheetExt->m_hwndNotifyObj = NULL;
                    }
                }
            }
    }
}
```

```
    pPropSheetExt->Release();
}
break;
}

return FALSE;
}
```

Registering the Property Page COM Object in a Display Specifier

6/3/2022 • 10 minutes to read • [Edit Online](#)

When you use COM to create a property sheet extension DLL for Active Directory Domain Services, you must also register the extension with the Windows registry and Active Directory Domain Services. Registering the extension enables the Active Directory administrative MMC snap-ins and the Windows shell to recognize the extension.

Registering in the Windows Registry

Like all COM servers, a property sheet extension must be registered in the Windows registry. The extension is registered under the following key.

```
HKEY_CLASSES_ROOT  
    CLSID  
        <clsid>
```

<clsid> is the string representation of the CLSID as produced by the [StringFromCLSID](#) function. Under the <clsid> key, there is an **InProcServer32** key that identifies the object as a 32-bit in-proc server. Under the **InProcServer32** key, the location of the DLL is specified in the default value and the threading model is specified in the **ThreadingModel** value. All property sheet extensions must use the "Apartment" threading model.

Registering with Active Directory Domain Services

Property sheet extension registration is specific to one locale. If the property sheet extension applies to all locales, it must be registered in the object class **displaySpecifier** object in all of the locale subcontainers in the Display Specifiers container. If the property sheet extension is localized for a certain locale, register it in the **displaySpecifier** object in that locale subcontainer. For more information about the Display Specifiers container and locales, see [Display Specifiers](#) and [DisplaySpecifiers Container](#).

There are three display specifier attributes that a property sheet extension can be registered under. These are **adminPropertyPages**, **shellPropertyPages**, and **adminMultiselectPropertyPages**.

The **adminPropertyPages** attribute identifies administrative property pages to display in Active Directory administrative snap-ins. The property page appears when the user views properties for objects of the appropriate class in one of the Active Directory administrative MMC snap-ins.

The **shellPropertyPages** attribute identifies end-user property pages to display in the Windows shell. The property page appears when the user views the properties for objects of the appropriate class in the Windows Explorer. Beginning with the Windows Server 2003 operating systems, the Windows shell no longer displays objects from Active Directory Domain Services.

The **adminMultiselectPropertyPages** is only available on the Windows Server 2003 operating systems. The property page appears when the user views properties for more than one object of the appropriate class in one of the Active Directory administrative MMC snap-ins.

All of these attributes are multi-valued.

The values for the **adminPropertyPages** and **shellPropertyPages** attributes require the following format.

```
<order number>,<clsid>,<optional data>
```

The values for the [adminMultiselectPropertyPages](#) attribute require the following format.

```
<order number>,<clsid>
```

The "<order number>" is an unsigned number that represents the page position in the sheet. When a property sheet is displayed, the values are sorted using a comparison of each value's "<order number>". If more than one value has the same "<order number>", those property page COM objects are loaded in the order they are read from the Active Directory server. If possible, you should use a non-existing "<order number>"; that is, one not used by other values in the property. There is no prescribed starting position, and gaps are allowed in the "<order number>" sequence.

The "<clsid>" is the string representation of the CLSID as produced by the [StringFromCLSID](#) function.

The "<optional data>" is a string value that is not required. This value can be retrieved by the property page COM object using the [IDataObject](#) pointer passed to its [IShellExtInit::Initialize](#) method. The property page COM object obtains this data by calling [IDataObject::GetData](#) with the [CFSTR_DSPROPERTYPAGEINFO](#) clipboard format. This provides an [HGLOBAL](#) that contains a [DSPROPERTYPAGEINFO](#) structure. The [DSPROPERTYPAGEINFO](#) structure contains a Unicode string that contains the "<optional data>". The "<optional data>" is not allowed with the [adminMultiselectPropertyPages](#) attribute. The following C/C++ code example shows how to retrieve the "<optional data>".

```
fe.cfFormat = RegisterClipboardFormat(CFSTR_DSPROPERTYPAGEINFO);
fe.ptd = NULL;
fe.dwAspect = DVASPECT_CONTENT;
fe.lindex = -1;
fe.tymed = TYMED_HGLOBAL;
hr = pDataObj->GetData(&fe, &stm);
if(SUCCEEDED(hr))
{
    DSPROPERTYPAGEINFO *pPageInfo;

    pPageInfo = (DSPROPERTYPAGEINFO*)GlobalLock(stm.hGlobal);
    if(pPageInfo)
    {
        LPWSTR pwszData;

        pwszData = (LPWSTR)((LPBYTE)pPageInfo + pPageInfo->offsetString);
        pwszData = NULL;

        GlobalUnlock(stm.hGlobal);
    }

    ReleaseStgMedium(&stm);
}
```

A property sheet extension can implement more than one property page; one possible use of the "<optional data>" is to name the pages to display. This provides the flexibility of choosing to implement multiple COM objects, one for each page, or a single COM object to handle multiple pages.

The following code example is an example value that can be used for the [adminPropertyPages](#), [shellPropertyPages](#), or [adminMultiselectPropertyPages](#) attributes.

```
1,{6dfe6485-a212-11d0-bcd5-00c04fd8d5b6}
```

IMPORTANT

For the Windows shell, display specifier data is retrieved at user logon, and cached for the user session. For administrative snap-ins, the display specifier data is retrieved when the snap-in is loaded, and is cached for the lifetime of the process. For the Windows shell, this indicates that changes to display specifiers take effect after a user logs off and then logs on again. For the administrative snap-ins, changes take effect when the snap-in or console file is loaded.

Adding a Value to the Property Sheet Extension Attributes

The following procedure describes how to register a property sheet extension under one of the property sheet extension attributes.

Registering a property sheet extension under one of the property sheet extension attributes

1. Ensure that the extension does not already exist in the attribute values.
2. Add the new value at the end of the property page ordering list. That is set the "<order number>" portion of the attribute value to the next value after the highest existing order number.
3. The [IADs::PutEx](#) method is used to add the new value to the attribute. The *InControlCode* parameter must be set to **ADS_PROPERTY_APPEND** so that the new value is appended to the existing values and does not, therefore, overwrite the existing values. The [IADs::SetInfo](#) method must be called afterward to commit the change to the directory.

Be aware that the same property sheet extension can be registered for more than one object class.

The [IADs::PutEx](#) method is used to add the new value to the attribute. The *InControlCode* parameter must be set to **ADS_PROPERTY_APPEND**, so that the new value is appended to the existing values and does not, therefore, overwrite the existing values. The [IADs::SetInfo](#) method must be called after to commit the change to the directory.

The following code example adds a property sheet extension to the group class in the computer's default locale. Be aware that the [AddPropertyPageToDisplaySpecifier](#) function verifies the property sheet extension CLSID in the existing values, gets the highest order number, and adds the value for the property page using [IADs::PutEx](#) with the **ADS_PROPERTY_APPEND** control code.

```
// Add msrvct.dll to the project.  
// Add activeds.lib to the project.  
// Add adsiid.lib to the project.  
  
#include "stdafx.h"  
#include <wchar.h>  
#include <objbase.h>  
#include <activeds.h>  
#include "atibase.h"  
  
HRESULT AddPropertyPageToDisplaySpecifier(LPOLESTR szClassName, // ldapDisplayName of the class.  
                                         CLSID *pPropPageCLSID // CLSID of property page COM object.  
                                         );  
  
HRESULT BindToDisplaySpecifiersContainerByLocale(LCID *locale,  
                                                IADsContainer **ppDispSpecCont  
                                                );  
  
HRESULT GetDisplaySpecifier(IADsContainer *pContainer, LPOLESTR szDispSpec, IADs **ppObject);  
  
// Entry point for the application.  
void wmain(int argc, wchar_t *argv[ ]) {
```

```

wprintf(L"This program adds a sample property page to the display specifier for group class in the local
computer's default locale.\n");

// Initialize COM.
CoInitialize(NULL);
HRESULT hr = S_OK;

// Class ID for the sample property page.
LPOLESTR szCLSID = L"{D9FCE809-8A10-11d2-A7E7-00C04F79DC0F}";
LPOLESTR szClass = L"group";
CLSID clsid;
// Convert to GUID.
hr = CLSIDFromString(
    szCLSID, // Pointer to the string representation of the CLSID.
    &clsid // Pointer to the CLSID.
);

hr = AddPropertyPageToDisplaySpecifier(szClass, // ldapDisplayName of the class.
                                      &clsid // CLSID of property page COM object.
                                     );
if (S_OK == hr)
    wprintf(L"Property page registered successfully\n");
else if (S_FALSE == hr)
    wprintf(L"Property page was not added because it was already registered.\n");
else
    wprintf(L"Property page was not added. HR: %x.\n");

// Uninitialize COM.
CoUninitialize();
return;
}

// Adds a property page to Active Directory admin snap-ins.
HRESULT AddPropertyPageToDisplaySpecifier(LPOLESTR szClassName, // ldapDisplayName of class.
                                         CLSID *pPropPageCLSID // CLSID of property page COM object.
                                         )
{
    HRESULT hr = E_FAIL;
    IADsContainer *pContainer = NULL;
    LPOLESTR szDispSpec = new OLECHAR[MAX_PATH];
    IADs *pObject = NULL;
    VARIANT var;
    CComBSTR sbstrProperty = L"adminPropertyPages";
    LCID locale = NULL;
    // Get the display specifier container using default system locale.
    // When adding a property page COM object, specify the locale
    // because the registration is locale-specific.
    // This means if you created a property page
    // for German, explicitly add it to the 407 container,
    // so that it will be used when a computer is running with locale
    // set to German and not the locale set on the
    // computer where this application is running.
    hr = BindToDisplaySpecifiersContainerByLocale(&locale,
                                                 &pContainer
                                                );
    // Handle fail case where dispsspec object
    // is not found and give option to create one.

    if (SUCCEEDED(hr))
    {
        // Bind to display specifier object for the specified class.
        // Build the display specifier name.
#ifdef _MBCS
        wcscpy_s(szDispSpec, szClassName);
        wcscat_s(szDispSpec, L"-Display");
        hr = GetDisplaySpecifier(pContainer, szDispSpec, &pObject);
#endif _MBCS#endif _MBCS
        if (SUCCEEDED(hr))

```

```

{
    // Convert GUID to string.
    LPOLESTR szDSGUID = new WCHAR [39];
    ::StringFromGUID2(*pPropPageCLSID, szDSGUID, 39);

    // Get the adminPropertyPages property.
    hr = pObject->GetEx(sbstrProperty, &var);
    if (SUCCEEDED(hr))
    {
        LONG lstart, lend;
        SAFEARRAY *sa = V_ARRAY(&var);
        VARIANT varItem;
        // Get the lower and upper bound.
        hr = SafeArrayGetLBound(sa, 1, &lstart);
        if (SUCCEEDED(hr))
        {
            hr = SafeArrayGetUBound(sa, 1, &lend);
        }
        if (SUCCEEDED(hr))
        {
            // Iterate the values to verify if the prop page CLSID
            // is already registered.
            VariantInit(&varItem);
            BOOL bExists = FALSE;
            UINT uiLastItem = 0;
            UINT uiTemp = 0;
            INT iOffset = 0;
            LPOLESTR szMainStr = new OLECHAR[MAX_PATH];
            LPOLESTR szItem = new OLECHAR[MAX_PATH];
            LPOLESTR szStr = NULL;
            for (long idx=lstart; idx <= lend; idx++)
            {
                hr = SafeArrayGetElement(sa, &idx, &varItem);
                if (SUCCEEDED(hr))
                {
                    // Verify that the specified CLSID is already registered.
                    wcscpy_s(szMainStr,varItem.bstrVal);
                    if (wcsstr(szMainStr,szDSGUID))
                        bExists = TRUE;
                    // Get the index which is the number before the first comma.
                    szStr = wcschr(szMainStr, ',');
                    iOffset = (int)(szStr - szMainStr);
                    wcsncpy_s(szItem, szMainStr, iOffset);
                    szItem[iOffset]=0L;
                    uiTemp = _wtoi(szItem);
                    if (uiTemp > uiLastItem)
                        uiLastItem = uiTemp;
                    VariantClear(&varItem);
                }
            }
            // If the CLSID is not registered, add it.
            if (!bExists)
            {
                // Build the value to add.
                LPOLESTR szValue = new OLECHAR[MAX_PATH];
                // Next index to add at end of list.

#ifdef _MBCS
                uiLastItem++;
                _itow_s(uiLastItem, szValue, 10);
                wcscat_s(szValue,L",");
                // Add the class ID for the property page.
                wcscat_s(szValue,szDSGUID);
                wprintf(L"Value to add: %s\n", szValue);
#endif _MBCS
                VARIANT varAdd;
                // Only one value to add

```

```

        LPOLESTR pszAddStr[1];
        pszAddStr[0]=szValue;
        ADsBuildVarArrayStr(pszAddStr, 1, &varAdd);

        hr = pObject->PutEx(ADS_PROPERTY_APPEND, sbstrProperty, varAdd);
        if (SUCCEEDED(hr))
        {
            // Commit the change.
            hr = pObject->SetInfo();
        }
        else
            hr = S_FALSE;
    }
}
VariantClear(&var);
}
if (pObject)
    pObject->Release();
}

return hr;
}

// This function returns a pointer to the display specifier container
// for the specified locale.
// If locale is NULL, use the default system locale and then return the locale in the locale param.
HRESULT BindToDisplaySpecifiersContainerByLocale(LCID *locale,
                                                IADsContainer **ppDispSpecCont
                                                )
{
    HRESULT hr = E_FAIL;

    if ((!ppDispSpecCont)||(!locale))
        return E_POINTER;

    // If no locale is specified, use the default system locale.
    if (!(*locale))
    {
        *locale = GetSystemDefaultLCID();
        if (!(*locale))
            return E_FAIL;
    }

    // Verify that it is a valid locale.
    if (!IsValidLocale(*locale, LCID_SUPPORTED))
        return E_INVALIDARG;

LPOLESTR szPath = new OLECHAR[MAX_PATH*2];
IADs *pObj = NULL;
VARIANT var;

hr = ADsOpenObject(L"LDAP://rootDSE",
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                    IID_IADs,
                    (void**)&pObj);

if (SUCCEEDED(hr))
{
    // Get the DN to the config container.
    hr = pObj->Get(CComBSTR("configurationNamingContext"), &var);
    if (SUCCEEDED(hr))
    {
#endif _MBCS
        // Build the string to bind to the DisplaySpecifiers container.
        swprintf_s(szPath,L"LDAP://cn=%x,cn=DisplaySpecifiers,%s", *locale, var.bstrVal);
        // Bind to the DisplaySpecifiers container.
}
}

```

```

        *ppDispSpecCont = NULL;
        hr = ADsOpenObject(szPath,
                           NULL,
                           NULL,
                           ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                           IID_IADsContainer,
                           (void**)ppDispSpecCont);

#endif _MBCS

        if(FAILED(hr))
        {
            if (*ppDispSpecCont)
            {
                (*ppDispSpecCont)->Release();
                (*ppDispSpecCont) = NULL;
            }
        }
    }

// Cleanup.
VariantClear(&var);
if (pObj)
    pObj->Release();

return hr;
}

HRESULT GetDisplaySpecifier(IADsContainer *pContainer, LPOLESTR szDispSpec, IADs **ppObject)
{
    HRESULT hr = E_FAIL;
    CComBSTR sbstrDSPPath;
    IDispatch *pDisp = NULL;

    if (!pContainer)
    {
        hr = E_POINTER;
        return hr;
    }

    // Verify other pointers.
    // Initialize the output pointer.
    (*ppObject) = NULL;

    // Build relative path to the display specifier object.
    sbstrDSPPath = "CN=";
    sbstrDSPPath += szDispSpec;

    // Use child object binding with IADsContainer::GetObject.
    hr = pContainer->GetObject(CComBSTR("displaySpecifier"),
                                 sbstrDSPPath,
                                 &pDisp);
    if (SUCCEEDED(hr))
    {
        hr = pDisp->QueryInterface(IID_IADs, (void**)ppObject);
        if (FAILED(hr))
        {
            // Clean up.
            if (*ppObject)
                (*ppObject)->Release();
        }
    }

    if (pDisp)
        pDisp->Release();

    return hr;
}

```


Context Menus for Use with Display Specifiers

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory administrative MMC snap-ins and Windows 2000 shell provide a mechanism to add an item to the context menu displayed for objects in Active Directory Domain Services. A context menu item can be added by implementing an COM in-proc server known as a *context menu extension*. A context menu item can also be added that invokes any file started with the [ShellExecute](#) API, such as an application or webpage URL. This is known as a *static context menu item*.

Developer Audience

This documentation assumes that the reader is familiar with COM operation and component development using C++. It is not currently possible to create an Active Directory Domain Services context menu extension using Microsoft Visual Basic.

Extending the Context Menu With a Context Menu Extension

A context menu extension is a COM in-proc server that implements certain interfaces and is registered with Active Directory Domain Services.

To create and install a context menu extension

1. Create the context menu extension DLL. A context menu extension is a COM in-proc server that, at a minimum, implements the [IShellExtInit](#) and [IContextMenu](#) interfaces. For more information, see [Implementing the Context Menu COM Object](#).
2. Install the context menu sheet extension on computers where the context menu extension is used. This is accomplished by creating a Microsoft Windows Installer package for the context menu extension DLL and deploying the package appropriately using the group policy. For more information, see [Distributing User Interface Components](#).
3. Register the context menu extension in the Windows registry and with Active Directory Domain Services. For more information, see [Registering the Context Menu COM Object in a Display Specifier](#).

Extending the Context Menu With a Static Context Menu Item

A static context menu item can be used to invoke any file started with the [ShellExecute](#) API, such as an application or webpage URL. To accomplish this, the static context menu item for a particular object class must be registered so that the static context menu item is added to the context menu of objects of that class. For more information, see [Registering a Static Context Menu Item](#).

Implementing the Context Menu COM Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

A context menu extension is a COM object implemented as an in-proc server. The context menu extension must implement the [IShellExtInit](#) and [IContextMenu](#) interfaces. A context menu extension is instantiated when the user displays the context menu for an object of a class for which the context menu extension has been registered.

Implementing IShellExtInit

After the context menu extension COM object is instantiated, the [IShellExtInit::Initialize](#) method is called. [IShellExtInit::Initialize](#) supplies the context menu extension with an [IDataObject](#) object that contains data pertinent to the directory object that the context menu applies to.

The [IDataObject](#) contains data in the [CFSTR_DSOBJECTNAMES](#) format. The [CFSTR_DSOBJECTNAMES](#) data format is an HGLOBAL that contains a [DSOBJECTNAMES](#) structure. The [DSOBJECTNAMES](#) structure contains data about the directory object that the property sheet extension applies to.

The [IDataObject](#) also contains data in the [CFSTR_DS_DISPLAY_SPEC_OPTIONS](#) format. The [CFSTR_DS_DISPLAY_SPEC_OPTIONS](#) data format is an HGLOBAL that contains a [DSDISPLAYSPECOPTIONS](#) structure. The [DSDISPLAYSPECOPTIONS](#) contains configuration data for use by the extension.

If any value other than [S_OK](#) is returned from [IShellExtInit::Initialize](#), the context menu extension will not be used.

The *pidlFolder* and *hkeyProgID* parameters of the [IShellExtInit::Initialize](#) method are not used.

Implementing IContextMenu

After [IShellExtInit::Initialize](#) returns, the [IContextMenu::QueryContextMenu](#) method is called to obtain the menu item or items that the context menu extension will add. The [QueryContextMenu](#) implementation is fairly straightforward. The context menu extension adds its menu items using the [InsertMenuItem](#) or similar function. The menu command identifiers must be greater than or equal to *idCmdFirst* and must be less than *idCmdLast*. [QueryContextMenu](#) must return the greatest numeric identifier added to the menu plus one. The best way to assign menu command identifiers is to start at zero and work up in sequence. If the context menu extension does not need to any menu items, it should simply not add any items to the menu and return zero from [QueryContextMenu](#).

[IContextMenu::GetCommandString](#) is called to retrieve textual data for the menu item, such as help text to be displayed for the menu item. It is possible that the context menu host will use Unicode strings while the extension uses ANSI strings. Because of this, the [GCS_HELPTEXTA](#), [GCS_HELPTEXTW](#), [GCS_VERBA](#) and [GCS_VERBW](#) cases must be handled individually. Implementation of this method is optional.

[IContextMenu::InvokeCommand](#) is called when one of the menu items installed by the context menu extension is selected. The context menu performs or initiates the desired actions in response to this method.

Example Code for Implementation of the Context Menu COM Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example can be used to implement an Active Directory context menu extension. The single context menu item will display a message box that contains the ADsPath of each item selected.

IShellExtInit Implementation

The following code example can be used to implement the **IShellExtInit** methods.

```
////////////////////////////////////////////////////////////////
//
// IShellExtInit Implementation
//
//*****CContMenuExt::Initialize()

*****STDMETHODIMP CContMenuExt::Initialize(  LPCITEMIDLIST pidlFolder,
                                             LPDATAOBJECT pDataObj,
                                             HKEY hKeyProgId)
{
    STGMEDIUM    stm;
    FORMATETC    fe;
    HRESULT      hr;

    if(NULL == pDataObj)
    {
        return E_INVALIDARG;
    }

    fe.cfFormat = RegisterClipboardFormat(CFSTR_DSOBJECTNAMES);
    fe.ptd = NULL;
    fe.dwAspect = DVASPECT_CONTENT;
    fe.lindex = -1;
    fe.tymed = TYMED_HGLOBAL;
    hr = pDataObj->GetData(&fe, &stm);
    if(SUCCEEDED(hr))
    {
        LPDSOBJECTNAMES pdson = (LPDSOBJECTNAMES)GlobalLock(stm.hGlobal);

        if(pdson)
        {
            DWORD    dwBytes = GlobalSize(stm.hGlobal);

            m_pObjectName = (DSOBJECTNAMES*)GlobalAlloc(GPTR, dwBytes);
            if(m_pObjectName)
            {
                CopyMemory(m_pObjectName, pdson, dwBytes);
            }

            GlobalUnlock(stm.hGlobal);
        }
        ReleaseStgMedium(&stm);
    }
}
```

```

        }

        fe.cfFormat = RegisterClipboardFormat(CFSTR_DS_DISPLAY_SPEC_OPTIONS);
        fe.ptd = NULL;
        fe.dwAspect = DVASPECT_CONTENT;
        fe.lindex = -1;
        fe.tymed = TYMED_HGLOBAL;
        hr = pDataObj->GetData(&fe, &stm);
        if(SUCCEEDED(hr))
        {
            PDSDISPLAYSPECOPTIONS pdso;

            pdso = (PDSDISPLAYSPECOPTIONS)GlobalLock(stm.hGlobal);
            if(pdso)
            {
                DWORD dwBytes = GlobalSize(stm.hGlobal);

                m_pDispSpecOpts = (PDSDISPLAYSPECOPTIONS)GlobalAlloc(GPTR, dwBytes);
                if(m_pDispSpecOpts)
                {
                    CopyMemory(m_pDispSpecOpts, pdso, dwBytes);
                }

                GlobalUnlock(stm.hGlobal);
            }

            ReleaseStgMedium(&stm);
        }

        return hr;
    }
}

```

IContextMenu Implementation

The following code example can be used to implement the [IContextMenu](#) methods.

```

///////////
// IContextMenu Implementation
//

*****CContextMenuExt::QueryContextMenu()

*****STDMETHODIMP CContextMenuExt::QueryContextMenu(      HMENU hMenu,
                                                       UINT indexMenu,
                                                       UINT idCmdFirst,
                                                       UINT idCmdLast,
                                                       UINT uFlags)

{
    if(m_pObjectNames && !(CMF_DEFAULTONLY & uFlags))
    {
        InsertMenu( hMenu,
                   indexMenu,
                   MF_STRING | MF_BYPOSITION,
                   idCmdFirst + IDM_CONTMENU,
                   TEXT("AD Sample Context Menu"));

        return MAKE_HRESULT(SEVERITY_SUCCESS, 0, USHORT(IDM_CONTMENU + 1));
    }

    return MAKE_HRESULT(SEVERITY_SUCCESS, 0, USHORT(0));
}

```

```

}

/***********************/

CContMenuExt::InvokeCommand()

***********************/

STDMETHODIMP CContMenuExt::InvokeCommand(LPCMINKOECOMMANDINFO lpcmi)
{
    if(LOWORD(lpcmi->lpVerb) > IDM_CONTMENU)
    {
        return E_INVALIDARG;
    }

    switch (LOWORD(lpcmi->lpVerb))
    {
    case IDM_CONTMENU:
        {
            LPWSTR pwszName;
            DWORD dwBytes;
            UINT i;

            for(i = 0, dwBytes = 0; i < m_pObjectNames->cItems; i++)
            {
                pwszName = (LPWSTR)((LPBYTE)m_pObjectNames + m_pObjectNames->aObjects[i].offsetName);
                dwBytes += (lstrlenW(pwszName) + 2) * sizeof(WCHAR);
            }

            LPWSTR pwszText = (LPWSTR)GlobalAlloc(GPTR, dwBytes);

            if(pwszText)
            {
                pwszText[0] = 0;
                for(i = 0; i < m_pObjectNames->cItems; i++)
                {
                    pwszName = (LPWSTR)((LPBYTE)m_pObjectNames + m_pObjectNames->aObjects[i].offsetName);
                    wcscat_s(pwszText, pwszName);
                    wcscat_s(pwszText, L"\n");
                }

                MessageBoxW(lpcmi->hwnd,
                           pwszText,
                           L"Sample Command",
                           MB_OK | MB_ICONINFORMATION);

                GlobalFree(pwszText);
            }
        }
        break;
    }

    return NOERROR;
}

/***********************/

CContMenuExt::GetCommandString()

***********************/

STDMETHODIMP CContMenuExt::GetCommandString(    UINT idCommand,
                                                UINT uFlags,
                                                LPUINT lpReserved,
                                                LPSTR lpszName,
                                                UINT uMaxNameLen)
{
    HRESULT hr = E_INVALIDARG;
    TCHAR szHelp[] = TEXT("Sample Context Menu Command");

```

```
TCHAR szVerb[] = TEXT("adcontmenu");

switch(uFlags)
{
case GCS_HELPTEXTA:
    switch(idCommand)
    {
    case IDM_CONTMENU:
        LocalToAnsi((LPSTR)lpszName, szHelp, uMaxNameLen);
        hr = S_OK;
        break;
    }
    break;

case GCS_HELPTEXTW:
    switch(idCommand)
    {
    case IDM_CONTMENU:
        LocalToWideChar((LPWSTR)lpszName, szHelp, uMaxNameLen);
        hr = S_OK;
        break;
    }
    break;

case GCS_VERBA:
    switch(idCommand)
    {
    case IDM_CONTMENU:
        LocalToAnsi((LPSTR)lpszName, szVerb, uMaxNameLen);
        hr = S_OK;
        break;
    }
    break;

case GCS_VERBW:
    switch(idCommand)
    {
    case IDM_CONTMENU:
        LocalToWideChar((LPWSTR)lpszName, szVerb, uMaxNameLen);
        hr = S_OK;
        break;
    }
    break;

case GCS_VALIDATE:
    hr = S_OK;
    break;
}

return hr;
}
```

Registering the Context Menu COM Object in a Display Specifier

6/3/2022 • 3 minutes to read • [Edit Online](#)

When you use COM to create a context menu extension DLL for an Active Directory directory service, the extension must be registered with the Windows registry and Active Directory Domain Services to notify the Active Directory administrative MMC snap-ins and the Windows shell of the extension.

Registering in the Windows Registry

Like all COM servers, a context menu extension must be registered in the registry. The extension is registered under the following key.

```
HKEY_CLASSES_ROOT  
  CLSID  
    <clsid>
```

<clsid> is the string representation of the CLSID as produced by the [StringFromCLSID](#) function. Under the <clsid> key, there is an **InProcServer32** key that identifies the object as a 32-bit in-proc server. Under the **InProcServer32** key, the location of the DLL is specified in the default value and the threading model is specified in the **ThreadingModel** value. All context menu extension must use the "Apartment" threading model.

Registering with Active Directory Domain Services

Context menu extension registration is specific to one locale. If the context menu extension applies to all locales, it must be registered in the object class **displaySpecifier** object in all of the locale subcontainers in the Display Specifiers container. If the context menu extension is localized for a certain locale, it must be registered in the **displaySpecifier** object in that locale's subcontainer. For more information about the Display Specifiers container and locales, see [Display Specifiers](#) and [DisplaySpecifiers Container](#).

There are two display specifier attributes that a context menu extension item can be registered under. These are **adminContextMenu** and **shellContextMenu**.

The **adminContextMenu** attribute identifies administrative context menus to display in Active Directory administrative snap-ins. The context menu appears when the user displays the context menu for objects of the appropriate class in one of the Active Directory administrative MMC snap-ins.

The **shellContextMenu** attribute identifies end-user context menus to display in the Windows shell. The context menu appears when the user views the context menu for objects of the appropriate class in the Windows Explorer. Beginning with Windows Server 2003, the Windows shell no longer displays objects of Active Directory Domain Services.

All of these attributes are multi-valued.

When registering a context menu extension, the values for the **adminContextMenu** and **shellContextMenu** attributes require the following format.

```
<order number>,<clsid>
```

The "<order number>" is an unsigned number that represents the item position in the context menu. When a context menu is displayed, the values are sorted using a comparison of each value's "<order number>". If more than one value has the same "<order number>", those context menu extensions are loaded in the order they are read from the Active Directory server. If possible, use a non-existing "<order number>", that is, one that has not been used by other values in the property. There is no prescribed starting position and gaps are allowed in the "<order number>" sequence.

The "<clsid>" is the string representation of the CLSID as produced by the [StringFromCLSID](#) function.

In the Windows shell, multiple-selection context menu items are supported. In this case, the context menu extension is invoked for each selected object. In Active Directory administrative snap-ins, multiple-selection context menu extension items are also supported. In this case, the [DSOBJECTNAMES](#) structure will contain a [DSOBJECT](#) structure for each directory object selected.

IMPORTANT

For the Windows shell, display specifier information is retrieved at user logon and cached for the user's session. For the administrative snap-ins, the display specifier data is retrieved when the snap-in is loaded and is cached for the duration of the process. For the Windows shell, this means changes to display specifiers take effect after a user logs off and back on again. For the administrative snap-ins, changes take effect when the snap-in or console file is reloaded, that is, if you start a new instance of the console file or new Mmc.exe instance and add the snap-in, the latest display specifier data is retrieved.

For more information, and a code example of how to implement a context menu extension, see [Example Code for Implementation of the Context Menu COM Object](#).

Registering a Static Context Menu Item

6/3/2022 • 3 minutes to read • [Edit Online](#)

The administrative MMC snap-ins of Active Directory Domain Services and the Windows shell provide a mechanism to add an item to the context menu displayed for objects in Active Directory Domain Services. The context menu can invoke any file that can be started with the **ShellExecute** API, such as an application or webpage URL.

Registering with Active Directory Domain Services

Context menu extension registration is specific to one locale. If the context menu extension applies to all locales, it must be registered in the object class **displaySpecifier** object in all of the locale subcontainers in the Display Specifiers container. If the context menu extension is localized for a certain locale, it must be registered in the **displaySpecifier** object in that locale subcontainer. For more information about the Display Specifiers container and locales, see [Display Specifiers](#) and [DisplaySpecifiers Container](#).

There are two display specifier attributes that a static context menu item can be registered under, **adminContextMenu** and **shellContextMenu**.

The **adminContextMenu** attribute identifies administrative context menus to display in the administrative snap-ins of Active Directory Domain Services. The context menu appears when the user displays the context menu for objects of the appropriate class in one of the administrative MMC snap-ins.

The **shellContextMenu** attribute identifies end-user context menus to display in the Windows shell. The context menu appears when the user views the context menu for objects of the appropriate class in the Windows Explorer. Beginning with Windows Server 2003, the Windows shell no longer displays objects that are from Active Directory Domain Services.

All of these attributes are multi-valued.

When registering a static context menu item, the values for the **adminContextMenu** and **shellContextMenu** attributes require the following format.

```
<order number>,<menu text>,<command>
```

The "<order number>" is an unsigned number that represents the item's position in the context menu. When a context menu is displayed, the values are sorted using a comparison of each value's "<order number>". If more than one value has the same "<order number>", those context menu extensions are loaded in the order they are read from the Active Directory server. If possible, use a non-existing "<order number>", that is, one that has not been used by other values in the property. There is no prescribed starting position, and gaps are allowed in the "<order number>" sequence.

The "<menu text>" is the string displayed in the context menu. The "<menu text>" can include one "&" character that precedes the keyboard shortcut character for the menu item. This will cause the preceded character to be underlined. For example, if the "<menu text>" is "&File", the menu text will be displayed as "File", the "F" will be underlined and "F" will be the keyboard shortcut for the menu item.

The "<command>" is the program or file executed by the snap-in. Either the full path must be specified or the file must exist in the computer path environment variable. The file is invoked using the **ShellExecute** function. The "<command>" cannot contain additional parameters, for example, Notepad.exe Myfile.txt. Because **ShellExecute** is used, any file or address that can be passed to **ShellExecute** can be used for "<command>".

For example, if "<command>" contains "d:\file.txt", d:\file.txt will be opened with the application associated with the .txt extension. Likewise, if "<command>" contains "https://www.fabrikam.com", the default web browser is opened and will display the specified webpage. Paths and application names with spaces are allowed. If "<command>" is an application, the selected object's ADsPath and class are passed as command line arguments, separated by a space.

In the Windows shell, multiple-selection context menu items are supported. In this case, the "<command>" is invoked for each selected object. In Active Directory Domain Services' administrative snap-ins, multiple-selection static context menu items are not supported.

IMPORTANT

For the Windows shell, display specifier data is retrieved at user logon and cached for the user's session. For the administrative snap-ins, the display specifier data is retrieved when the snap-in is loaded and is cached for the duration of the process. For the Windows shell, this means changes to display specifiers take effect after a user logs off and back on again. For the administrative snap-ins, changes take effect when the snap-in or console file is reloaded; that is, if you start a new instance of the console file or new Mmc.exe instance and add the snap-in, the latest display specifier data is retrieved.

For more information, and a code example, see [Example Code for Installing a Static Context Menu Item](#).

Example Code for Installing a Static Context Menu Item

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example uses two scripts. The first script (Frommenu.vbs) is the command that is run when the menu item is selected. The second script (Addmenu.vbs) installs the display specifier context menu item to execute the script Frommenu.vbs. This example assumes locale 409 (US English) and extends the user object's context menu in Active Directory administrative snap-ins.

To run the example code

1. Copy the code for Frommenu.vbs below, open Notepad, paste the code into Notepad, save the file as C:\frommenu.vbs, and close Notepad.
2. Copy the code for Addmenu.vbs below, open Notepad, paste the code into Notepad, save the file as C:\addmenu.vbs, and close Notepad.
3. Run Addmenu.vbs.
4. Start the Active Directory Users and Computers snap-in.

FROMMENU.VBS

```

'Frommenu.vbs is the script run when the menu item is chosen.

.....
' Parse the arguments
' First arg is ADsPath of the selected object. Second is Class.
.....
On Error Resume Next

Set oArgs = WScript.Arguments
sText = "This script was run from a display specifier context menu." & vbCrLf & "Selected Item:"
If oArgs.Count > 1 Then
    sText = sText & vbCrLf & " ADsPath: " & oArgs.item(0)
    sText = sText & vbCrLf & " Class: " & oArgs.item(1)
Else
    sText = sText & vbCrLf & "Arg Count: " & oArgs.Count
End If
show_items sText
Err.Number = 0
sBind = oArgs.item(0)
Set dsobj= GetObject(sBind)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If
objname = dsobj.Get("name")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get method"
End If
sText = "Use ADsPath from first argument to bind and get RDN (name) property."
sText = sText & vbCrLf & "Name: " & objname
show_items sText

.....
' Display subroutines
.....
Sub show_items(strText)
    MsgBox strText, vbInformation, "Script from Context Menu"
End Sub

Sub BailOnFailure(ErrNum, ErrText)      strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub

```

ADDMENU.VBS

```

' Addmenu.vbs adds the menu item to run Frommenu.vbs
' from user object's context menu in the admin snap-ins.
On Error Resume Next
Set root= GetObject("LDAP://rootDSE")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If
sConfig = root.Get("configurationNamingContext")
'hardcoded for user class.
sClass = "user"
'hardcoded for US English
sLocale = "409"
sPath = "LDAP://cn=" & sClass & "-Display,cn=" & sLocale & ",cn=DisplaySpecifiers," & sConfig
show_items "Display Specifier: " & sPath
Set obj= GetObject(sPath)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If
'TODO--check if this is already there.
'Add the value for the context menu
sValue = "5,Run My Test Script,c:\frommenu.vbs"
vValue = Array(sValue)
obj.PutEx 3, "adminContextMenu", vValue
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::PutEx method"
End If
' Commit the change.
obj.SetInfo
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADs::SetInfo method"
End If

show_items "Success! Added value to adminContextMenu property of user-Display: " & sValue
.....
' Display subroutines
.....
Sub show_items(strText)
    MsgBox strText, vbInformation, "Add admin context menu"
End Sub

Sub BailOnFailure(ErrNum, ErrText)    strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub

```

Object Creation Wizards

6/3/2022 • 2 minutes to read • [Edit Online](#)

In the administrative MMC snap-ins of Active Directory Domain Services, the user can create new objects in a directory by opening the context menu for the container where the new object will be created, choosing **New**, and choosing the class of object to create. Creating new instances of an object starts the object creation wizard. Each object class may specify the use of a specific creation wizard, or it may use a generic creation wizard. For common classes, such as **user** and **organizationalUnit**, the Active Directory Users and Computers snap-in provides a standard set of creation wizards.

There are two ways to extend a creation wizard:

- Replace an existing wizard, or provide one if one does not exist for the class: The existing wizard is replaced by creating a *primary object creation extension*. A primary creation extension provides the first set of pages and is hosted in the same way as native pages. A primary creation extension also supports the extensibility mechanism so that other creation wizard extensions can be invoked. For an example of a primary extension, see the `scpwizard` sample in the Platform Software Development Kit (SDK).
- Extend an existing wizard: An existing wizard can be extended with an *secondary object creation extension*. A secondary creation extension adds wizard pages to the native pages or primary extension. For more information and an example of a secondary creation extension, see the `userwizard` sample in the Platform SDK.

Developer Audience

This documentation assumes that the reader is familiar with COM operation and component development using C++. It is not currently possible to create an extension to the Active Directory object creation wizard using Visual Basic.

Creating an Active Directory Object Creation Extension

Both primary and secondary object creation extensions are COM in-proc servers that implement certain interfaces and are registered with Active Directory Domain Services.

To create and install an object creation extension

1. Create the object creation extension DLL. An object creation extension is a COM in-proc server that, at a minimum, implements the **IDsAdminNewObjExt** interface. For more information, see [Implementing the Object Creation Extension COM Object](#).
2. Install the creation extension on computers where the creation extension is to be used. To do this, create a Microsoft Windows Installer package for the creation extension DLL and deploy the package appropriately using the group policy. For more information, see [Distributing User Interface Components](#).
3. Register the creation extension in the Windows registry and with Active Directory Domain Services. For more information, see [Registering the Object Creation Extension](#).

Using an Object Creation Wizard

An object creation wizard can also be invoked from an application other than the administrative MMC snap-ins of Active Directory Domain Services. For more information, see [Invoking Creation Wizards from Your Application](#).

If a creation wizard is not registered for an object class, the administrative snap-ins provide a generic creation

wizard. The generic creation wizard is built at run time from the list of mandatory properties for the class of object created. For each mandatory property, a page is added to the UI. The generic creation wizard is not extensible. If extensibility is required, it must be replaced with a primary object creation extension.

Implementing the Object Creation Extension COM Object

6/3/2022 • 3 minutes to read • [Edit Online](#)

An object creation extension is a COM object implemented as an in-proc server. Both primary and secondary object creation extensions must implement the [IDsAdminNewObjExt](#) interface.

Implementing [IDsAdminNewObjExt](#)

When the object creation wizard is created, it initializes each object creation extension by calling the extension's [IDsAdminNewObjExt::Initialize](#) method. The [Initialize](#) method supplies the extension with information about the container the object is being created in, the class name of the new object and information about the wizard itself. If the object creation wizard is started to create a new object from an existing object, the *pADsCopySource* parameter will not be **NULL**. In this case, the extension should attempt to obtain as much data from the object being copied as is feasible.

After the extension is initialized, the [IDsAdminNewObjExt::AddPages](#) method is called. The extension must add the page or pages to the wizard during this method. A wizard page is created by filling in a [PROPSHEETPAGE](#) structure and then passing this structure to the [CreatePropertySheetPage](#) function. The page is then added to the wizard by calling the callback function that is passed to [AddPages](#) in the *lpfnAddPage* parameter.

Before the extension page is displayed, [IDsAdminNewObjExt::SetObject](#) is called. This supplies the extension with an [IADs](#) interface pointer for the object being created.

While the wizard page is displayed, the page should handle and respond to any necessary wizard notification messages, such as [PSN_SETACTIVE](#) and [PSN_WIZNEXT](#).

When the user completes all of the wizard pages, the wizard will display a "Finish" page that provides a summary of the data entered. The wizard obtains this data by calling the [IDsAdminNewObjExt::GetSummaryInfo](#) method for each of the extensions. The [GetSummaryInfo](#) method provides a **BSTR** that contains the text data displayed in the "Finish" page. An object creation extension does not have to supply summary data. In this case, [GetSummaryInfo](#) should return **E_NOTIMPL**. [GetSummaryInfo](#) is only called one time for each extension, not per page, so if the object creation extension adds more than one page, the extension must combine the summary data into one string.

When the user clicks the **Finish** button in the "Finish" page, the wizard calls each of the extension's [IDsAdminNewObjExt::WriteData](#) methods with the **DSA_NEWOBJ_CTX_PRECOMMIT** context. When this occurs, the extension should write the gathered data into the appropriate properties using the [IADs::Put](#) or [IADs::PutEx](#) method. The [IADs](#) interface is provided to the extension in the [IDsAdminNewObjExt::SetObject](#) method. The extension should not commit the cached properties by calling [IADs::SetInfo](#). When all of the properties have been written, the primary object creation extension commits the changes by calling [IADs::SetInfo](#). This is discussed in more detail below.

If an error occurs, the extension will be notified of the error and during which operation it occurred when the [IDsAdminNewObjExt::OnError](#) method is called.

Implementing a Primary Object Creation Wizard

The implementation of a primary object creation wizard is identical to a secondary object creation wizard, except that a primary object creation wizard must perform a few more steps.

Prior to the first page being dismissed, the object creation wizard must create the temporary directory object. To do this, call the [IDsAdminNewObjPrimarySite::CreateNew](#) method. A pointer to the [IDsAdminNewObjPrimarySite](#) interface is obtained by calling [QueryInterface](#) with [IID_IDsAdminNewObjPrimarySite](#) on the [IDsAdminNewObj](#) interface that is passed to [IDsAdminNewObjExt::Initialize](#). The [CreateNew](#) method creates a new temporary object and calls [IDsAdminNewObjExt::SetObject](#) for each extension.

When an object creation wizard contains more than one page, the system implements a "Finish" page that displays a summary of the object information to be saved. When the **Finish** button on the "Finish" page is clicked, the system will call each of the object creation extension's [IDsAdminNewObjExt::WriteData](#) method and then commit the temporary object to persistent memory. If, however, the object creation wizard only contains one page, the page will have **OK** and **Cancel** buttons instead of the **Back**, **Next** and **Cancel** buttons normally found in a wizard and no "Finish" page is provided. Because of this, a single-page object creation extension wizard must call [IDsAdminNewObjPrimarySite::Commit](#) to perform the write and save operations. A single-page primary object creation extension should call **Commit** in response to the [PSN_WIZFINISH](#) notification.

Because other object creation extensions can add pages to the wizard, the primary object creation extension may not know if there is more than one page in the wizard. This is not an issue for two reasons: First, if the system implements the "Finish" page, the primary object creation extension will receive the [PSN_WIZNEXT](#) notification instead of the [PSN_WIZNEXT](#) notification. Second, **Commit** will fail harmlessly if the wizard contains more than one page.

Registering the Object Creation Extension

6/3/2022 • 2 minutes to read • [Edit Online](#)

When an object creation extension DLL in Active Directory Domain Services is created, it must be registered with the Windows registry and Active Directory Domain Services to make COM and the Active Directory administrative MMC snap-ins aware of the extension.

Registering in the Windows Registry

Like all COM servers, an object creation extension must be registered in the Windows registry. The extension is registered under the following key:

```
HKEY_CLASSES_ROOT
  CLSID
    <extension CLSID>
      InProcServer32
        (Default) = <extension path>
        ThreadingModel = Apartment
```

"<extension CLSID>" is the string representation of the CLSID as produced by the [StringFromCLSID](#) function. "<extension path>" contains the path and file name of the extension DLL. The **ThreadingModel** value for all object creation extensions must be "Apartment".

Registering with Active Directory Domain Services

Object creation extension registration is specific to one locale. If the object creation extension applies to all locales, it must be registered in the object class's **displaySpecifier** object in all of the locale subcontainers in the **DisplaySpecifiers** container. If the object creation extension is localized for a certain locale, register it in the **displaySpecifier** object in that locale's subcontainer. For more information about the **DisplaySpecifiers** container and locales, see [Display Specifiers](#) and [DisplaySpecifiers Container](#).

There are two **DisplaySpecifier** attributes that an object creation extension can be registered under. These are **creationWizard** and **createWizardExt**.

The **creationWizard** attribute identifies primary object creation extensions to replace the existing or native object creation wizard in Active Directory administrative snap-ins. A primary creation extension provides the first set of pages and is hosted in the same way as native pages. This attribute is single-valued and requires the following format:

```
<CLSID>
```

The "<CLSID>" is the string representation of the COM object's CLSID as produced by the [StringFromCLSID](#) function.

The **createWizardExt** attribute identifies secondary object creation extensions. A secondary creation extension adds wizard pages to the native pages or primary extension. This attribute is multi-valued and requires the following format:

```
<order number>,<CLSID>
```

The "<order number>" is an unsigned number that represents the page's position in the wizard. When a creation wizard is displayed, the values are sorted using a comparison of each value's "<order number>". If more than one value has the same "<order number>", those pages are loaded in the order they are read from the Active Directory server. If possible, you should use a non-existing "<order number>" (that is, one that has not been used by other values in the property). There is no prescribed starting position, and gaps are allowed in the "<order number>" sequence.

The "<CLSID>" is the string representation of the COM object's CLSID as produced by the [StringFromCLSID](#) function.

Invoking Creation Wizards from Your Application

6/3/2022 • 2 minutes to read • [Edit Online](#)

An application or component can use the same directory service object creation wizards used by the Active Directory administrative MMC snap-ins. This is accomplished with the **IDsAdminCreateObj** interface.

Using the **IDsAdminCreateObj** Interface

An application or component (client) creates an instance of the **IDsAdminCreateObj** interface by calling **CoCreateInstance** with the **CLSID_DsAdminCreateObj** class identifier. COM must be initialized by calling **CoInitialize** before **CoCreateInstance** is called.

The client then calls **IDsAdminCreateObj::Initialize** to initialize the **IDsAdminCreateObj** object.

IDsAdminCreateObj::Initialize accepts an **IADsContainer** interface pointer that represents the container that the object should be created in, and the class name of the object to be created. When creating user objects, it is also possible to specify an existing object that will be copied to the new object.

When the **IDsAdminCreateObj** object has been initialized, the client calls **IDsAdminCreateObj::CreateModal** to display the object creation wizard.

Unlike most class and interface identifiers, **CLSID_DsAdminCreateObj** and **IID_IDsAdminCreateObj** are not defined in a library file. This means you must define the storage for these identifiers within your application. To do this, you must include the file **initguid.h** immediately before including **dsadmin.h**. The **initguid.h** file must only be included once in an application. The following code example shows how to include these files.

```
#include <initguid.h>
#include <dsadmin.h>
```

The following code example shows how the **IDsAdminCreateObj** interface can be created and used to start the object creation wizard for a user object.

```
// Add activeds.lib to your project
// Add adsiid.lib to your project

#include "stdafx.h"
#include <atibase.h>
#include <atlstr.h>
#include "activeds.h"
#include <initguid.h> // Only include this in one source file
#include <dsadmin.h>

// GetUserContainer() function binds to the user container
IADsContainer* GetUserContainer(void)
{
    IADsContainer *pUsers = NULL;
    HRESULT hr;
    IADs *pRoot;

    // Bind to the rootDSE.
    hr = ADsGetObject(L"LDAP://rootDSE", IID_IADs, (LPVOID*)&pRoot);

    if(SUCCEEDED(hr))
    {
        VARIANT var;
        VariantInit(&var);
        CComBSTR sbstr(L"defaultNamingContext");
```

```

// Get the default naming context (domain) DN.
hr = pRoot->Get(sbstr, &var);
if(SUCCEEDED(hr) && (VT_BSTR == var.vt))
{
    CStringW sstr(L"LDAP://CN=Users,");
    sstr += var.bstrVal;

    // Bind to the User container.
    hr = ADsGetObject(sstr, IID_IADsContainer, (LPVOID*)&pUsers);

    VariantClear(&var);
}
}

return pUsers;
}

// The LaunchNewUserWizard() function launches the user wizard.
HRESULT LaunchNewUserWizard(IADs** ppAdsOut)
{
    if(NULL == ppAdsOut)
    {
        return E_INVALIDARG;
    }

    HRESULT hr;
    IDsAdminCreateObj* pCreateObj = NULL;

    hr = ::CoCreateInstance(CLSID_DsAdminCreateObj,
                           NULL,
                           CLSCTX_INPROC_SERVER,
                           IID_IDsAdminCreateObj,
                           (void**)&pCreateObj);

    if(SUCCEEDED(hr))
    {
        IADsContainer *pContainer;

        pContainer = GetUserContainer();

        if(pContainer)
        {
            hr = pCreateObj->Initialize(pContainer, NULL, L"user");
            if(SUCCEEDED(hr))
            {
                HWND      hwndParent;

                hwndParent = GetDesktopWindow();

                hr = pCreateObj->CreateModal(hwndParent, ppAdsOut);
            }
        }

        pContainer->Release();
    }

    pCreateObj->Release();
}

return hr;
}

// Entry point to the application
int main(void)
{
    HRESULT hr;
    IADs *pAds = NULL;

```

```
CoInitialize(NULL);

hr = LaunchNewUserWizard(&pAds);
if(S_OK == hr) && (NULL != pAds)
{
    pAds->Release();
}

CoUninitialize();

return 0;
}
```

Using Standard Dialog Boxes to Handle Objects in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Windows provides a Common Dialog Box library for common operations, such as File Open, File Browse, and so on. Beginning with Windows 2000, the system provides dialog boxes that can be used for common user interface operations in Active Directory Domain Services.

The following dialog boxes are available for use with Active Directory Domain Services:

- [Directory Object Picker](#)
- [Domain Browser](#)
- [Container Browser](#)
- [Active Directory Users and Computers Property Sheets](#)

In addition, Active Directory Domain Services creation wizards can also be invoked from an application. For more information, see [Invoking Creation Wizards from Your Application](#).

Directory Object Picker

6/3/2022 • 3 minutes to read • [Edit Online](#)

The directory object picker dialog box enables a user to select one or more objects from either the global catalog, a domain or computer, or a workgroup. The object types from which a user can select include user, contact, group, and computer objects. For more information about Active Directory Domain Services, see [Active Directory Domain Services](#).

To display an object picker dialog box:

1. Call the [CoCreateInstance](#) or [CoCreateInstanceEx](#) function to create an instance of the [IDsObjectPicker](#) interface.
2. Call the [IDsObjectPicker::Initialize](#) method to initialize the dialog box.
3. Call the [IDsObjectPicker::InvokeDialog](#) method to display the dialog box.
4. Call the [IDataObject::GetData](#) method of the [IDataObject](#) instance returned by the object picker dialog box to retrieve the [CFSTR_DSOP_DS_SELECTION_LIST](#) data. The [CFSTR_DSOP_DS_SELECTION_LIST](#) clipboard format provides an [HGLOBAL](#) that contains a [DS_SELECTION_LIST](#) structure. The [DS_SELECTION_LIST](#) structure contains data about the items selected in the object picker dialog box.

If the Security Identifier (SID) is required for an object, this should be requested directly from the object picker by adding the [objectSID](#) attribute to the list of attributes to retrieve for the selected object. Passing the returned object name to the [LsaLookupNames](#) or [LookupAccountName](#) function is not recommended because the name lookup will be redundant and may fail in some cases.

If a reference to any selected objects will be saved, the distinguished name should not be saved because the object may move, get renamed, or may change due to locale differences. For security principals, the [objectSID](#) should be requested for the object and securely saved. If the name of the security principal is needed later, it can be retrieved with the [LookupAccountSid](#) function. For all other objects, the [objectGUID](#) should be requested and saved.

Initialization

When the object picker dialog box is initialized, a set of scope types and filters is specified. The specified scope types determine the locations, domains or computers for example, from which a user can select objects. The filters determine the types of objects that a user can select from a given scope type. For more information, see the Scopes and Filters section below.

By default, a user can select a single object in the directory object picker dialog box. To enable multiple selections, set the [DSOP_FLAG_MULTISELECT](#) flag in the [fOptions](#) member of the [DSOP_INIT_INFO](#) structure when the dialog box initialized.

Scopes and Filters

The **Look in** drop-down list contains the scopes from which a user can select objects. A scope is a domain, computer, workgroup, or global catalog that stores data about, and provides access to, a set of available objects. The entries in the scope list depend on the scope types and the target computer specified when the [IDsObjectPicker::Initialize](#) method was last called to initialize the object picker dialog box.

A scope type is a generic category of scopes, such as all domains in the enterprise to which the target computer belongs, or the global catalog for the target computer's enterprise, or the target computer itself. For each specified scope type, the dialog box uses the context of the target computer to determine the scope list entries.

The `IDsObjectPicker::Initialize` method takes a pointer to a `DSOP_INIT_INFO` structure that contains an array of `DSOP_SCOPE_INIT_INFO` structures. Each entry in the `DSOP_SCOPE_INIT_INFO` array specifies one or more scope types as well as applicable filters and other attributes. The filters determine the types of objects, such as users, groups, contacts, and computers, that the user can select from a given scope type. When the user selects a scope from the list, the dialog box applies the filters for that scope type to display a list of objects from which the user can select.

Each `DSOP_SCOPE_INIT_INFO` structure contains a `DSOP_FILTER_FLAGS` structure that specifies the filters for that scope type. The `DSOP_FILTER_FLAGS` structure distinguishes between up-level and down-level scopes:

- An up-level scope is a global catalog or a domain that supports the ADSI [LDAP provider](#).
- A down-level scope includes workgroups and all individual computers. The dialog box uses the ADSI WinNT provider to access a down-level scope.

There are two sets of filter flags defined for use in the `DSOP_FILTER_FLAGS` structure: one for up-level scopes and one for down-level scopes. The `Uplevel` member of the `DSOP_FILTER_FLAGS` structure is a `DSOP_UPLEVEL_FILTER_FLAGS` structure that specifies the filters for up-level scopes. The `fIDownlevel` member of the `DSOP_FILTER_FLAGS` structure is a set of flags that specify the filters for down-level scopes.

Domain Browser

6/3/2022 • 2 minutes to read • [Edit Online](#)

Using the [IDsBrowseDomainTree](#) interface, an application can display a domain browser dialog box and obtain the DNS name of the domain selected by the user. An application can also use the [IDsBrowseDomainTree](#) interface to obtain data about all domain trees and domains within a forest.

An instance of the [IDsBrowseDomainTree](#) interface is created by calling [CoCreateInstance](#) with the [CLSID_DsDomainTreeBrowser](#) class identifier as shown below.

The [IDsBrowseDomainTree::SetComputer](#) method can be used to specify which computer and credentials are used as the basis for retrieving the domain data. When [SetComputer](#) is called on a particular [IDsBrowseDomainTree](#) instance, [IDsBrowseDomainTree::FlushCachedDomains](#) must be called before [SetComputer](#) is called again.

The [IDsBrowseDomainTree::BrowseTo](#) method is used to display the domain browser dialog box. When the user selects a domain and clicks the OK button, the [IDsBrowseDomainTree::BrowseTo](#) returns [S_OK](#) and the *ppszTargetPath* parameter contains the name of the selected domain. When the name string is no longer required, the caller must free the string by calling [CoTaskMemFree](#).

The following code example shows how to use the [IDsBrowseDomainTree](#) interface to display the domain browser dialog box.

```

#include <shlobj.h>
#include <dsclient.h>

void main(void)
{
    HRESULT     hr;

    hr = CoInitialize(NULL);
    if(FAILED(hr))
    {
        return;
    }

    IDsBrowseDomainTree *pDsDomains = NULL;

    hr = ::CoCreateInstance( CLSID_DsDomainTreeBrowser,
                           NULL,
                           CLSCTX_INPROC_SERVER,
                           IID_IDsBrowseDomainTree,
                           (void **)(&pDsDomains));

    if(SUCCEEDED(hr))
    {
        LPOLESTR     pstr;

        hr = pDsDomains->BrowseTo( GetDesktopWindow(),
                                    &pstr,
                                    0);

        if(S_OK == hr)
        {
            wprintf(pstr);
            wprintf(L"\n");
            CoTaskMemFree(pstr);
        }

        pDsDomains->Release();
    }

    CoUninitialize();
}

```

The [IDsBrowseDomainTree::GetDomains](#) method is used to obtain domain tree data. The domain data is supplied in a [DOMAINTREE](#) structure. The DOMAINTREE structure contains the size of the structure and the number of domain elements in the tree. The DOMAINTREE structure also contains one or more [DOMAINDESC](#) structures. The DOMAINDESC contains data about a single element in the domain tree, including child and sibling data. The siblings of a domain can be enumerated by accessing the [pdNextSibling](#) member of each subsequent DOMAINDESC structure. The children of the domain can be retrieved in a similar manner by accessing the [pdChildList](#) member of each subsequent DOMAINDESC structure.

The following code example shows how to obtain and access the domain tree data using the [IDsBrowseDomainTree::GetDomains](#) method.

```

// Add dsuiext.lib to the project.

#include "stdafx.h"
#include <shlobj.h>
#include <dsclient.h>

//The PrintDomain() function displays the domain name
void PrintDomain(DOMAINDESC *pDomainDesc, DWORD dwIndentLevel)
{
    DWORD i;

```

```

// Display the domain name.
for(i = 0; i < dwIndentLevel; i++)
{
    wprintf(L"  ");
}
wprintf(pDomainDesc->pSzName);
wprintf(L"\n");
}

//The WalkDomainTree() function traverses the domain tree and prints the current domain name
void WalkDomainTree(DOMAINDESC *pDomainDesc, DWORD dwIndentLevel = 0)
{
    DOMAINDESC *pCurrent;

    // Walk through the current item and any siblings it may have.
    for(pCurrent = pDomainDesc;
        NULL != pCurrent;
        pCurrent = pCurrent->pdNextSibling)
    {
        // Print the current domain name.
        PrintDomain(pCurrent, dwIndentLevel);

        // Walk the child tree, if one exists.
        if(NULL != pCurrent->pdChildList)
        {
            WalkDomainTree(pCurrent->pdChildList,
                           dwIndentLevel + 1);
        }
    }
}

// Entry point for application
int main(void)
{
    HRESULT hr;
    IDsBrowseDomainTree *pBrowseTree;
    CoInitialize(NULL);

    hr = CoCreateInstance(CLSID_DsDomainTreeBrowser,
                          NULL,
                          CLSCTX_INPROC_SERVER,
                          IID_IDsBrowseDomainTree,
                          (void**)&pBrowseTree);

    if(SUCCEEDED(hr))
    {
        DOMAINTREE *pDomTreeStruct;
        hr = pBrowseTree->GetDomains(&pDomTreeStruct,
                                      DBDTF_RETURNFQDN);
        if(SUCCEEDED(hr))
        {
            WalkDomainTree(&pDomTreeStruct->aDomains[0]);
            hr = pBrowseTree->FreeDomains(&pDomTreeStruct);
        }
        pBrowseTree->Release();
    }
    CoUninitialize();
}

```

Container Browser

6/3/2022 • 4 minutes to read • [Edit Online](#)

An application can use the [DsBrowseForContainer](#) function to display a dialog box that can be used to browse through the containers in an Active Directory Domain Service. The dialog box displays the containers in a tree format and enables the user to navigate through the tree using the keyboard and mouse. When the user selects an item in the dialog box, the ADsPath of the container selected by the user is provided.

[DsBrowseForContainer](#) takes a pointer to a [DSBROWSEINFO](#) structure that contains data about the dialog box and returns the ADsPath of the selected item.

The **pszRoot** member is a pointer to a Unicode string that contains the root container of the tree. If **pszRoot** is **NULL**, the tree will contain the entire tree.

The **pszPath** member receives the ADsPath of the selected object. It is possible to specify a particular container to be visible and selected when the dialog box is first displayed. This is accomplished by setting **pszPath** to the ADsPath of the item to be selected and setting the **DSBI_EXPANDONOPEN** flag in **dwFlags**.

The contents and behavior of the dialog box can also be controlled at runtime by implementing a [BFFCallBack](#) function. The [BFFCallBack](#) function is implemented by the application and is called when certain events occur. An application can use these events to control how the items in the dialog box are displayed as well as the actual contents of the dialog box. For example, an application can filter the items displayed in the dialog box by implementing a [BFFCallBack](#) function that can handle the **DSBM_QUERYINSERT** notification. When a **DSBM_QUERYINSERT** notification is received, use the **pszADsPath** member of the [DSBITEM](#) structure to determine which item is about to be inserted. If the application determines that the item should not be displayed, it can hide the item by performing the following steps.

1. Add the **DSBF_STATE** flag to the **dwMask** member of the [DSBITEM](#) structure.
2. Add the **DSBS_HIDDEN** flag to the **dwStateMask** member of the [DSBITEM](#) structure.
3. Add the **DSBS_HIDDEN** flag to the **dwState** member of the [DSBITEM](#) structure.
4. Return a nonzero value from the [BFFCallBack](#) function.

In addition to hiding certain items, an application can modify the text and icon displayed for the item by handling the **DSBM_QUERYINSERT** notification. For more information, see [DSBITEM](#).

The [BFFCallBack](#) function can use the **BFFM_INITIALIZED** notification to obtain the handle of the dialog box. The application can use this handle to send messages, such as **BFFM_ENABLEOK**, to the dialog box. For more information about the messages that can be sent to the dialog box, see [BFFCallBack](#).

The dialog box handle can also be used to gain direct access to the controls in the dialog box. **DSBID_BANNER** is the identifier for the static text control that the **pszTitle** member of the [DSBROWSEINFO](#) structure is displayed in. **DSBID_CONTAINERLIST** is the identifier of the tree view control used to display the tree contents. Use of these items should be avoided, if possible, to prevent future application compatibility problems.

The following C++ code example shows how to use the [DsBrowseForContainer](#) function to create the container browser dialog box and implement a [BFFCallBack](#) function. The [BFFCallBack](#) uses the **DSBM_QUERYINSERT** notification to change the display name for each item to the distinguished name of the item.

```
#include <shlobj.h>
#include <dsclient.h>
#include <atbase.h>
```

```

*****
WideCharToLocal()

*****/


int WideCharToLocal(LPTSTR pLocal, LPWSTR pWide, DWORD dwChars)
{
    *pLocal = NULL;
    size_t nWideLength = 0;
    wchar_t *pwszSubstring;

    nWideLength = wcslen(pWide);

#ifndef UNICODE
    if(nWideLength < dwChars)
    {
        wcsncpy_s(pLocal, pWide, dwChars);
    }
    else
    {
        wcsncpy_s(pLocal, pWide, dwChars-1);
        pLocal[dwChars-1] = NULL;
    }
#else
    if(nWideLength < dwChars)
    {
        WideCharToMultiByte(    CP_ACP,
                               0,
                               pWide,
                               -1,
                               pLocal,
                               dwChars,
                               NULL,
                               NULL);
    }
    else
    {
        pwszSubstring = new WCHAR[dwChars];
        wcsncpy_s(pwszSubstring,pWide,dwChars-1);
        pwszSubstring[dwChars-1] = NULL;

        WideCharToMultiByte(    CP_ACP,
                               0,
                               pwszSubstring,
                               -1,
                               pLocal,
                               dwChars,
                               NULL,
                               NULL);
    }
    delete [] pwszSubstring;
}
#endif

return lstrlen(pLocal);
}

*****


BrowseCallback()

*****/


int CALLBACK BrowseCallback(HWND hWnd,
                           UINT uMsg,
                           LPARAM lParam,
                           LPARAM lpData)
{

```

```

switch(uMsg)
{
case DSBM_QUERYINSERT:
{
    BOOL fReturn = FALSE;
    DSBITEM *pItem = (DSBITEM*)lParam;

    /*
    If this is to the root item, get the distinguished name
    for the object and set the display name to the
    distinguished name.
    */
    if(!(pItem->dwState & DSBS_ROOT))
    {
        HRESULT hr;
        IADS    *pads;

        hr = ADsGetObject(pItem->pszADsPath ,
                          IID_IADS, (LPVOID*)&pads);
        if(SUCCEEDED(hr))
        {
            VARIANT var;

            VariantInit(&var);
            hr = pads->Get(CComBSTR("distinguishedName"),
                             &var);
            if(SUCCEEDED(hr))
            {
                if(VT_BSTR == var.vt)
                {
                    WideCharToLocal(pItem->szDisplayName,
                                    var.bstrVal,
                                    DSB_MAX_DISPLAYNAME_CHARS);
                    pItem->dwMask |= DSBF_DISPLAYNAME;
                    fReturn = TRUE;
                }
            }

            VariantClear(&var);
        }

        pads->Release();
    }
}

return fReturn;
}

break;
}

return FALSE;
}

*****  

BrowseForContainer()

*****/  

HRESULT BrowseForContainer(HWND hwndParent,
                           LPOLESTR *ppContainerADsPath)
{
    HRESULT hr = E_FAIL;
    DSBROWSEINFO dsbi;
    OLECHAR wszPath[MAX_PATH * 2];
    DWORD result;

    if(!ppContainerADsPath)
    {

```

```

    {
        return E_INVALIDARG;
    }

    ZeroMemory(&dsbi, sizeof(dsbi));
    dsbi.hwndOwner = hwndParent;
    dsbi.cbStruct = sizeof(DSBROWSEINFO);
    dsbi.pszCaption = TEXT("Browse for a Container");
    dsbi.pszTitle = TEXT("Select an Active Directory container.");
    dsbi.pszRoot = NULL;
    dsbi.pszPath = wszPath;
    dsbi.cchPath = sizeof(wszPath)/sizeof(OLECHAR);
    dsbi.dwFlags = DSBI_INCLUDEHIDDEN |
        DSBI_IGNORETREATASLEAF |
        DSBI_RETURN_FORMAT;
    dsbi.pfnCallback = BrowseCallback;
    dsbi.lParam = 0;
    dsbi.dwReturnFormat = ADS_FORMAT_X500;

    // Display the browse dialog box.
    // Returns -1, 0, IDOK, or IDCANCEL.
    result = DsBrowseForContainer(&dsbi);
    if(IDOK == result)
    {
        // Allocate memory for the string.
        *ppContainerADsPath = (OLECHAR*)CoTaskMemAlloc(
            sizeof(OLECHAR)*(wcslen(wszPath) + 1));
        if(*ppContainerADsPath)
        {
            wcscpy_s(*ppContainerADsPath, wszPath);
            // Caller must free using CoTaskMemFree.
            hr = S_OK;
        }
        else
        {
            hr = E_FAIL;
        }
    }
    else
    {
        hr = E_FAIL;
    }
}

return hr;
}

```

Active Directory Users and Computers Property Sheets

6/3/2022 • 8 minutes to read • [Edit Online](#)

The Active Directory Users and Computers MMC snap-in is designed to display a property sheet for various objects in an Active Directory server. The property sheet contains one or more pages that are used to view and modify object data. Different object types have different sets of pages displayed for them. The Active Directory Users and Computers MMC snap-in also enables third party vendors to add custom pages to the property sheet for a specific type of object. For more information, see [Property Pages for Use with Display Specifiers](#).

Some applications, other than the Active Directory Users and Computers MMC snap-in, must provide the user with the ability view and edit attributes for an object in an Active Directory server. The application could implement its own property sheets, but it is better to offer a consistent user interface to reduce confusion and learning time. Fortunately, the Active Directory Users and Computers MMC snap-in allows any OLE COM application to display a property sheet for an object that is identical to the property sheet that would be displayed by the Active Directory Users and Computers MMC snap-in for the same object.

For more information and a code example that hosts an Active Directory Users and Computers property sheet, see the PropSheetHost sample in the Platform Software Development Kit (SDK).

Developer Audience

This documentation assumes that the reader is familiar with COM operation and component development using C++. Currently, it is not possible to create an Active Directory property sheet extension using Visual Basic.

Hosting an Active Directory Users and Computers Property Sheet

To display a property sheet for an object in an Active Directory server

1. Create a window that can be used to process messages. This can be an existing window or a special purpose window. This is known as the *hidden window*.
2. Create an OLE COM object that is derived from [IDataObject](#). This data object must support the following data formats:
 - **CFSTR_DSOBJECTNAMES** This data format contains a **DSOBJECTNAMES** that identifies the object that the property sheet applies to. When hosting a property sheet, the more significant members of the **DSOBJECTNAMES** structure are shown in the following list.

clsidNamespace Reserved. Set this to a GUID for your application here in case that it is used in the future.

aObjects Contains an array of **DSOBJECT** structures. Each **DSOBJECT** structure represents a single directory object. The **cItems** member contains the number of elements in the array. Only the first object in this array is used. Other objects are ignored.

- **CFSTR_DSDISPLAYSPECOPTIONS** This data format contains a **DSDISPLAYSPECOPTIONS** structure that contains data that will be used by the property pages, such as where to load the property pages from, the server and credentials to use, and so on. The more significant members of the **DSDISPLAYSPECOPTIONS** are shown in the following list.

offsetAttribPrefix The attribute prefix string determines where the list of property pages is

obtained. This must contain one of the following strings.

ATTRIBUTE PREFIX STRING	DESCRIPTION
"admin"	The property pages are loaded from the adminPropertyPages attribute.
"shell"	The property pages are loaded from the shellPropertyPages attribute.

- **CFSTR_DS_PROPSHEETCONFIG** This data format contains a **PROPSHEETCFG** structure that contains property sheet host data. When hosting a property sheet, the more significant members of the **PROPSHEETCFG** structure contain the data shown in the following list.

INotifyHandle Must be zero. **hwndParentSheet** Contains the handle of the window to receive **WM_ADSPROP_NOTIFY_CHANGE** messages when something in one of the pages changes and is applied. Can be **NULL** if this message is not desired.

hwndHidden Contains the handle of the window to receive **WM_DSA_SHEET_CREATE_NOTIFY** and **WM_DSA_SHEET_CLOSE_NOTIFY** messages. Set this to the handle of your hidden window.

wParamSheetClose Contains an application-defined identifier that is returned in the *wParam* in the **WM_DSA_SHEET_CLOSE_NOTIFY** message. If this member is zero, the **WM_DSA_SHEET_CLOSE_NOTIFY** message will not be posted to the hidden window.

3. Create an instance of the **CLSID_DsPropertyPages** object and obtain the **IShellExtInit** interface for the object. It is also possible to duplicate the behavior of the **CLSID_DsPropertyPages** object. For more information, see Duplicating the Behavior of the **CLSID_DsPropertyPages** Object.
4. Initialize the **CLSID_DsPropertyPages** object by calling the **IShellExtInit::Initialize** method. The *pidlFolder* and *hkeyProgID* parameters are not used in this method. The *pdtobj* parameter is the pointer to the data object created in Step 2. When the **IShellExtInit::Initialize** method is called, the **CLSID_DsPropertyPages** object will save a reference to the data object.
5. Obtain the **IShellPropSheetExt** interface for the **CLSID_DsPropertyPages** object and call the **IShellPropSheetExt::AddPages** method. The *lpfnAddPage* parameter is the address of a callback function that you must implement. The format of this function is shown below. If the callback function is declared as a member of a C++ class, the callback function must be declared as **static**. The *lParam* parameter is an application-defined value that can be used to identify the object that implements the callback function. When the **IShellPropSheetExt::AddPages** method is called, the **CLSID_DsPropertyPages** object will obtain the data from the data object and enumerate the property pages registered for the object display specifiers. The **CLSID_DsPropertyPages** object will then enumerate the property page objects, calling each object's **IShellPropSheetExt::AddPages** method.

```
BOOL CALLBACK AddPagesCallback(HPROPSHEETPAGE, LPARAM)
```

6. Each page added by the property page objects will result in your callback function being called with the handle to the property page and the application-defined value. Your callback function must store each property page handle that is passed. When the **CLSID_DsPropertyPages** object's **IShellPropSheetExt::AddPages** method returns, all pages will have been added via your callback function.
7. Fill in a **PROPSHEETHEADER** structure to display the property sheet. The **phpage** member receives a pointer to an array of page handles that were collected by your callback function. The **nPages** member receives the number of pages in the page handle array.

8. Display the property sheet by calling the [PropertySheet](#) function.

If the data in any page is changed and the **OK** or **Apply** buttons are clicked, the window identified by the **hwndParentSheet** member of the [PROPSHEETCFG](#) structure will receive a [WM_ADSPROP_NOTIFY_CHANGE](#) message. This message is strictly a notification and requires no specific action.

When the page is closed, the window identified by the **hwndHidden** member of the [PROPSHEETCFG](#) structure will receive a [WM_DSA_SHEET_CLOSE_NOTIFY](#) message. This message is strictly a notification and requires no specific action to be performed.

In some cases, the existing property sheets will need to display a secondary property sheet. For example, if you display the property sheet for a user object and select the **Member Of** page, a list of groups that the user is a member of will be displayed. If you double-click one of these groups in the list, the property sheet for that group will be displayed. The primary property sheet does not display the secondary sheet by itself. It requests that the host display the secondary sheet by sending a [WM_DSA_SHEET_CREATE_NOTIFY](#) message to the window identified by the **hwndHidden** member of the [PROPSHEETCFG](#) structure. The *wParam* of the [WM_DSA_SHEET_CREATE_NOTIFY](#) message is a pointer to a [DSA_SEC_PAGE_INFO](#) structure that contains information about the secondary property sheet and the object that it represents. In response to this message, the property sheet host must display the secondary property sheet in the same manner as shown above. After processing the [WM_DSA_SHEET_CREATE_NOTIFY](#) message, the message receiver must free the [DSA_SEC_PAGE_INFO](#) structure by passing the *wParam* value to the [LocalFree](#) function.

Duplicating the Behavior of the CLSID_DsPropertyPages Object

To duplicate the behavior of the [CLSID_DsPropertyPages](#) object

1. Enumerate the values in the [adminPropertyPages](#) or [shellPropertyPages](#) attribute for the display specifier for the object class. Each value is a string that contains a number, followed by a comma, followed by the string representation of the class identifier of the property page extension. For more information about the format of the property pages display specifier values, see [Registering the Property Page COM Object in a Display Specifier](#).
2. Convert each class identifier string into a [CLSID](#) using the [CLSIDFromString](#) function.
3. Sort the extension class identifiers by the number that precedes each class identifier string in the attribute value. If two numbers are identical, sort the class identifiers in the order that the attribute values are obtained from the Active Directory server.
4. Enumerate the extension class identifiers, creating an instance of each extension.
5. For each extension, in the order sorted above, call the extension's [IShellExtInit::Initialize](#) with the same information described in Step 4 of the Hosting an Active Directory Users and Computers Property Sheet procedure.
6. For each extension, in the order sorted above, call the extension's [IShellPropSheetExt::AddPages](#) with the same information described in Step 5 of the Hosting an Active Directory Users and Computers Property Sheet procedure.

If possible, use the [CLSID_DsPropertyPages](#) object to create the pages rather than do this manually. The [CLSID_DsPropertyPages](#) has been optimized and will correctly handle failure cases, such as when no display specifier is available for the current locale. Also, the [CLSID_DsPropertyPages](#) object may change in the future, which means your property sheets may not exactly match those displayed by the Active Directory Users and Computers MMC snap-in.

Special Programming Elements

Currently, the following programming elements are not defined in a published header file. To use these elements, you must define them yourself in the exact format shown in the particular reference page.

- [CFSTR_DS_PROPSHEETCONFIG](#)
- [PROPSHEETCFG](#)
- [WM_DSA_SHEET_CLOSE_NOTIFY](#)
- [WM_DSA_SHEET_CREATE_NOTIFY](#)
- [DSA_SEC_PAGE_INFO](#)

Example Code

The following C++ code example shows a safe way to define these elements that will continue to work even if these elements are defined in a published header file in the future.

```
#ifndef CFSTR_DS_PROPSHEETCONFIG
#define CFSTR_DS_PROPSHEETCONFIG_W L"DsPropSheetCfgClipFormat"
#define CFSTR_DS_PROPSHEETCONFIG_A "DsPropSheetCfgClipFormat"

#ifndef UNICODE
#define CFSTR_DS_PROPSHEETCONFIG CFSTR_DS_PROPSHEETCONFIG_W
#else
#define CFSTR_DS_PROPSHEETCONFIG CFSTR_DS_PROPSHEETCONFIG_A
#endif //UNICODE
#endif //CFSTR_DS_PROPSHEETCONFIG

#ifndef WM_AdSPROP_SHEET_CREATE
#define WM_AdSPROP_SHEET_CREATE (WM_USER + 1108)
#endif

#ifndef WM_DSA_SHEET_CREATE_NOTIFY
#define WM_DSA_SHEET_CREATE_NOTIFY (WM_USER + 6)
#endif

#ifndef WM_DSA_SHEET_CLOSE_NOTIFY
#define WM_DSA_SHEET_CLOSE_NOTIFY (WM_USER + 5)
#endif

#ifndef DSA_SEC_PAGE_INFO
typedef struct _DSA_SEC_PAGE_INFO
{
    HWND     hwndParentSheet;
    DWORD    offsetTitle;
    DSOBJECTNAMES dsObjectNames;
} DSA_SEC_PAGE_INFO, *PDSA_SEC_PAGE_INFO;
#endif //DSA_SEC_PAGE_INFO

#ifndef PROPSHEETCFG
typedef struct _PROPSHEETCFG
{
    LONG_PTR lNotifyHandle;
    HWND     hwndParentSheet;
    HWND     hwndHidden;
    WPARAM   wParamSheetClose;
} PROPSHEETCFG, *PPROPSHEETCFG;
#endif //PROPSHEETCFG
```

Administrative Notification Handlers

6/3/2022 • 3 minutes to read • [Edit Online](#)

The Microsoft Active Directory Users and Computers MMC snap-in provides a mechanism to enable components to receive notifications when the user deletes, renames, moves, or changes the properties of an object using the snap-in. The component that receives the notifications is known as a "notification handler".

This is useful when multiple objects are linked together and must exist within the same container. If one of the linked objects is moved, a notification is supplied to the notification handler and the notification handler can move the other linked objects to the same folder.

When one of the operations is performed and one or more notification handlers is installed, the Users and Computers snap-in will display a confirmation dialog box that lists the notification handlers and a check box for each handler. If the check box for a handler is selected, the handler is notified. If the check box is not selected, the handler is not notified.

Implementing a Notification Handler

A notification handler is a COM object implemented as an in-proc server. The notification handler must implement the [**IDsAdminNotifyHandler**](#) interface.

When an event occurs that will cause a notification, the Users and Computers snap-in enumerates the registered notification handlers and creates each one using the CLSID for the handler. After the handler is created, the snap-in calls the [**IDsAdminNotifyHandler::Initialize**](#) method. The **Initialize** method supplies the snap-in with the events the handler should receive.

If the event is one that should be sent to the notification handler, the snap-in calls the [**IDsAdminNotifyHandler::Begin**](#) method. The **Begin** method provides the handler with the event occurring, data about the object that the event is occurring on and, depending on the event, data about what the object will become. The **Begin** method also provides the snap-in with the text that should be displayed for the handler in the confirmation dialog box.

When the **Begin** method for each handler has been called, the snap-in displays the confirmation dialog box. The confirmation dialog box prompts the user to select which handlers will receive the notification. If the user presses the **No** push button in the confirmation dialog, none of the handlers are notified. If the user presses the **Yes** push button, each of the handlers selected in the confirmation dialog box receive the notification. The snap-in sends the notification to the handler by calling the [**IDsAdminNotifyHandler::Notify**](#) method.

After all of the handlers have been notified, the snap-in calls the [**IDsAdminNotifyHandler::End**](#) method. The **End** method is always called, even if the [**Notify**](#) method is not called.

Registering a Notification Handler in the Windows Registry

Like all COM servers, a notification handler must be registered in the Windows registry. The handler is registered under the following key:

HKEY_CLASSES_ROOT - CLSID - <CLSID>

<CLSID> is the string representation of the CLSID as produced by the [**StringFromCLSID**](#) function. Under the <CLSID> key, there is an **InProcServer32** key that identifies the object as a 32-bit in-proc server. Under the **InProcServer32** key, the location of the DLL is specified in the default value and the threading model is

specified in the **ThreadingModel** value. All notification handlers must use the **Apartment** threading model.

Registering a Notification Handler with an Active Directory Server

Within Active Directory Domain Services, notification handler registration is specific to one locale. If the notification handler applies to all locales, it must be registered in the **displaySpecifier** object in all of the locale subcontainers in the **DisplaySpecifiers** container. If the notification handler is localized for a certain locale, it is registered in the **displaySpecifier** object in that locale's subcontainer. For more information about the **DisplaySpecifiers** container and locales, see [Display Specifiers](#) and [DisplaySpecifiers Container](#).

Notification handlers are registered under the **dsUIAdminNotification** attribute in the **DS-UI-Default-Settings** container. This a multi-valued Unicode string value where each value requires the following format:

```
<order number>,<CLSID>
```

The "<order number>" is an unsigned number that represents the position of the handler in the confirmation dialog. When the confirmation dialog is displayed, the values are sorted using a comparison of each value's "<order number>". If more than one value has the same "<order number>", those handlers are displayed in the order they are read from the Active Directory server. A non-existing, that is, one not used by other values in the property, "<order number>" should be used if possible. There is no prescribed starting position, and gaps can appear in the "<order number>" sequence.

The "<CLSID>" is the string representation of the CLSID as produced by the [StringFromCLSID](#) function.

Distributing User Interface Components

6/3/2022 • 2 minutes to read • [Edit Online](#)

The COM components, applications and/or files used by a user interface extension can be distributed by creating a Windows Installer package and using a group policy to distribute the package. For more information about creating the Installer package, see [Windows Installer](#). For more information about group policies, see [Group Policy](#).

How Applications Should Use Display Specifiers

6/3/2022 • 3 minutes to read • [Edit Online](#)

To display Active Directory Domain Service objects, use display specifiers to obtain localized display data for class and attribute objects. This enables localized display names and icons to be used and avoids unnecessary localization of the display data.

Display Names

The **classDisplayName** and **attributeDisplayNames** properties of the display specifier objects for the appropriate locale should be used to obtain display text for class and attribute names. Do not use the **cn**, **classDisplayName** or **ldapDisplayName** properties of the **classSchema** or **attributeSchema** objects to obtain display text labels because these values are not localized. For more information about how to retrieve localized text for a class object, see the following sample code.

Icons

The **iconPath** property of the display specifier objects for the appropriate locale should be used to obtain the icon to display for a class object. For more information, see [Class Icons](#). If no localized icon is specified for a class object, a default icon should be displayed for the item.

Creating New Objects

When possible, use the appropriate object creation wizards to create new objects. For more information, see [Invoking Creation Wizards from Your Application](#).

The following code example shows how to obtain the display text for a class and an attribute of a class.

```
#include <atldbase.h>

/***********************/

GetClassDisplaySpecifierContainer()

/***********************/

HRESULT GetClassDisplaySpecifierContainer(LPWSTR pwszClass,
                                         LCID locale,
                                         IADs **ppads)
{
    if((NULL == pwszClass) || (NULL == ppads))
    {
        return E_INVALIDARG;
    }

    *ppads = NULL;

    // If no locale is specified, use the default system locale.
    if(0 == locale)
    {
        locale = GetSystemDefaultLCID();
        if(0 == locale)
        {
            return E_FAIL;
        }
    }
}
```

```

// Verify that it is a valid locale.
if(!IsValidLocale(locale, LCID_SUPPORTED))
{
    return E_INVALIDARG;
}

HRESULT hr;
IADs *padsRoot = NULL;

hr = ADsOpenObject( L"LDAP://rootDSE",
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION,
                    IID_IADs,
                    (void**)&padsRoot);

if(SUCCEEDED(hr))
{
    VARIANT var;

    VariantInit(&var);

    // Get the DN to the configuration container.
    hr = padsRoot->Get(CComBSTR(L"configurationNamingContext"), &var);

    if(SUCCEEDED(hr))
    {
        WCHAR wszPath[MAX_PATH * 2];

        // Build the string to bind to the container for the
        // specified locale in the DisplaySpecifiers container.
        swprintf_s(wszPath,
                   L"LDAP://cn=%s-Display,cn=%x,cn=DisplaySpecifiers,%s",
                   pwszClass,
                   locale,
                   var.bstrVal);

        VariantClear(&var);

        // Bind to the container.
        hr = ADsOpenObject( wszPath,
                            NULL,
                            NULL,
                            ADS_SECURE_AUTHENTICATION,
                            IID_IADs,
                            (void**)ppads);
    }
}

padsRoot->Release();
}

return hr;
}

*****  

GetClassDisplayLabel()  

*****  

HRESULT GetClassDisplayLabel(LPWSTR pwszClass,
                           LCID locale,
                           BSTR *pbstrClassLabel)
{
    if((NULL == pwszClass) || (NULL == pbstrClassLabel))
    {
        return E_INVALIDARG;
    }
}

```

```

}

*pbstrClassLabel = NULL;

HRESULT hr;
IADS *padsDispSpec;
hr = GetClassDisplaySpecifierContainer(pwszClass, locale, &padsDispSpec);
if(SUCCEEDED(hr))
{
    VARIANT var;

    VariantInit(&var);

    // Get the classDisplayName property value.
    hr = padsDispSpec->Get(CComBSTR(L"classDisplayName"), &var);

    if(SUCCEEDED(hr))
    {
        if(VT_BSTR == var.vt)
        {
            // Do not free the BSTR. The caller will handle it.
            *pbstrClassLabel = var.bstrVal;
        }
        else
        {
            VariantClear(&var);
            hr = E_FAIL;
        }
    }
    padsDispSpec->Release();
}

return hr;
}

*****  

GetAttributeDisplayLabel()  

*****  

HRESULT GetAttributeDisplayLabel(LPWSTR pwszClass,
                                 LPWSTR pwszAttribute,
                                 LCID locale,
                                 BSTR *pbstrAttributeLabel)
{
    if( (NULL == pwszClass) ||
        (NULL == pwszAttribute) ||
        (NULL == pbstrAttributeLabel))
    {
        return E_INVALIDARG;
    }

    *pbstrAttributeLabel = NULL;

    HRESULT hr;
    IADS *padsDispSpec;
    hr = GetClassDisplaySpecifierContainer(pwszClass, locale, &padsDispSpec);
    if(SUCCEEDED(hr))
    {
        VARIANT var;

        VariantInit(&var);

        // Get the attributeDisplayNames property values
        hr = padsDispSpec->GetEx(CComBSTR(L"attributeDisplayNames"), &var);

        if(SUCCEEDED(hr))

```

```

{
    LONG      lStart,
    lEnd,
    i;
    SAFEARRAY *psa;
    VARIANT   varItem;

    VariantInit(&varItem);

    psa = V_ARRAY(&var);

    // Get the lower and upper bound.
    hr = SafeArrayGetLBound(psa, 1, &lStart);
    hr = SafeArrayGetUBound(psa, 1, &lEnd);

    /*
    The attributeDisplayNames values take the form
    "<attribute name>,<display name>". Enumerate the values, looking
    for the one that begins with the specified attribute name.
    */
    for(i = lStart; i <= lEnd; i++)
    {
        hr = SafeArrayGetElement(psa, &i, &varItem);
        if(SUCCEEDED(hr))
        {
            WCHAR   wszTemp[MAX_PATH];

            wcsncpy_s(wszTemp,
                      V_BSTR(&varItem),
                      lstrlenW(pwszAttribute) + 1);

            if(0 == lstrcmpiW(pwszAttribute, wszTemp))
            {
                LPWSTR pwszDisplayLabel;

                /*
                The proper value was found. Now, parse the value, looking
                for the first comma, which delimits the attribute name
                from the display name.
                */
                for(    pwszDisplayLabel = V_BSTR(&varItem);
                     *pwszDisplayLabel;
                     pwszDisplayLabel = CharNextW(pwszDisplayLabel))
                {
                    if(',') == *pwszDisplayLabel)
                    {
                        /*
                        The delimiter was found. Set the string
                        pointer to the next character, which is the
                        first character of the display name.
                        */
                        pwszDisplayLabel = CharNextW(pwszDisplayLabel);
                        break;
                    }
                }

                if(*pwszDisplayLabel)
                {
                    // Copy the display name to the output.
                    *pbstrAttributeLabel = CComBSTR(pwszDisplayLabel).Detach();

                    hr = S_OK;
                }
            }
        }
    }

    /*
    Release the item variant because the break prevents
    it from getting released by the VariantClear call below.
    */
    VariantClear(&varItem);
}

```

```
        break;
    }

    VariantClear(&varItem);
}
}

padsDispSpec->Release();
}

return hr;
}
```

Debugging an Active Directory Extension

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Microsoft Active Directory directory service property sheet, context menu, and object creation wizard extensions documented in this topic are implemented as COM in-proc servers. That is, each extension is a DLL that runs in the context of the host process. To debug the extension, it is necessary to associate the extension with an application and run the application in a debugger.

Debugging Active Directory Extensions Displayed in the Windows Shell

Active Directory extensions displayed in the Windows shell are loaded in the context of the Explorer.exe process. These extensions can be debugged like a standard shell extension. For more information about debugging shell extensions, see [Debugging with the Shell](#).

Debugging Active Directory Extensions Displayed in the Active Directory MMC Snap-ins

Active Directory extensions displayed in the Active Directory administrative MMC snap-ins are loaded in the context of the Microsoft Management Console. To debug an extension, locate Mmc.exe on the local system and set the debugger to use this as the application for debugging. On most systems, Mmc.exe is located in the Windows system directory, for example, C:\WINNT\System32. Depending on the debugger, you may or may not have to set the extension DLL to also be loaded by the debugger. Many debuggers also allow you to attach the debugger to a running MMC process. For more information, see your debugger User's Guide.

It can be convenient to have MMC automatically load a specific snap-in. To do this, set the application arguments to the path and file name of an MSC file. This can either be a system-installed MSC file or one you create. An MSC file can be created by following these steps.

1. Run Mmc.exe.
2. Load the desired snap-in by selecting **File - Add/Remove Snap-in...** in the MMC menu and select the desired snap-in.
3. Save the MSC file by selecting **File - Save As...** in the MMC menu.

If you do not set a start-up MSC file, you must manually load the desired snap-in when you run the application in the debugger.

When the host application is run in the debugger, the debugger may display a warning message stating that the application being run does not contain any debug symbols. This is expected and can safely be ignored because you are actually debugging the DLL, not the host application.

In most cases, the extension will not be called until the user performs some action that causes the extension to be loaded and initialized. For example, if you are debugging a context menu extension displayed for user objects, the extension will not load until the first time the context menu for a user object is displayed.

You should now be able to set breakpoints and view debug output. If the extension does not appear to load, set a breakpoint in the extension's [DIIGetClassObject](#) function. If [DIIGetClassObject](#) is not called, the extension is probably not registered correctly.

When the debug is complete, exit MMC and the debugger should unload normally.

Managing Users

6/3/2022 • 2 minutes to read • [Edit Online](#)

User accounts are created and stored as objects in Active Directory Domain Services. These user objects represent users and computers. This section defines what users are and how they are used, and explains how to programmatically manage users in Active Directory Domain Services. This section discusses the following topics:

- [Users in Active Directory Domain Services](#)
- [Security Principals](#)
- [What is a User?](#)
- [Example Code for Binding to the Users Container](#)
- [User Object Attributes](#)
- [Creating a User](#)
- Deleting a user. A user is deleted in the same was as any other object in Active Directory Domain Services.
For more information about deleting objects, see [Creating and Deleting Objects in Active Directory Domain Services](#).
- [Enumerating Users](#)
- [Querying for Users](#)
- Moving users. A user is moved in the same was as any other Active Directory object. For more information about moving Active Directory objects, see [Moving Objects](#).
- [Managing Users on Member Servers and Windows 2000 Professional](#)

Users in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services includes a directory service that stores domain, user, user group, and security data.

On Windows NT 4.0 and earlier, you can use functions such as [NetUserAdd](#), [NetUserEnum](#), [NetUserDel](#), and so on, to manage users, user groups, and other network items. On Windows 2000 and later versions of Windows, ADSI provides uniform and secure access to these items and their properties. Be aware that ADSI provides a Windows NT 4.0 provider that enables you to use ADSI to manage user, user groups, and computers on Windows NT 4.0 systems. There are also providers for Windows Server 2008 Enterprise (Server Core installation), Microsoft Exchange 5.5, Microsoft Internet Information Server (IIS), Novell NetWare Directory Services (NDS), and Novell NetWare 3. That is, a single set of standardized methods for managing users and user groups for Windows NT, NDS, and NetWare 3.

In addition, Windows 2000 is a multi-master directory. That is, changes to users, user groups, and other data stored in the directory can be made at any domain controller. On Windows 2000, you are not required to locate the primary domain controller (PDC) and make user and user group changes on the PDC.

Windows 2000 also introduces a new hierarchical namespace within a domain called an organizational unit (OU). An OU can contain computers, users, user groups, and other network objects. Usually, an OU is used for the purpose of grouping things for administrative purposes, such as delegating administrative rights and assigning policies to the group as a single unit.

Domains, OUs, users, user groups, computers, and other network items are stored as objects in Active Directory Domain Services. In Windows 2000 and later versions of Windows, you still add users, user groups, and computers to a domain. However, you now have the option of adding these objects to an OU container or any other type of container that the object you want to add defines in its **classSchema** object's **possSuperiors** attribute; this is an attribute on an object's **classSchema** object and this attribute restricts what types of objects can contain that object.

Security Principals

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Windows 2000, a security principal is a user, group, or computer — an entity that the security system recognizes. This includes human users as well as autonomous processes. Strictly speaking, the security system cannot tell the difference between users who are logged in and processes running on the computer. It sees both as security principals with security principal names.

Users, groups, and computers are created and stored as objects in Active Directory Domain Services. There are also well-known security principals that represent special identities defined by the Windows 2000 security system, such as Everyone, Local System, Principal Self, Authenticated User, Creator Owner, and so on. Objects representing the well-known security principals, such as Anonymous Logon, are stored in the WellKnown Security Principals container beneath the Configuration container.

What is a User?

6/3/2022 • 2 minutes to read • [Edit Online](#)

User accounts are created and stored as objects in Active Directory Domain Services. User accounts can be used by human users or programs such as system services use to log on to a computer. When a user logs on, the system verifies the user's password by comparing it with data stored in the user's user object in the Active Directory server. If the password is authenticated, that is, the password presented matches the password stored in the user object, the system produces an access token. An access token is an object that describes the security context of a process or thread. The data in a token includes the security identity and group memberships of the user account associated with the process or thread. Every process executed on behalf of this user has a copy of this access token.

Each user or application that accesses resources in a Windows domain must have an account in the Active Directory server. Windows uses this user account to verify that the user or application has permission to use a resource.

A user account can be used to:

- Enable human users to log on to a computer and to access resources based on that user account's identity.
- Enable programs and services to run under a specific security context.
- Manage user access to shared resources such as objects and their properties, network shares, files, directories, printer queues, and so on.

Groups can contain members, which are references to users and other groups. Groups can also be used to control access to shared resources. When assigning permissions for resources, for example file shares, printers, and so on, administrators should assign those permissions to a group rather than to the individual users. The permissions are assigned once to the group, instead of several times to each individual user. This helps simplify the maintenance and administration of a network.

Users compared to Contacts

Both users and contacts can be used to represent human users. However, a user is a security principal; a contact is not.

A user can be used to enable a human user to log on and access shared resources.

A contact is used only for distribution list and email purposes. However, a contact can contain most of the data stored in a user object such as address, phone numbers, and so on because both user and contact are derived from the person **classSchema** object. A contact has no security context; therefore, a contact cannot be used to control access to shared resources and cannot be used to log on to a computer.

Users compared to Computers

The computer object class inherits from the user object class. A computer object represents a computer; however, the computer and the computer's local services often require access to the network and shared resources. When the computer accesses shared resources, not the user logged on to the computer, it needs an access token just as a human user logged on as a user does. When a computer accesses the network, it uses an access token that contains the security identifier for the computer's computer account and the groups that account is a member.

A service can run in the context of LocalSystem or a specific service account. On computers running Windows 2000, a service that runs in the context of the LocalSystem account uses the credentials of the computer.

Example Code for Binding to the User's Container

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ code example binds to the users container in the current domain and return and **IADsContainer** interface for the container. For more information about binding to well-known objects, see [Binding to Well-Known Objects Using WKGUID](#).

```
*****  
//  
//  GetUsersContainer()  
//  
//  Binds to the well-known Users container in the current domain  
//  with the current user credentials. GUID_USERS_CONTAINER_W is  
//  defined in NTDSAPI.H.  
//  
*****  
  
HRESULT GetUsersContainer(IADsContainer **ppContainer)  
{  
    if(NULL == ppContainer)  
    {  
        return E_INVALIDARG;  
    }  
  
    HRESULT hr;  
    IADs *pRoot;  
  
    *ppContainer = NULL;  
  
    // Bind to the rootDSE object.  
    hr = ADsOpenObject(L"LDAP://rootDSE",  
                       NULL,  
                       NULL,  
                       ADS_SECURE_AUTHENTICATION,  
                       IID_IADs,  
                       (LPVOID*)&pRoot);  
    if(SUCCEEDED(hr))  
    {  
        VARIANT var;  
  
        VariantInit(&var);  
  
        // Get the current domain DN.  
        hr = pRoot->Get(CComBSTR("defaultNamingContext"), &var);  
        if(SUCCEEDED(hr))  
        {  
            // Build the binding string.  
            LPWSTR pwszFormat = L"LDAP://<WKGUID=%s,%s>";  
            LPWSTR pwszPath;  
  
            pwszPath = new WCHAR[wcslen(pwszFormat) +  
                           wcslen(GUID_USERS_CONTAINER_W) +  
                           wcslen(var.bstrVal)];  
            if(NULL != pwszPath)  
            {  
                swprintf_s(pwszPath,  
                           pwszFormat,  
                           GUID_USERS_CONTAINER_W,  
                           var.bstrVal);  
  
                // Bind to the object.  
                hr = ADsOpenObject(pwszPath,
```

```
    m_hObject = ADSOpenObject(pwszPath,
                             NULL,
                             NULL,
                             ADS_SECURE_AUTHENTICATION,
                             IID_IADsContainer,
                             (LPVOID*)ppContainer);

    delete pwszPath;
}
else
{
    hr = E_OUTOFMEMORY;
}

VariantClear(&var);
}

pRoot->Release();
}

return hr;
}
```

User Object Attributes

6/3/2022 • 2 minutes to read • [Edit Online](#)

A user object has multiple attributes. This section documents key attributes used by Windows, administrative tools, and the Windows Address Book (WAB). It does not describe all attributes; many attributes are not used for the user object.

Some attributes are stored in the directory, such as [cn](#), [nTSecurityDescriptor](#), [objectGUID](#), and so on, and replicated to all domain controllers within a domain. A subset of these attributes is also replicated to the global catalog.

Non-replicated attributes are stored on each domain controller, but are not replicated elsewhere, such as [badPwdCount](#), [lastLogon](#), [lastLogoff](#), and so on. The non-replicated attributes are attributes that pertain to a particular domain controller. For example, [lastLogon](#) is the last date and time that the user network logon was validated by the particular domain controller that is returning the property.

A user object also has constructed attributes that are not stored in the directory, but are calculated by the domain controller, such as [canonicalName](#), [distinguishedName](#), [allowedAttributes](#), and so on.

Attributes for user objects are classified as:

Base Object Attributes

This category includes attributes required for all directory objects, such as [objectClass](#), [nTSecurityDescriptor](#), and so on.

Naming Attributes

This category includes attributes used to refer to or identify the object, such as [distinguishedName](#), [objectGUID](#), [objectSID](#), and so on. For more information about naming attributes for user objects, see [User Naming Attributes](#).

Security Attributes

This category includes attributes for logon and access control. For more information about security attributes for user objects, see [User Security Attributes](#).

Address Book Attributes

This category includes attributes for email and user data. For more information about address book attributes for user objects, see [User Address Book Attributes](#).

Application Specific Attributes

This category includes user-specific configuration data for specific applications.

For more information about reading and modifying attributes for a user object, see [Reading and Writing Attributes of Objects in Active Directory Domain Services](#).

For more information about the User class, including a complete list of the [mayContain](#) and [mustContain](#) attributes of the class, see [User](#).

Setting Passwords

The password for a user cannot be modified directly because this would involve sending an unencrypted

password over the network. To set the password for a user, it is necessary to use the [IADsUser.ChangePassword](#) or [IADsUser.SetPassword](#) method. The [IADsUser.ChangePassword](#) method is used when the application is allowing the user to change their own password. The [IADsUser.SetPassword](#) method is used when the application enables an administrator to reset a password.

User Naming Attributes

6/3/2022 • 4 minutes to read • [Edit Online](#)

User naming attributes identify user objects, such as logon names and IDs used for security purposes. The **cn**, **name**, and **distinguishedName** attributes are examples of user naming attributes. A user object is a security principal object, so it also includes the following user naming attributes:

- **userPrincipalName** — the logon name for the user
- **objectGUID** — the unique identifier of a user
- **sAMAccountName** — a logon name that supports previous version of Windows
- **objectSid** — security identifier (SID) of the user
- **sidHistory** — the previous SIDs for the user object

NOTE

You can view and manage these attributes using the Active Directory User and Computers MMC snap-in, which is available in the [Remote Server Administration Tools \(RSAT\)](#).

userPrincipalName

The **userPrincipalName** attribute is the logon name for the user. The attribute consists of a user principal name (UPN), which is the most common logon name for Windows users. Users typically use their UPN to log on to a domain. This attribute is an indexed string that is single-valued.

A UPN is an Internet-style login name for a user based on the Internet standard RFC 822. The UPN is shorter than a distinguished name and easier to remember. By convention, this should map to the user's email name. The point of the UPN is to consolidate the email and logon namespaces so that the user only needs to remember a single name.

UPN Format

A UPN consists of a UPN prefix (the user account name) and a UPN suffix (a DNS domain name). The prefix is joined with the suffix using the "@" symbol. For example, "someone@example.com". A UPN must be unique among all security principal objects within a directory forest. This means the prefix of a UPN can be reused, just not with the same suffix.

A UPN suffix has the following restrictions:

- It must be the DNS name of a domain, but does not need to be the name of the domain that contains the user.
- It must be the name of a domain in the current domain forest, or an alternate name listed in the **upnSuffixes** attribute of the Partitions container within the Configuration container.

UPN Management

A UPN can be assigned, but is not required, when a user account is created. When a UPN is created, it is unaffected by changes to other attributes of the user object such as the user being renamed or moved. This allows the user to keep the same login name if a directory is restructured. However, an administrator can change a UPN. When you create a new user object, you should check the local domain and the global catalog for the proposed name to ensure it does not already exist.

When a user uses a UPN to log on to a domain, the UPN is validated by searching the local domain and then the global catalog. If the UPN is not found in the global catalog, the logon attempt fails.

objectGUID

The **objectGUID** attribute is the unique identifier of a user. The attribute is a single-valued 128-bit Globally Unique Identifier (GUID), and is stored as an [ADS_OCTET_STRING](#) structure. The GUID is created by the Active Directory server when a user object is created.

Because an object's distinguished name changes if the object is renamed or moved, the distinguished name is not a reliable identifier of an object. In Active Directory Domain Services, an object's **objectGUID** attribute never changes, even if the object is renamed or moved. You can retrieve the string form of the **objectGUID** using the **GUID** property method in [IADs Property Methods](#).

sAMAccountName

The **sAMAccountName** attribute is a logon name used to support clients and servers from previous version of Windows, such as Windows NT 4.0, Windows 95, Windows 98, and LAN Manager. The logon name must be 20 or fewer characters and be unique among all security principal objects within the domain.

objectSid

The **objectSid** attribute is the security identifier (SID) of the user. The SID is used by the system to identify a user and their group memberships during interactions with Windows security. The attribute is single-valued. The SID is a unique binary value used to identify the user as a security principal.

The SID is set by the system when the user is created. Each user has a unique SID issued by a Windows domain and stored in the **objectSid** attribute of the user object in the directory. Each time a user logs on, the system retrieves the user's SID from the directory and places it in the user's access token. The user's SID is also used to retrieve the SIDs for the groups of which the user is a member and places them in the user's access token. When a SID has been used as the unique identifier for a user or group, it cannot be used again to identify another user or group.

sIDHistory

The **sIDHistory** attribute contains the previous SIDs for the user object. This is a multi-valued attribute. A user object has previous SIDs if the user was moved to another domain. Whenever a user object is moved to a new domain, a new SID is created and assigned the **objectSid** attribute, and the previous SID is added to the **sIDHistory** attribute.

Related topics

[User Object Attributes](#)

User Security Attributes

6/3/2022 • 10 minutes to read • [Edit Online](#)

In addition to naming properties for user objects, for example, **objectGUID**, **objectSid**, **cn**, **distinguishedName**, and so on, there are other security properties used for logon, network access, and access control. These properties are used by the Windows 2000 security system. These properties can be viewed and managed by the Active Directory User and Computers snap-in.

accountExpires

The **accountExpires** attribute specifies when an account expires. This value is stored as a large integer that represents the number of 100-nanosecond intervals since January 1, 1601 (UTC). A value of **TIMEQ_FOREVER** (defined in Lmaccess.h) indicates that an account never expires.

altSecurityIdentities

The **altSecurityIdentities** attribute is a multi-valued attribute that contains mappings for X.509 certificates or external Kerberos user accounts to this user for the purpose of authentication. Various security packages, including Public Key authentication package and Kerberos, use this data to authenticate users when they present the alternative form of identification such as certificate, UNIX Kerberos ticket, and so on. Build a Windows 2000 token based on the corresponding user account such that they can access system resources.

For X.509 certificates, the values should be the Issuer and Subject names in 509v3 certificates, issued by an external public certification authority, that map to the user account used to find an account for authentication. The SSL (Schannel) package uses the following syntax: X509:<somecertinfotype>somecertinfo. For example, the following value specifies the issuer DN "<I>" with the DN "C=US,O=InternetCA,CN=APublicCertificateAuthority" and the subject DN "<S>" with the DN "C=US,O=Fabrikam,OU=Sales,CN=Jeff Smith".

```
X509:<I>C=US,O=InternetCA,CN=APublicCertificateAuthority<S>C=US,O=Fabrikam,OU=Sales,CN=Jeff Smith
```

Be aware that "<S>" or "<I>" and "<S>" are supported. Having only "<I>" is not supported. Applications should not modify the values within "<I>" or "<S>" because partial DN matching is not supported.

For external Kerberos accounts, the values should be the Kerberos account name. The Kerberos package uses the following syntax: "Kerberos:MITaccountname". For example, the following is the value for an account at Fabrikam.com:

```
Kerberos:Jeff.Smith@Fabrikam.com
```

badPasswordTime

Non-replicated. The **badPasswordTime** attribute specifies the last time the user attempted to log on to the account using an incorrect password. This value is stored as a large integer that represents the number of 100-nanosecond intervals since January 1, 1601 (UTC). This attribute is maintained separately on each domain controller in the domain. A value of zero means that the last bad password time is unknown. To get an accurate value for the user's last bad password time in the domain, each domain controller in the domain must be queried and the largest value should be used.

badPwdCount

Non-replicated. The **badPwdCount** attribute specifies the number of times the user attempted to log on to the account using an incorrect password. This attribute is maintained separately on each domain controller in the domain. A value of 0 indicates that the value is unknown. To get an accurate value for the user's total bad password attempts in the domain, each domain controller in the domain must be queried and the sum of the values should be used.

codePage

The **codePage** attribute specifies the code page for the user's chosen language. This value is not used by Windows 2000.

countryCode

The **countryCode** attribute specifies the country/region code for the user's language. This value is not used by Windows 2000.

homeDirectory

The **homeDirectory** attribute specifies the path of the home directory for the user. The string can be null.

If **homeDrive** is set and specifies a drive letter, **homeDirectory** should be a UNC path. The path must be a network UNC path of the form \\server\share\directory. This value can be a null string.

If **homeDrive** is not set, **homeDirectory** should be a local path, for example, C:\mylocaldir.

homeDrive

The **homeDrive** attribute specifies the drive letter to which to map the UNC path specified by **homeDirectory**. The drive letter must be specified in the following form:

```
<drive letter>:
```

where "<drive letter>" is the letter of the drive to map. For example:

```
z:
```

If this attribute is not set, the **homeDirectory** should be a local path, for example, C:\mylocaldir.

lastLogoff

Non-replicated. The **lastLogoff** attribute specifies when the last logoff occurred. This value is stored as a large integer that represents the number of 100-nanosecond intervals since January 1, 1601 (UTC). The high part of this large integer corresponds to the **dwHighDateTime** member of the **FILETIME** structure and the low part corresponds to the **dwLowDateTime** member of the **FILETIME** structure. This attribute is maintained separately on each domain controller in the domain. A value of zero means that the last logoff time is unknown. To get an accurate value for the user's last logoff in the domain, each domain controller in the domain must be queried and the largest value should be used.

lastLogon

Non-replicated. The **lastLogon** attribute specifies when the last logon occurred. This value is stored as a large integer that represents the number of 100-nanosecond intervals since January 1, 1601 (UTC). The high part of this large integer corresponds to the **dwHighDateTime** member of the **FILETIME** structure and the low part corresponds to the **dwLowDateTime** member of the **FILETIME** structure. This attribute is maintained separately on each domain controller in the domain. A value of zero means that the last logon time is unknown. To get an accurate value for the user's last logon in the domain, each domain controller in the domain must be queried and the largest value should be used.

ImPwdHistory

The **ImPwdHistory** attribute is the password history of the user in LAN Manager (LM) one-way format (OWF). The LM OWF is used for compatibility with LAN Manager 2.x clients, Windows 95, and Windows 98. This attribute is used only by the operating system. Be aware that you cannot derive the plaintext password from the OWF form of the password.

logonCount

Non-replicated. The **logonCount** attribute counts the number of successful times that the user tried to log on to this account. This attribute is maintained on each domain controller in the domain. A value of 0 indicates that the value is unknown. To get an accurate value for the user's total number of successful logon attempts in the domain, each domain controller in the domain must be queried and the sum of the values should be used.

mail

The **mail** attribute is a single-valued attribute that contains the SMTP address for the user, for example, "jeff@Fabrikam.com".

maxStorage

The **maxStorage** attribute specifies the maximum amount of hard-disk drive space that the user can use. Use the **USER_MAXSTORAGE_UNLIMITED** (defined in Lmaccess.h) value to use all available disk space.

memberOf

The **memberOf** attribute is a multi-valued attribute that contains groups of which the user is a direct member, except for the primary group, which is represented by the **primaryGroupId**. Group membership is dependent on the domain controller (DC) from which this attribute is retrieved:

- At a DC for the domain that contains the user, **memberOf** for the user is complete with respect to membership for groups in that domain; however, **memberOf** does not contain the user's membership in domain local and global groups in other domains.
- At a GC server, **memberOf** for the user is complete with respect to all universal group memberships.

If both conditions are true for the DC, both sets of data are contained in **memberOf**.

Be aware that this attribute lists the groups that contain the user in their member attribute—it does not contain the recursive list of nested predecessors. For example, if user O is a member of group C and group B and group B were nested in group A, the **memberOf** attribute of user O would list group C and group B, but not group A.

This attribute is not stored—it is a computed back-link attribute.

ntPwdHistory

The **ntPwdHistory** attribute is the password history of the user in Windows NT one-way format (OWF). Windows 2000 uses the Windows NT OWF. This attribute is used only by the operating system. Be aware that you cannot derive the plaintext password back from the OWF form of the password.

otherMailbox

The **otherMailbox** attribute is a multi-valued attribute that contains other additional mail addresses in a form, for example, "CCMAIL: JeffSmith".

PasswordExpirationDate

The password expiration date is not an attribute on the user object. It is a calculated value based on the sum of **pwdLastSet** for the user and **maxPwdAge** of the user's domain. To get the password expiration date, get the **IADsUser.PasswordExpirationDate** property. You cannot modify this attribute for a user; instead, set the **IADsDomain.MaxPasswordAge** property to change the setting for the domain.

primaryGroupId

The **primaryGroupId** attribute is a single-valued attribute that contains the **primaryGroupToken** of the group that is the primary group of the object. The primary group of the object is not included in the **memberOf** attribute. For example, by default, the primary group of a user object is the **primaryGroupToken** of the Domain Users group, but the Domain Users group is not part of the user object's **memberOf** attribute.

profilePath

The **profilePath** attribute specifies a path to the user's profile. This value can be a null string, a local absolute path, or a UNC path.

pwdLastSet

The **pwdLastSet** attribute specifies when the password was last changed. This value is stored as a large integer that represents the number of 100-nanosecond intervals since January 1, 1601 (UTC).

The system uses the value of this attribute and the **maxPwdAge** attribute of the domain that contains the user object to calculate the password expiration date. That is, the sum of **pwdLastSet** for the user and **maxPwdAge** of the user's domain.

This attribute controls whether the user must change the password when the user logs on next. If **pwdLastSet** is zero, the default, the user must change the password at next logon. The value -1 indicates that the user is not required to change the password at next logon. The system sets this value to -1 after user has set the password.

sAMAccountType

The **sAMAccountType** attribute specifies an integer that represents the account type. This is set by the operating system when the object is created.

scriptPath

The **scriptPath** attribute specifies the path of the user's logon script, .cmd, .exe, or .bat file. The string can be null.

unicodePwd

The **unicodePwd** attribute is the user password.

To set the user password, use the **IADsUser.ChangePassword** method, if your script or application enables the user to change his/her own password, or **IADsUser.SetPassword** method, if your script or application is allowing an administrator to reset a password.

The password of the user in Windows NT one-way format (OWF). Windows 2000 uses the Windows NT OWF. This attribute is used only by operating system. Be aware that you cannot derive the plaintext password back from the OWF form of the password.

userAccountControl

The **userAccountControl** attribute specifies flags that control password, lockout, disable/enable, script, and home directory behavior for the user. This attribute also contains a flag that indicates the account type of the object. The user object usually has the UF_NORMAL_ACCOUNT set.

The following flags are defined in Lmaccess.h.

FLAG	DESCRIPTION
UF_SCRIPT	The logon script executed. This value must be set for LAN Manager 2.0 or Windows NT.
UF_ACCOUNTDISABLE	The user account is disabled.

FLAG	DESCRIPTION
UF_HOMEDIR_REQUIRED	The home directory is required. This value is ignored in Windows NT and Windows 2000.
UF_PASSWD_NOTREQD	No password is required.
UF_PASSWD_CANT_CHANGE	The user cannot change the password.
UF_LOCKOUT	The account is currently locked. This value can be cleared to unlock a previously locked account. This value cannot be used to lock a previously locked account.
UF_DONT_EXPIRE_PASSWD	Represents the password, which should never expire on the account.

The following flags describe the account type. Only one value can be set. You cannot change the account type.

FLAG	DESCRIPTION
UF_NORMAL_ACCOUNT	This is a default account type that represents a typical user.
UF_TEMP_DUPLICATE_ACCOUNT	This is an account for users whose primary account is in another domain. This account provides user access to this domain, but not to any domain that trusts this domain. The User Manager refers to this account type as a local user account.
UF_WORKSTATION_TRUST_ACCOUNT	This is a computer account for a Windows NT Workstation/Windows 2000 Professional or Windows NT Server/Windows 2000 Server that is a member of this domain.
UF_SERVER_TRUST_ACCOUNT	This is a computer account for a Windows NT Backup Domain Controller that is a member of this domain.
UF_INTERDOMAIN_TRUST_ACCOUNT	This is a permit to trust account for a Windows NT domain that trusts other domains.

userCertificate

The **userCertificate** attribute is a multi-valued attribute that contains the DER-encoded X509v3 certificates issued to the user. Be aware that this attribute contains the public key certificates issued to this user by Microsoft Certificate Service.

userSharedFolder

The **userSharedFolder** attribute specifies a UNC path to the user's shared documents folder. The path must be a network UNC path of the form \\server\share\directory. This value can be a null string.

userWorkstations

The **userWorkstations** attribute is a single-valued attribute that contains the NetBIOS names of the workstations from which the user can log on to. Each NetBIOS name is separated by a comma.

If no values are set, this indicates that there is no restriction. To disable logons from all workstations to this account, set the UF_ACCOUNTDISABLE value (defined in Lmaccess.h) in [userAccountControl](#) attribute.

User Address Book Attributes

6/3/2022 • 3 minutes to read • [Edit Online](#)

The address book attributes are used to provide supplementary identification and information for a user. Their content is defined by the user and the user's organization. These attributes can be viewed and managed by the Active Directory User and Computers snap-in or the Windows Address Book (WAB).

ATTRIBUTE	DESCRIPTION
c	The country/region in the user's address. The country/region is represented as the two-character country/region code based on ISO-3166. For the valid codes, see Values for countryCode .
co	The country/region in which the user is located.
notes	A comment. This string can be a null string.
department	The name for the department in which the user works.
description	The description to display for the user.
displayName	The name displayed in the address book for a particular user. This is usually the combination of the user's first name, middle initial, and last name.
directReports	The list of users that directly report to the user. The users listed as reports are those that have the manager attribute set to this user. Each item in the list is a linked reference to the object that represents the user; therefore, the Active Directory server automatically updates this attribute when a user's manager attribute adds or removes this user as a manager. The items are represented as distinguished names.
facsimileTelephoneNumber	The telephone number of the user's business fax machine.
givenName	The given name (first name) of the user.
homePhone	The primary home telephone number for the user.
initials	The initials for parts of the user's full name. This may be used as the middle initial in the Windows Address Book.
ipPhone	Used by Telephony.
l	The locality, such as the town or city, in the user's address.

ATTRIBUTE	DESCRIPTION
managedObjects	The list of objects that are managed by the user. The objects listed are those that have the managedBy attribute set to this user. Each item in the list is a linked reference to the managed object; therefore, the Active Directory server automatically updates the managedObjects attribute when an object's managedBy attribute adds or removes this user as its manager. The items are represented as distinguished names.
manager	The user who is the user's manager. The manager's user object contains a directReports attribute that contains references to all user objects that have their manager attribute set to the manager's user object.
mobile	The primary cellular telephone number for the user.
otherFacsimileTelephoneNumber	The list of telephone numbers of alternate fax machines for the user.
otherIpPhone	Used by Telephony.
otherMobile	The list of alternate cellular telephone numbers for the user.
otherPager	The list of alternate pager telephone numbers for the user.
otherTelephone	The list of alternate business telephone numbers for the user.
pager	The primary pager telephone number for the user.
physicalDeliveryOfficeName	The office location in the user's place of business.
postalAddress	The user's postal address.
postalCode	The postal code for the user's postal address. The postal code is specific to the user's country/region. In the United States of America, this attribute contains the ZIP code.
postOfficeBox	The number or identifier of the user's post office box.
sn	The user's surname (family name or last name).
st	The state or province in the user's address.
streetAddress	The street address of the user's place of business.
telephoneNumber	The primary telephone number of the user's place of business.
title	The user's job title. This attribute is commonly used to indicate the formal job title, such as Senior Programmer, rather than occupational class, such as programmer. It is not typically used for "suffix" titles such as Esq. or DDS. Examples: Managing Director, Programmer II, Associate Professor, and Development Lead.

ATTRIBUTE	DESCRIPTION
url	The list of URLs for the user's alternate webpages.
wWWHomePage	The URL for the user's primary webpage.

Creating a User

6/3/2022 • 3 minutes to read • [Edit Online](#)

To create a user in Active Directory Domain Services, create a user object in the domain container of the domain where you want to place the user. Users can be created at the root of the domain, within an organizational unit, or within a container.

When you create a user object, you must also set the attributes, listed in the following table, to set the object as a legal user that is recognized by Active Directory Domain Services and the Windows Security system.

ATTRIBUTE	DESCRIPTION
cn	Specifies the name of the user object in the directory. This will be the object's relative distinguished name (RDN).
sAMAccountName	Specifies a string that is the name used to support clients and servers from a previous version of Windows. The sAMAccountName should be less than 20 characters to support clients from a previous version of Windows. The sAMAccountName must be unique among all security principal objects within the domain. You should perform a query against the domain to verify that the sAMAccountName is unique within the domain. sAMAccountName is an optional attribute. The server will create a random sAMAccountName value if none is specified.

You can also set other attributes. The following user attributes are set with default values if you do not explicitly set them at creation time.

ATTRIBUTE	DESCRIPTION
accountExpires	Specifies when the account will expire. The default is TIMEQ_FOREVER , which indicates that the account will never expire.
nTSecurityDescriptor	A security descriptor is created based on specific rules. For more information, see How Security Descriptors are Set on New Directory Objects .
objectCategory	Specifies the user category. The default is "Person".
name	Specifies the user name. The default is the value set for cn .
pwdLastSet	Specifies when the user last set the password. The default is zero, which indicates that the user must change the password at next logon.

ATTRIBUTE	DESCRIPTION
userAccountControl	<p>Contains values that determine several logon and account features for the user.</p> <p>By default, the following flags are set:</p> <ul style="list-style-type: none"> • UF_ACCOUNTDISABLE - The account is disabled. • UF_PASSWD_NOTREQD - No password is required. • UF_NORMAL_ACCOUNT - Default account type that represents a typical user.
memberOf	Specifies the group or groups that the user is a direct member of. The default is "Domain Users".

A user is created by binding to the desired container and then using one of the following methods. The [cn](#) and [sAMAccountName](#) attributes must be set before the user is committed to the server.

METHOD	DESCRIPTION
IADsContainer.Create	The cn attribute is taken from the <i>bstrRelativeName</i> parameter. The new user must be committed by calling IADs.SetInfo or the object will not be created. For more information, see Example Code for Creating a User .
IDirectoryObject::CreateDSObject	The cn attribute is taken from the <i>pszRDNName</i> parameter. The new user is committed when CreateDSObject is called. For more information, see Example Code for Creating a User .
DirectoryEntries.Add	The cn attribute is taken from the <i>name</i> parameter. The new user object must be committed by calling DirectoryEntry.CommitChanges or the object will not be created. For more information, see Adding Directory Objects .

The new user must be committed to the server before any attributes other than [cn](#) and [sAMAccountName](#) can be modified. This is because the user account does not actually exist until the user is committed. If an attribute is retrieved or modified for an object that does not exist on the server, an error will occur. This includes calling the [IADsUser.SetPassword](#) method. For example, the following sequence would be followed when creating a user with [IADsContainer.Create](#):

1. Call [IADsContainer.Create](#) to create the user in the local cache with the specified [cn](#).
2. Set the [sAMAccountName](#) attribute to the desired value with the [IADs.Put](#) method.
3. Now modify other attributes, such as [userAccountControl](#). This restriction also applies to the ADSI properties, such as [IADsUser.AccountDisabled](#), and methods such as [IADsUser.SetPassword](#).
4. Call [IADs.SetInfo](#) to commit the new user to the server.

When a new user account is created, it is disabled by default. The account must be enabled manually or programmatically. To programmatically enable a user account, remove the **ADS_UF_ACCOUNTDISABLE** flag from the [userAccountControl](#) attribute.

When a new user account is created, the [userAccountControl](#) attribute for the account automatically has the **UF_PASSWD_NOTREQD** flag set, which indicates that no password is required for the account. If the security policies of the domain that the account is created in requires a password for all user accounts, then the **UF_PASSWD_NOTREQD** flag must be removed from the [userAccountControl](#) attribute for the account.

Related topics

Example Code for Creating a User

Example Code for Creating a User

6/3/2022 • 4 minutes to read • [Edit Online](#)

This topic includes code examples that create a user in a domain controlled by Active Directory Domain Services.

```
Const ADS_UF_SCRIPT = &H1
Const ADS_UF_ACCOUNTDISABLE = &H2
Const ADS_UF_HOMEDIR_REQUIRED = &H8
Const ADS_UF_LOCKOUT = &H10
Const ADS_UF_PASSWD_NOTREQD = &H20
Const ADS_UF_PASSWD_CANT_CHANGE = &H40
Const ADS_UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED = &H80
Const ADS_UF_TEMP_DUPLICATE_ACCOUNT = &H100
Const ADS_UF_NORMAL_ACCOUNT = &H200
Const ADS_UF_INTERDOMAIN_TRUST_ACCOUNT = &H800
Const ADS_UF_WORKSTATION_TRUST_ACCOUNT = &H1000
Const ADS_UF_SERVER_TRUST_ACCOUNT = &H2000
Const ADS_UF_DONT_EXPIRE_PASSWD = &H10000
Const ADS_UF_MNS_LOGON_ACCOUNT = &H20000
Const ADS_UF_SMARTCARD_REQUIRED = &H40000
Const ADS_UF_TRUSTED_FOR_DELEGATION = &H80000
Const ADS_UF_NOT_DELEGATED = &H100000
Const ADS_UF_USE_DES_KEY_ONLY = &H200000
Const ADS_UF_DONT_REQUIRE_PREAUTH = &H400000
Const ADS_UF_PASSWORD_EXPIRED = &H800000
Const ADS_UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION = &H1000000

Public Sub CreateUser(strName As String,
                     strSAMAccountName As String,
                     strInitialPassword As String)
    Dim objRootDSE As IADS
    Dim objUsers As IADsContainer
    Dim objNewUser As IADsUser

    On Error Resume Next

    ' Bind to the rootDSE object.
    Set objRootDSE = GetObject("LDAP://rootDSE")
    If (Err.Number <> 0) Then
        Exit Sub
    End If

    ' Bind to the Users folder in the domain.
    Set objUsers = GetObject("LDAP://CN=Users," &
                            objRootDSE.Get("defaultNamingContext"))
    If (Err.Number <> 0) Then
        Exit Sub
    End If

    ' Create the user object.
    Set objNewUser = objUsers.Create("user", "CN=" + strName)
    If (Err.Number <> 0) Then
        Exit Sub
    End If

    ' Set the sAMAccountName property.
    objNewUser.Put "sAMAccountName", strSAMAccountName
    If (Err.Number <> 0) Then
        Exit Sub
    End If
```

```

' Commit the new user.
objNewUser.SetInfo
If (Err.Number <> 0) Then
    Exit Sub
End If

' Set the initial password. This must be performed after
' SetInfo is called because the user object must
' already exist on the server.
objNewUser.SetPassword strInitialPassword
If (Err.Number <> 0) Then
    Exit Sub
End If

' Set the pwdLastSet property to zero, which forces the
' user to change their password at next log on.
objNewUser.Put "pwdLastSet", 0
If (Err.Number <> 0) Then
    Exit Sub
End If

' To enable the user account, remove the
' ADS_UF_ACCOUNTDISABLE flag from the userAccountControl
' property. Also, remove the ADS_UF_PASSWD_NOTREQD and
' ADS_UF_DONT_EXPIRE_PASSWD flags from the
' userAccountControl property.
userActCtrl = objNewUser.Get("userAccountControl")
userActCtrl = userActCtrl And Not (ADS_UF_ACCOUNTDISABLE +
                                    ADS_UF_PASSWD_NOTREQD +
                                    ADS_UF_DONT_EXPIRE_PASSWD)
objNewUser.Put "userAccountControl", userActCtrl
If (Err.Number <> 0) Then
    Exit Sub
End If

' Commit the updated properties.
objNewUser.SetInfo
End Sub

```

```

//*****
// CreateUserFromADs()
//*****
HRESULT CreateUserFromADs(LPCWSTR pwszContainerDN,
                          LPCWSTR pwszName,
                          LPCWSTR pwszSAMAccountName,
                          LPCWSTR pwszInitialPassword)
{
    HRESULT hr;

    // Build the DN of the container.
    CComBSTR sbstrADsPath = "LDAP://";
    sbstrADsPath += pwszContainerDN;

    IADsContainer *pUsers = NULL;

    // Bind to the container.
    hr = ADsGetObject(sbstrADsPath,
                      IID_IADsContainer,
                      (LPVOID*)&pUsers);
    if(SUCCEEDED(hr))
    {
        IDispatch *pDisp = NULL;

        CComBSTR sbstrName = "CN=";

```

```

sbstrName += pwszName;

// Create the new object in the User folder.
hr = pUsers->Create(CComBSTR("user"), sbstrName, &pDisp);

if(SUCCEEDED(hr))
{
    IADsUser *padsUser = NULL;

    // Get the IADs interface.
    hr = pDisp->QueryInterface(IID_IADsUser,
                                (void**) &padsUser);

    if(SUCCEEDED(hr))
    {
        CComBSTR sbstrProp;
        /*
        The sAMAccountName property is required on operating system
        versions prior to Windows Server 2003.
        The Windows Server 2003 operating system will create a
        sAMAccountName value if one is not specified.
        */
        CComVariant svar;
        svar = pwszSAMAccountName;
        sbstrProp = "sAMAccountName";
        hr = padsUser->Put(sbstrProp, svar);

        /*
        Commit the new user to persistent memory.
        The user does not exist until this is called.
        */
        hr = padsUser->SetInfo();

        /*
        Set the initial password. This must be done after
        SetInfo is called because the user object must
        already exist on the server.
        */
        hr = padsUser->SetPassword(CComBSTR(pwszInitialPassword));

        /*
        Set the pwdLastSet property to zero, which forces the
        user to change the password the next time they log on.
        */
        sbstrProp = "pwdLastSet";
        svar = 0;
        hr = padsUser->Put(sbstrProp, svar);

        /*
        Enable the user account by removing the
        ADS_UF_ACCOUNTDISABLE flag from the userAccountControl
        property. Also, remove the ADS_UF_PASSWD_NOTREQD and
        ADS_UF_DONT_EXPIRE_PASSWD flags from the
        userAccountControl property.
        */
        svar.Clear();
        sbstrProp = "userAccountControl";
        hr = padsUser->Get(sbstrProp, &svar);
        if(SUCCEEDED(hr))
        {
            svar = svar.lVal & ~(ADS_UF_ACCOUNTDISABLE |
                                ADS_UF_PASSWD_NOTREQD |
                                ADS_UF_DONT_EXPIRE_PASSWD);

            hr = padsUser->Put(sbstrProp, svar);
            hr = padsUser->SetInfo();
        }
        hr = padsUser->put_AccountDisabled(VARIANT_FALSE);
    }
}

```

```

        hr = padsUser->SetInfo();

        padsUser->Release();
    }

    pDisp->Release();
}

pUsers->Release();
}

return hr;
}

```

```

//*****
// CreateUserFromDirObject()
//*****
//*****

HRESULT CreateUserFromDirObject(LPCWSTR pwszContainerDN,
                                LPCWSTR pwszName,
                                LPCWSTR pwszSAMAccountName,
                                LPCWSTR pwszInitialPassword)
{
    HRESULT hr;

    // Build the DN of the container.
    CComBSTR sbstrADsPath = "LDAP://";
    sbstrADsPath += pwszContainerDN;

    IDirectoryObject *pdoUsers = NULL;

    // Bind to the container.
    hr = ADsGetObject(sbstrADsPath,
                      IID_IDirectoryObject,
                      (LPVOID*)&pdoUsers);
    if(SUCCEEDED(hr))
    {
        IDispatch *pDisp;
        ADS_ATTR_INFO rgAttrInfo[3];

        // Setup the objectClass property.
        ADSVALUE classValue;
        classValue.dwType = ADSTYPE_CASE_IGNORE_STRING;
        classValue.CaseIgnoreString = L"User";
        rgAttrInfo[0].pszAttrName = L"objectClass";
        rgAttrInfo[0].dwControlCode = ADS_ATTR_UPDATE;
        rgAttrInfo[0].dwADsType = ADSTYPE_CASE_IGNORE_STRING;
        rgAttrInfo[0].pADsValues = &classValue;
        rgAttrInfo[0].dwNumValues = 1;

        /*
        The sAMAccountName property is required on operating system versions
        prior to Windows Server 2003.
        The Windows Server 2003 operating system will create a
        sAMAccountName value if one is not specified.
        */
        ADSVALUE sAMValue;
        sAMValue.dwType = ADSTYPE_CASE_IGNORE_STRING;
        sAMValue.CaseIgnoreString =
            (ADS_CASE_IGNORE_STRING)pwszSAMAccountName;
        rgAttrInfo[1].pszAttrName = L"sAMAccountName";
        rgAttrInfo[1].dwControlCode = ADS_ATTR_UPDATE;
        rgAttrInfo[1].dwADsType = ADSTYPE_CASE_IGNORE_STRING;
        rgAttrInfo[1].pADsValues = &sAMValue;
        rgAttrInfo[1].dwNumValues = 1;
    }
}

```

```

/*
Set the initial userAccountControl attribute so that
the user is created as an enabled account and a
password is required.
*/
ADSVALUE userAccountControlValue;
userAccountControlValue.dwType = ADSTYPE_INTEGER;
userAccountControlValue.Integer = ADS_UF_NORMAL_ACCOUNT;
rgAttrInfo[2].pszAttrName = L"userAccountControl";
rgAttrInfo[2].dwControlCode = ADS_ATTR_UPDATE;
rgAttrInfo[2].dwADsType = ADSTYPE_INTEGER;
rgAttrInfo[2].pADsValues = &userAccountControlValue;
rgAttrInfo[2].dwNumValues = 1;

CComBSTR sbstrName = "CN=";
sbstrName += pwszName;

/*
Create the object in the Users container with the specified
property values.
*/
hr = pdoUsers->CreateDSObject(
    sbstrName,
    rgAttrInfo,
    sizeof(rgAttrInfo)/sizeof(ADS_ATTR_INFO),
    &pDisp
);

if(SUCCEEDED(hr))
{
    IDirectoryObject *pdoNewUser;

    hr = pDisp->QueryInterface(IID_IDirectoryObject,
        (LPVOID*)&pdoNewUser);
    if(SUCCEEDED(hr))
    {
        ADSVALUE adsValue;
        DWORD dw;

        /*
        Set the initial password.
        */
        IADsUser *padsUser;
        hr = pdoNewUser->QueryInterface(IID_IADsUser,
            (LPVOID*)&padsUser);
        if(SUCCEEDED(hr))
        {
            hr =
                padsUser->SetPassword(CComBSTR(pwszInitialPassword));
            padsUser->Release();
        }

        /*
        Set the pwdLastSet property to zero, which forces the
        user to change their password at next log on.
        */
        adsValue.dwType = ADSTYPE_LARGE_INTEGER;
        adsValue.LargeInteger.LowPart = 0;
        adsValue.LargeInteger.HighPart = 0;
        rgAttrInfo[0].pszAttrName = L"pwdLastSet";
        rgAttrInfo[0].dwControlCode = ADS_ATTR_UPDATE;
        rgAttrInfo[0].dwADsType = ADSTYPE_LARGE_INTEGER;
        rgAttrInfo[0].pADsValues = &adsValue;
        rgAttrInfo[0].dwNumValues = 1;
        hr = pdoNewUser->SetObjectAttributes(rgAttrInfo,
            1,
            &dw);
    }
}

```

```
    pdoNewUser->Release();
}

pDisp->Release();
}

pdoUsers->Release();
}

return hr;
}
```

Enumerating Users

6/3/2022 • 2 minutes to read • [Edit Online](#)

Unlike Windows NT 4.0 domains, Windows 2000 users can be placed in any container or organizational unit (OU) in a domain as well as the root of the domain. This means that users can be in numerous locations in the directory hierarchy. Therefore, you have two choices for enumerating users:

- Enumerate the users directly contained in a container, OU, or at the root of the domain:

Explicitly bind to the container object that contains the users you are interested in enumerating, set a filter containing "user" as the class using the **IADsContainer.Filter** property, and use the **IADsContainer::get_NewEnum** method to enumerate the user objects.

This technique can be used to enumerate users that are directly contained in a container or OU object. If the container contains other containers that can potentially contain other users, you must bind to those containers and recursively enumerate the users on those containers. If it is not required that you manipulate the user objects, and only need to read specific properties, use the deep search described in Option 2.

Because enumeration returns pointers to ADSI COM objects representing each user object, you can call **QueryInterface** to get **IADs**, **IADsUser**, and **IADsPropertyList** interface pointers to the user object. This means you can get interface pointers to each enumerated user object in a container without having to explicitly bind to each user object. To perform operations on all the users directly within a container, enumeration avoids having to bind to each user in order to call **IADs** or **IADsUser** methods. To retrieve specific properties from users, use **IDirectorySearch** as described in option 2.

- Perform a deep search for `(&(objectClass=user)(objectCategory=person))` to find all users in a tree:

First, bind to the container object where to begin the search. For example, to find all users in a domain, bind to root of the domain; to find all users in the forest, bind to the global catalog and search from the root of the GC.

Then use **IDirectorySearch** to query using a search filter containing `(&(objectClass=user)(objectCategory=person))` and search preference of **ADS_SCOPE_SUBTREE**.

You can perform a search with a search preference of **ADS_SCOPE_ONELEVEL** to limit the search to the direct contents of the container object that you bound to.

IDirectorySearch retrieves only the values of specific properties from users. To retrieve values, use **IDirectorySearch**. To manipulate the user objects returned from a search, that is, you want to use **IADs** or **IADsUser** methods, you must explicitly bind to them. To do this, specify **distinguishedName** as one of the properties to return from the search and use the returned distinguished names to bind to each user returned in the search.

Only specific properties are retrieved. You cannot retrieve all attributes without explicitly specifying every possible attribute of the user class.

Querying for Users

6/3/2022 • 2 minutes to read • [Edit Online](#)

To query for a user, the query must contain the search expression "(&(objectClass=user)(objectCategory=person))".

Because the computer class is a subclass of user, a query containing only (objectClass=user) would return user objects and computer objects. Also, the object category of the user object is person (not user); therefore, the expression (objectCategory=user) does not return any users. If you use the expression (objectCategory=person), the query returns user objects and contact objects.

Users can be placed in any container or organizational unit in a domain as well as the root of the domain. This means that users can be in numerous locations in the directory hierarchy. You can perform a deep search for "(objectCategory=user)" to find all users in a container, organizational unit, domain, domain tree, or forest—depending on the object that the **IDirectorySearch** pointer you are using is bound to.

Example Code for Using the Global Catalog to Find Users in a Forest

6/3/2022 • 3 minutes to read • [Edit Online](#)

This topic includes code examples used to search for users in a forest.

The following Visual Basic code example shows how to use ADSI to search for users in a forest.

```

' PrintAllUsersInGlobalCatalog()

' Prints to the debug window the cn and distinguishedName of all users in the
' current forest whose givenName begins with "jeff". This is accomplished
' by binding to the global catalog and then searching for all users
' objects that meet the criteria.

' Be aware that this code example can be modified to print all users in the
' forest, but on a large enterprise, it can take a long time to search for
' and print all users. The filter in this example is only given to
' avoid excessive network traffic.

Public Sub PrintAllUsersInGlobalCatalog()
    Const ADS_SECURE_AUTHENTICATION = 1

    Dim oGC As IADsContainer
    Dim oEnterprise As IADs

    ' Get the enterprise object from the GC namespace.
    Set oGC = GetObject("GC:")
    For Each child In oGC
        Set oEnterprise = child
        Exit For
    Next

    ' Setup ADO.
    Set oConn = CreateObject("ADODB.Connection")
    Set oComm = CreateObject("ADODB.Command")

    oConn.Provider = "ADsDSOObject"
    oConn.Properties("ADSI Flag") = ADS_SECURE_AUTHENTICATION

    oConn.Open
    oComm.ActiveConnection = oConn

    ' Set the search command and filter.
    oComm.CommandText = "<" & oEnterprise.ADsPath & ">;(&(objectCategory=person)(objectClass=user)
(givenName=jeff*));cn,distinguishedName;subTree"

    ' Execute the query.
    Set oRS = oComm.Execute

    ' Print the results.
    oRS.MoveFirst
    While Not oRS.EOF
        For Each field In oRS.Fields
            Debug.Print field
        Next

        Debug.Print ""
        oRS.MoveNext
    Wend
End Sub

```

The following C++ code example uses ADSI to search for users in a forest.

```

*****PrintAllUsersInGlobalCatalog()

Prints to the console the cn and distinguishedName of all users in the
current forest whose givenName begins with "jeff". This is accomplished
by binding to the global catalog and then searching for all users

```

objects that meet the criteria.

Be aware that this code example can be modified to print all users in the forest, but on a large enterprise, it can take a long time to search for and print all users. The filter in this example is only given to avoid excessive network traffic.

```
*****  
  
HRESULT PrintAllUsersInGlobalCatalog()  
{  
    HRESULT hr;  
    CComPtr<IADsContainer> spContainer;  
  
    // Bind to global catalog.  
    hr = ADsOpenObject(L"GC:",  
        NULL,  
        NULL,  
        ADS_SECURE_AUTHENTICATION,  
        IID_IADsContainer,  
        (void**)&spContainer);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    CComPtr<IEnumVARIANT> spEnum;  
    hr = spContainer->get__NewEnum((IUnknown**)&spEnum);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
  
    // Enumerate. There should be only one item.  
    CComPtr<IDirectorySearch> spGCSearch;  
    CComVariant svar;  
    ULONG ulFetched;  
  
    /*  
     * Get the first item in the enumeration. The global catalog container will  
     * only have one object in it.  
     */  
    hr = spEnum->Next(1, &svar, &ulFetched);  
    if(SUCCEEDED(hr) && (ulFetched == 1) && (VT_DISPATCH == svar.vt))  
    {  
        hr = svar.pdispVal->QueryInterface(IID_IDirectorySearch, (LPVOID*)&spGCSearch);  
    }  
  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
    else if(NULL == spGCSearch.p)  
    {  
        return E_FAIL;  
    }  
  
    ADS_SEARCHPREF_INFO SearchPrefs[2];  
  
    // Search entire subtree from root.  
    SearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;  
    SearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;  
    SearchPrefs[0].vValue.Integer = ADS_SCOPE_SUBTREE;  
  
    /*  
     * Use paging in case there are more results than the server can provide in a single page.  
     */  
    SearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;  
    SearchPrefs[1].vValue.dwType = ADSTYPE_INTEGER;
```

```

SearchPrefs[1].vValue.Integer = 1000;

// Set the search preference.
hr = spGCSearch->SetSearchPreference(SearchPrefs, sizeof(SearchPrefs)/sizeof(ADS_SEARCHPREF_INFO));
if(FAILED(hr))
{
    return hr;
}

ADS_SEARCH_HANDLE hSearch;

// Create the search filter.
LPWSTR pwszSearchFilter = L"(&(objectCategory=person)(objectClass=user)(givenName=jeff*))";

// Set attributes to return.
LPWSTR rgpszAttributes[] = {L"cn", L"distinguishedName"};
DWORD dwNumAttributes = sizeof(rgpszAttributes)/sizeof(LPWSTR);

// Execute the search.
hr = spGCSearch->ExecuteSearch( pwszSearchFilter,
                                  rgpszAttributes,
                                  dwNumAttributes,
                                  &hSearch);
if(FAILED(hr))
{
    return hr;
}

// Get the first result row.
hr = spGCSearch->GetFirstRow(hSearch);
while(S_OK == hr)
{
    ADS_SEARCH_COLUMN col;

    // Enumerate the retrieved attributes.
    for(DWORD i = 0; i < dwNumAttributes; i++)
    {
        hr = spGCSearch->GetColumn(hSearch, rgpszAttributes[i], &col);
        if(SUCCEEDED(hr))
        {
            switch(col.dwADsType)
            {
                case ADSTYPE_DN_STRING:
                    wprintf(L"%s: %s\n\n", rgpszAttributes[i], col.pADsValues[0].DNString);
                    break;

                case ADSTYPE_CASE_IGNORE_STRING:
                    wprintf(L"%s: %s\n\n", rgpszAttributes[i], col.pADsValues[0].CaseExactString);
                    break;

                default:
                    break;
            }

            // Free the column.
            spGCSearch->FreeColumn(&col);
        }
    }

    // Get the next row.
    hr = spGCSearch->GetNextRow(hSearch);
}

// Close the search handle to cleanup.
hr = spGCSearch->CloseSearchHandle(hSearch);

return hr;
}

```


Managing Users on Member Servers and Windows 2000 Professional

6/3/2022 • 2 minutes to read • [Edit Online](#)

On member servers and Windows 2000 Professional, there is a local security database. That local security database can contain its own local users whose scope is only the particular computer where they are created. When managing these types of users on member servers and workstations, you use the WinNT provider.

When managing users on a Windows 2000 domain using ADSI, you use the LDAP provider. When managing users on member servers and workstations, you use the WinNT provider.

Enumerating Users on Member Servers and Windows 2000 Professional

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic shows how to enumerate all users, in a local security database, on member servers and computers running on Windows 2000 Professional.

To enumerate the users

1. Bind to the computer using the following rules:
 - a. Use an account that has sufficient rights to access that computer.
 - b. Use the following binding string format using the WinNT provider, computer name, and an extra parameter to instruct ADSI that it is binding to a computer: "WinNT://<computer name>, <computer>". The "<computer name>" parameter is the name of the computer for whose groups to access. This parameter instructs ADSI that it is binding to a computer and enables the WinNT provider parser to skip some ambiguity resolution queries to determine what type of object you are binding to.
 - c. Bind to the **IADsContainer** interface.
2. Add "user" to the **IADsContainer.Filter** property. This causes the enumeration to only contain objects that have the "user" object class.
3. Enumerate the objects. For each user object, use the **IADs** or **IADsUser** interface methods to read the user properties.

Example Code for Enumerating Users on a Member Server

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following Visual Basic code example enumerates all users on a member server or Windows 2000 Professional.

```
Const ADS_SECURE_AUTHENTICATION = 1

' ListObjectsWithWinNtProvider()

' Uses the WinNT provider to list child objects based on a filter.

' Parameters:
'
' pwszComputer - Contains the computer to list the objects for.
' pwszClass - Contains the object class name to filter for.
' pwszUsername - Contains the user name to be used for
'                 authentication.
' pwszPassword - Contains the password to be used for
'                 authentication.

Public Sub ListObjectsWithWinNtProvider(Computer As String, _
                                         Class As String, _
                                         Username As String, _
                                         Password As String)
    Dim BindingString As String
    Dim oComputer As IADsContainer
    Dim oObject As IADs
    Dim oNSP As IADsOpenDSObject

    ' Build the binding string.
    BindingString = "WinNT://" + Computer + ",computer"

    ' Bind to the computer.
    If Username = "" Then
        Set oComputer = GetObject(BindingString)
    Else
        Set oNSP = GetObject("WinNT:")
        Set oComputer = oNSP.OpenDSObject(BindingString, _
                                           Username, _
                                           Password, _
                                           ADS_SECURE_AUTHENTICATION)
    End If

    ' Add the class to the Filter property of the
    ' IADsContainer object.
    oComputer.Filter = Array(Class)

    For Each oObject In oComputer
        Debug.Print ""
        Debug.Print "Name:" + oObject.Name
        Debug.Print "ADsPath:" + oObject.ADPath
        Debug.Print "-----"
    Next
End Sub
```

The following C++ code example enumerates all objects of a specified class, such as a user, and displays the members contained in each object on a member server or Windows 2000 Professional.

```

*****
ListObjectsWithWinNtProvider()

Uses the WinNT provider to list children based on a filter.
Returns S_OK if successful.

Parameters:

pwszComputer - Contains the computer for which to
list the objects.
pwszClass - Contains the object class name to filter for.
pwszUsername - Contains the user name to be used for
authentication.
pwszPassword - Contains the password to be used for
authentication.

*****

```

```

HRESULT ListObjectsWithWinNtProvider(
    LPCWSTR pwszComputer,
    LPCWSTR pwszClass,
    LPCWSTR pwszUsername,
    LPCWSTR pwszPassword)
{
    HRESULT hr;

    IADsContainer * pIADsCont = NULL;

    // Build the binding string.
    CComBSTR sbstrBindingString;
    sbstrBindingString = "WinNT://";
    sbstrBindingString += pwszComputer;
    sbstrBindingString += ",computer";

    // Bind to the container.
    hr = ADsOpenObject( sbstrBindingString,
                        pwszUsername,
                        pwszPassword,
                        ADS_SECURE_AUTHENTICATION,
                        IID_IADsContainer,
                        (void**) &pIADsCont);

    if (SUCCEEDED(hr))
    {
        VARIANT vFilter;
        VariantInit(&vFilter);
        LPWSTR rgpwszFilter[] = {(LPWSTR)pwszClass};

        // Build a Variant of array type, using the filter passed.
        hr = ADsBuildVarArrayStr(rgpwszFilter, 1, &vFilter);
        if (SUCCEEDED(hr))
        {
            // Set the filter for the results of the enumeration.
            hr = pIADsCont->put_Filter(vFilter);
            if (SUCCEEDED(hr))
            {
                IEnumVARIANT *pEnumVariant = NULL;

                // Build an enumerator interface. This is used
                // to enumerate the objects contained in
                // the IADsContainer.
                hr = ADsBuildEnumerator(pIADsCont, &pEnumVariant);

                if(SUCCEEDED(hr))
                {
                    VARIANT Variant;
                    ULONG ulElementsFetched;
```

```

// Loop through and print the data.
while(SUCCEEDED(ADsEnumerateNext(pEnumVariant,
    1,
    &Variant,
    &ulElementsFetched))
    && (ulElementsFetched > 0))
{
    if(VT_DISPATCH == Variant.vt)
    {
        IADS *pIADS= NULL;

        // Query the variant IDispatch *
        // for the IADS interface
        hr = Variant.pdispVal->QueryInterface(IID_IADS,
            (VOID**)&pIADS);

        if (SUCCEEDED(hr))
        {
            // Print the object data.
            CComBSTR sbstrResult;
            hr = pIADS->get_Name(&sbstrResult);
            if(SUCCEEDED(hr))
            {
                wprintf(L"Name : %s\n",
                    sbstrResult);
            }

            hr = pIADS->get_ADsPath(&sbstrResult);
            if(SUCCEEDED(hr))
            {
                wprintf(L"ADSPath : %s\n",
                    sbstrResult);
            }

            wprintf(L"-----\n\n");

            pIADS->Release();
        }
    }

    VariantClear(&Variant);
}

pEnumVariant->Release();
}

}

VariantClear(&vFilter);
}

return hr;
}

```

Creating Users on Member Servers and Windows 2000 Professional

6/3/2022 • 2 minutes to read • [Edit Online](#)

To create a user

1. Bind to the computer using the following rules:
 - a. Use an account that has sufficient rights to access that computer.
 - b. Use the following binding string format using the WinNT provider, computer name, and an extra parameter to tell ADSI that it is binding to a computer: "WinNT://<computer name>,<computer>".
The "<computer name>" computer is the name of the computer whose groups you want to access.
This parameter tells ADSI that it is binding to a computer and allows the WinNT provider's parser to skip some ambiguity resolution queries to determine what type of object you are binding to.
 - c. Bind to the **IADsContainer** interface.
2. Specify "user" as the class using **IADsContainer.Create** to add the user.
3. Write the user to the computer's security database using **IADs.SetInfo**.

Example Code for Creating Users on a Member Server or Windows 2000 Professional

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following Visual Basic code example creates a user on a member server or Windows 2000 Professional.

```
' Example: Creating a user on a member server or workstation

Dim cont As IADsContainer
Dim oUser As IADsUser
Dim v As Variant

On Error GoTo Cleanup

.....
' Parse the arguments
......



sComputer =
    InputBox("This script creates a user on a member " _
        & "server or workstation." _
        & vbCrLf & vbCrLf _
        & "Specify the computer name:")
If sComputer = "" Then
    Exit Sub
End If

sUser = InputBox("Specify the user name:")
If sUser = "" Then
    Exit Sub
End If

.....
' Bind to the computer.
......



Set cont = GetObject("WinNT://" & sComputer & ",computer")

.....
' Create the user.
......



Set oUser = cont.Create("user", sUser)

.....
' Write the user to the computer's security database.
......



oUser.SetInfo

strText = "The user " & sUser & " was successfully added."
strText = strText & vbCrLf _
    & "The user has the following properties:"



.....
' Read the user that was just created
' and display its name and its properties.
......



oUser.GetInfo

strText = strText & "Number of properties: " & Count
For cprop = 1 To Count
    Set v = oUser.Next()
    If IsNull(v) Then
```

```
    Exit For
End If
strText = strText & vbCrLf & cprop & ") " & v.Name _
& " (" & v.ADsType & ")"
Next

MsgBox strText, vbInformation, "Create user on " & sComputer

Cleanup:
If (Err.Number<>0) Then
    MsgBox ("An error has occurred. " & Err.Number)
Set cont = Nothing
Set oUser = Nothing
Set v = Nothing
```

Deleting Users on Member Servers and Windows 2000 Professional

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic shows how to delete a user from a member server or computer running on Windows 2000 Professional.

To delete a user

1. Bind to the computer using the following rules:
 - a. Use an account with sufficient rights to access that computer.
 - b. Use the following binding string format using the WinNT provider, computer name, and an extra parameter to tell ADSI that it is binding to a computer: "WinNT://<computer name>,<computer>".
The "<computer name>" parameter is the name of the computer whose groups you want to access.
This parameter notifies ADSI that it is binding to a computer and allows the WinNT provider parser to skip some ambiguity resolution queries to determine what type of object you are binding to.
 - c. Bind to the [IADsContainer](#) interface.
2. Specify "user" as the class using [IADsContainer.Delete](#) to delete the user.

Be aware that you do not need to call [IADs.SetInfo](#) to commit the change to the container. The [IADsContainer.Delete](#) call commits the deletion of the user directly to the directory.

Example Code for Deleting Users on a Member Server or Windows 2000 Professional

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following Visual Basic code deletes a user on a member server or workstation.

```
Dim cont As IADsContainer
Dim oGroup As IADsGroup

'Example: Deleting a user on a member server or workstation

.....
'Parse the arguments
.....
On Error Resume Next
sComputer = InputBox("This script deletes a user from a member " _
    & "server or workstation." _
    & vbCrLf & vbCrLf _ 
    & "Specify the computer name:")
If sComputer = "" Then
    Exit Sub
End If

sUser = InputBox("Specify the user name:")
If sUser = "" Then
    Exit Sub
End If

.....
'Bind to the computer
.....
Set cont = GetObject("WinNT://" & sComputer & ",computer")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If

.....
'Delete the user
.....
cont.Delete "user", sUser

If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADsContainer::Delete method"
End If

strText = "The user " & sUser & " was deleted on computer " _
    & sComputer & "."

Call show_users(strText, sComputer)
.....
'Display subroutines
.....
Sub show_users(strText, strName)
    MsgBox strText, vbInformation, "Delete user on " & strName
End Sub
```

Values for the countryCode and c Properties

6/3/2022 • 2 minutes to read • [Edit Online](#)

The values for the **countryCode** and **c** properties are obtained from ISO standard 3166. This standard is available at the [International Organization for Standardization](#) website. The **countryCode** property value is the numeric country code from ISO 3166 and the **c** property value is the two letter country/region designation from ISO 3166.

Managing Groups

6/3/2022 • 2 minutes to read • [Edit Online](#)

This section includes topics about how to use groups in Active Directory Domain Services:

- [Group Objects](#)
- [How Security Groups are Used in Access Control](#)
- [Creating Groups in a Domain](#)
- [Adding Members to Groups in a Domain](#)
- [Removing Members from Groups in a Domain](#)
- [Nesting a Group in Another Group](#)
- [Determining a User's or Group's Membership in a Group](#)
- [Enumerating Groups](#)
- [Querying for Groups in a Domain](#)
- [Changing a Group's Scope or Type](#)
- Deleting Groups. A group is deleted in the same was as any other object in Active Directory Domain Services. For more information about deleting Active Directory objects, see [Creating and Deleting Objects in Active Directory Domain Services](#).
- Moving Groups. A group is moved in the same was as any other Active Directory object. For more information, see [Moving Objects](#).
- [Getting the Domain Account-Style Name of a Group](#)
- [Groups on Member Servers and Windows 2000 Professional](#)
- [What Application and Service Developers Need to Know About Groups](#)

For more information about groups in Active Directory Domain Services, see [Understanding groups](#).

Group Objects

6/3/2022 • 5 minutes to read • [Edit Online](#)

A group is represented as a **group** object in Active Directory Domain Services. The following table lists important attributes of the **group** object.

ATTRIBUTE	DESCRIPTION
cn	The cn (or Common-Name) is a single-value attribute that is the object's relative distinguished name. The cn is the name of the group in Active Directory Domain Services. As with all other objects, the cn of a group must be unique among the sibling objects in the container that contains the group.
member	The member attribute is a multi-value attribute that contains the list of distinguished names for the user, group, and contact objects that are members of the group. Each item in the list is a linked reference to the object that represents the member; therefore, the Active Directory server automatically updates the distinguished names in the member property when a member object is moved or renamed.
groupType	<p>The groupType attribute is a single-value attribute that is an integer that specifies the group type and scope using the following bit flags:</p> <ul style="list-style-type: none">• ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP• ADS_GROUP_TYPE_GLOBAL_GROUP• ADS_GROUP_TYPE_UNIVERSAL_GROUP• ADS_GROUP_TYPE_SECURITY_ENABLED <p>The first three flags specify the group scope. The ADS_GROUP_TYPE_SECURITY_ENABLED flag indicates the group type. If this flag is set, the group is a security group. If this flag is not set, the group is a distribution group. For more information, see Group Types.</p>
memberOf	The memberOf attribute is a multiple-value attribute that contains the list of distinguished names for groups that contain the group as a member. This attribute lists the groups beneath which the group is directly nested it does not contain the recursive list of nested predecessors. For example, if group D were nested in group C and group B and group B were nested in group A, the memberOf attribute of group D would list group C and group B, but not group A.
objectGUID	The objectGUID attribute is a single-value attribute that is the unique identifier for the object. This attribute is a Globally Unique Identifier (GUID). When an object is created in the directory, the Active Directory server generates a GUID and assigns it to the object's objectGUID attribute. The GUID is unique across the enterprise and anywhere else. The objectGUID is a 128-bit GUID structure stored as an OctetString.

ATTRIBUTE	DESCRIPTION
objectSid	<p>The objectSid attribute is a single-value attribute that specifies the security identifier (SID) of the group. The SID is a unique value used to identify the group as a security principal. It is a binary value that the system sets when the group is created.</p> <p>Each group has a unique SID that the Windows NT/Windows 2000 Server domain issues that is stored in the objectSid attribute of the group object in the directory. Each time a user logs on, the system retrieves the SID for the groups of which the user is a member and places it in the user's access token. The system uses the SIDs in the user's access token to identify the user and his/her group memberships in all subsequent interactions with Windows NT/Windows 2000 security.</p> <p>When an SID has been used as the unique identifier for a user or group, it cannot ever be used again to identify another user or group.</p>
sAMAccountName	<p>The sAMAccountName attribute is a single-value attribute that is the logon name used to support clients and servers from a previous version (Windows 95, Windows 98, and LAN Manager). The sAMAccountName should be less than 20 characters to support clients and servers from a previous version.</p> <p>The sAMAccountName must be unique among all security principal objects within a domain.</p>

Group Types

There are two types of groups defined by Active Directory Domain Services, *Security Groups* and *Distribution Groups*.

A security group provides a logical grouping of objects and the group itself can be used as a security principal in an Access Control List (ACL). When a security group is given access to an object, all members of the security group automatically receive the same access to the object. Security groups with Universal scope can also be used as an email entity. Sending an email message to a universal security group sends the message to all the members of the group.

A distribution group also provides a logical grouping of objects, but cannot provide any access privileges. Distribution groups are not security enabled and cannot be used as a security principal in an ACL. Distribution groups are only used for grouping purposes. For example, distribution lists can be used with email applications, such as Exchange, to send email to a collection of users.

For more information about group types in Active Directory Domain Services, see the [Group types](#) topic on Microsoft TechNet.

Group Scope

There are three group scopes that are defined by Active Directory Domain Services, *Universal*, *Global* and *Domain Local*. The scope of the group defines what types of object can belong to the group, what types of groups the group can be a member of and the scope of objects that security groups can be given access to. When the domain functional level is set to Windows 2000 mixed mode, security groups with universal scope cannot be created.

The following table lists the three group scopes and more information about each scope for a security group.

SCOPE	POSSIBLE MEMBERS	SCOPE CONVERSION	CAN GRANT PERMISSIONS	POSSIBLE MEMBER OF
Universal	Accounts from any domain in the same forest. Global groups from any domain in the same forest. Other universal groups from any domain in the same forest.	Can be converted to domain local scope. Can be converted to global scope as long as the group does not contain any other universal groups.	On any domain in the same forest or trusting forests.	Other universal groups in the same forest. Domain local groups in the same forest or trusting forests. Local groups on machines in the same forest or trusting forests.
Global	Accounts from the same domain. Other global groups from the same domain.	Can be converted to universal scope as long as the group is not a member of any other global group.	On any domain in the same forest or trusting domains or forests.	Universal groups from any domain in the same forest. Other global groups from the same domain. Domain local groups from any domain in the same forest or from any trusting domain.
Domain Local	Accounts from any domain or any trusted domain. Global groups from any domain or any trusted domain. Universal groups from any domain in the same forest. Other domain local groups from the same domain.	Can be converted to universal scope as long as the group does not contain any other domain local groups.	Within the same domain.	Other domain local groups from the same domain. Local groups on machines in the same domain, excluding built-in groups that have well-known SIDs.

How Security Groups are Used in Access Control

6/3/2022 • 2 minutes to read • [Edit Online](#)

The security identifier (SID) is the object identifier of the user or security group when the user or group is used for security purposes. The name of the user or group is not used as the unique identifier within the system. The SID is stored in the **objectSid** attribute of user objects and security group objects. The Active Directory server generates the **objectSid** when the user or group is created. The system ensures that the SIDs are unique across a forest. Be aware that the **objectGuid** is the unique identifier of a user, group, or any other directory object. The SID changes if a user or group is moved to another domain; the **objectGuid** remains the same.

When a user or group is given permission to access a resource, such as a printer or a file share, the SID of the user or group is added to the access control entry (ACE) defining the granted permission in the resource's discretionary access control list (DACL). In Active Directory Domain Services, each object has an **nTSecurityDescriptor** attribute that stores a DACL defining the access to that particular object or attributes on that object. For more information about setting access control on objects in Active Directory Domain Services, see [Controlling Access to Objects in Active Directory Domain Services](#).

When a user logs on to a Windows 2000 domain, the operating system generates an access token. This access token is used to determine which resources the user may access. The user access token includes the following data:

- User SID.
- SIDs of all global and universal security groups that the user is a member of.
- SIDs of all nested global and universal security groups.

Every process executed on behalf of this user has a copy of this access token.

When the user attempts to access resources on a computer, the service through which the user accesses the resource will impersonate the user by creating a new access token based on the access token created at user logon time. This new access token will also contain the following SIDs:

- SIDs for all domain local groups in the target domain that the user is a member of.
- SIDs for all machine local groups on the target computer that the user is a member of.

The service uses this new access token to evaluate access to the resource. If a SID in the access token appears in any ACEs in the DACL, the service gives the user the permissions specified in those ACEs.

Creating Groups in a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

A group object is created in Active Directory Domain Services in the domain container where the new group will be contained. Groups can be created at the root of the domain, within an organizational unit, or within a container. To create a group object, use the [IADsContainer::Create](#) or the [IDirectoryObject::CreateDSObject](#) method.

The following attributes are required to make the group object a legal group that the Active Directory server and the Windows security system will recognize:

cn

Specifies the name of the group object in the directory. This will be the object's relative distinguished name within the container where the group is created.

groupType

Contains an integer that specifies the group type and scope. The [ADS_GROUP_TYPE_ENUM](#) enumeration defines the possible values for the **groupType** attribute.

The following list defines common group types and values for this attribute.

Domain Local Distribution

ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP

Domain Local Security

ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP | ADS_GROUP_TYPE_SECURITY_ENABLED

Global Distribution

ADS_GROUP_TYPE_GLOBAL_GROUP

Global Security

ADS_GROUP_TYPE_GLOBAL_GROUP | ADS_GROUP_TYPE_SECURITY_ENABLED

Universal Distribution

ADS_GROUP_TYPE_UNIVERSAL_GROUP

Universal Security

ADS_GROUP_TYPE_UNIVERSAL_GROUP | ADS_GROUP_TYPE_SECURITY_ENABLED

If the group is intended for setting access control on directory objects, the group should be a Global Security or Universal Security group.

Be aware that Universal Security groups can only be created on Windows 2000 domains running in native mode. For more information about detecting mixed and native mode, see [Detecting the Operation Mode of a Domain](#).

sAMAccountName

Contains a string that is the name used to support clients and servers from a previous version. The **sAMAccountName** should be less than 20 characters to support clients of a previous version of Windows.

The **sAMAccountName** must be unique among all security principal objects within the domain. A query should be performed against the domain to verify that the **sAMAccountName** is unique within the domain.

The members of the group can be added at creation time using the [IDirectoryObject::CreateDSObject](#) method. Optionally, members can be added to the group after creation using the [IADsGroup::Add](#) method. For more information about adding members to a group, see [Adding Members to Groups in a Domain](#).

Example Code for Creating a Group

6/3/2022 • 5 minutes to read • [Edit Online](#)

The following C++ code example contains a function that creates a group with only the essential properties explicitly set (**cn**, **sAMAccountType**, **groupType**) and containing no members.

```
//////////  
/* CreateSimpleGroup() - Function for creating a basic group  
  
Parameters  
  
    IDirectoryObject *pDirObject - Parent Directory Object  
                                    for the new group  
    LPWSTR pwCommonName        - Common Name for the new group  
    IADs ** ppObjRet          - Pointer to the Pointer which  
                                will receive the new Group  
    int iGroupType             - Bitflags for new group:  
  
                                ADS_GROUP_TYPE_GLOBAL_GROUP,  
                                ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP,  
                                ADS_GROUP_TYPE_UNIVERSAL_GROUP,  
                                ADS_GROUP_TYPE_SECURITY_ENABLED  
*/  
  
HRESULT CreateSimpleGroup(IDirectoryObject *pDirObject,  
                         LPWSTR pwCommonName,  
                         LPWSTR pwSamAcctName,  
                         IADs **ppObjRet,  
                         int iGroupType)  
{  
    if(!pDirObject)  
        return E_INVALIDARG;  
  
    // Verify that the group type is Universal Security  
    // if true, ensure that domain is in native mode.  
    if(((iGroupType & ADS_GROUP_TYPE_UNIVERSAL_GROUP) ==  
        ADS_GROUP_TYPE_UNIVERSAL_GROUP)  
        &&((iGroupType & ADS_GROUP_TYPE_SECURITY_ENABLED) ==  
            ADS_GROUP_TYPE_SECURITY_ENABLED))  
    {  
        // Verify that the domain that contains the container  
        // is in mixed mode.  
        hr = CheckDomainModeForObject(pDirObject, &bIsMixed);  
  
        if (SUCCEEDED(hr))  
        {  
            if (bIsMixed)  
                return E_INVALIDARG;  
        }  
        else  
        {  
            return hr;  
        }  
    }  
  
    // SamAccountName cannot be larger than 20 characters.  
    if (wcslen(pwSamAcctName) >20)  
    {  
        return E_FAIL;  
    }  
  
    HRESULT hr;
```

```

/*
ADSVVALUE is used to specify attribute data for calling
IDirectoryObject::CreateDSObject()
When Creating a new group, the required attributes are :
objectClass,sAMAccountName and groupType.

In this function the "objectClass" is set to "group".
The "sAMAccountName" is the Windows NT 4.0 name.
In Windows 2000/Windows NT 4 mixed-mode environment,
this attribute is exposed to the Windows NT 4 computers.
Therefore, this name must be globally unique throughout
the network and cannot exceed 20 characters in length.

*/
ADSVVALUE    sAMValue;
ADSVVALUE    classValue;
ADSVVALUE    groupType;

LPDISPATCH pDisp;
WCHAR        *pwCommonNameFull;

// Build an Array of ADS_ATTR_INFO structures
// Be aware that the sAMAccountName and groupType entries
// contain a pointer to the respective ADSVALUE
// structures defined previously.
ADS_ATTR_INFO attrInfo[] =
{
    {L"objectClass", ADS_ATTR_UPDATE,
     ADSTYPE_CASE_IGNORE_STRING, &classValue, 1},
    {L"sAMAccountName", ADS_ATTR_UPDATE,
     ADSTYPE_CASE_IGNORE_STRING, &sAMValue, 1},
    {L"groupType", ADS_ATTR_UPDATE,
     ADSTYPE_CASE_IGNORE_STRING, &groupType, 1}
};

// Get the size of the array.
DWORD dwAttrs = sizeof(attrInfo)/sizeof(ADS_ATTR_INFO);

/*
For ADSVALUES, the dwType member and
the data value member (in this case "CaseIgnoreString")
must be set.
*/

// To create a group, this parameter must be set to "group".
classValue.dwType = ADSTYPE_CASE_IGNORE_STRING;
classValue.CaseIgnoreString = L"group";

// Set sAMAccountName to the name passed to this function
sAMValue.dwType=ADSTYPE_CASE_IGNORE_STRING;
sAMValue.CaseIgnoreString = pwSamAcctName;

// Set the groupType to the group type passed to this function
groupType.dwType=ADSTYPE_INTEGER;
groupType.Integer = iGroupType;

// Allocate a buffer to hold the full common name string
pwCommonNameFull = new WCHAR[wcslen(pwCommonName)+4];

// Be aware that CN is limited to 64 characters.
// Take the passed commonname and prefix a 'CN=' to conform
// to the format that IDirectoryObject::CreateDSObject() requires
swprintf_s(pwCommonNameFull,L"CN=%s",pwCommonName);

// Create the new group.
hr = pDirObject->CreateDSObject( pwCommonNameFull, attrInfo,
                                 dwAttrs, &pDisp );
if (SUCCEEDED(hr))
{

```

```

        // Query the new group for an IADs to be returned
        // from this function.
        hr = pDisp->QueryInterface(IID_IADs,(void**) ppObjRet);

        pDisp->Release();
        pDisp = NULL;
    }
    return hr;
}

HRESULT GetDomainMode(IADs *pDomain, BOOL *bIsMixed)
{
    HRESULT hr = E_FAIL;
    VARIANT var;
    if (pDomain)
    {
        VariantClear(&var);
        // Get the ntMixedDomain attribute
        hr = pDomain->Get(CComBSTR("ntMixedDomain"), &var);
        if (SUCCEEDED(hr))
        {
            // Type should be VT_I4.
            if (var.vt==VT_I4)
            {
                // Zero (0) indicates native mode.
                if (var.lVal == 0)
                    *bIsMixed = FALSE;
                // One (1) indicates mixed mode.
                else if (var.lVal == 1)
                    *bIsMixed = TRUE;
                else
                    hr=E_FAIL;
            }
        }
        VariantClear(&var);
    }
    return hr;
}

HRESULT CheckDomainModeOfObject(IDirectoryObject *pDirObject,
                                BOOL *bIsMixed)
{
    HRESULT hr = E_FAIL;
    IADs *pDomain = NULL;
    VARIANT VarTest;
    WCHAR *pFound = NULL;
    int iLen;
    WCHAR *pDomainPath = new WCHAR[MAX_PATH*2];

    // Verify that the domain that contains the container
    // is in mixed mode
    WCHAR *pVal = NULL;
    pVal = GetDirectoryObjectAttrib(pDirObject,L"canonicalName");

    if (pVal)
    {
        // Parse the canonical name for the DNS name of the domain
        pFound = wcschr(pVal,'/');
        // Bind to the domain using the dns name,
        // get defaultnamingcontext,
        if (pFound)
        {
            iLen = pFound - pVal;
            wcscpy_s(pDomainPath, L"LDAP://");
            wcsncat_s(pDomainPath, pVal,iLen);
            wcsncat_s(pDomainPath, L"/rootDSE").

```

```

wcscat_s(pDomainPath, L"/rootDSE");
wprintf(L"DNS Name: %s\n", pDomainPath);
VariantClear(&VarTest);
hr = ADsOpenObject(pDomainPath,
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION, // Use Secure
                                                // Authentication
                    IID_IADs,
                    (void**)&pDomain);
if (SUCCEEDED(hr))
{
    hr = pDomain->Get(CComBSTR("defaultNamingContext"),
                         &VarTest);
    if (SUCCEEDED(hr))
    {
        wcscopy_s(pDomainPath, L"LDAP://");
        wcsncat_s(pDomainPath, pVal,iLen);
        wscat_s(pDomainPath, L"/");
        wscat_s(pDomainPath, VarTest.bstrVal);
        VariantClear(&VarTest);
        if (pDomain)
            pDomain->Release();
        if (SUCCEEDED(hr))
        {
            hr = ADsOpenObject(pDomainPath,
                               NULL,
                               NULL,
                               ADS_SECURE_AUTHENTICATION,
                               IID_IADs,
                               (void**)&pDomain);
            if (SUCCEEDED(hr))
            {
                hr = GetDomainMode(pDomain, bIsMixed);
            }
        }
    }
    if (pDomain)
        pDomain->Release();
}
}
return hr;
}

/*
 */

```

```

GetDirectoryObjectAttrib() - Returns the value of the attribute
                           named in pAttrName from the
                           IDirectoryObject passed.

Parameters

```

```

IDirectoryObject *pDirObject - Object from which to retrieve
                                an attribute value
LPWSTR pAttrName           - Name of attribute to retrieve
*/
WCHAR * GetDirectoryObjectAttrib(IDirectoryObject *pDirObject,
                                 LPWSTR pAttrName)
{
    HRESULT     hr;
    ADS_ATTR_INFO *pAttrInfo=NULL;
    DWORD      dwReturn;
    static WCHAR pwReturn[1024];

    pwReturn[0] = 0;

    hr = pDirObject->GetObjectAttributes( &pAttrName,
                                            1,
                                            0);

```

```

    &pAttrInfo,
    &dwReturn );
}

if ( SUCCEEDED(hr) )
{
    for(DWORD idx=0; idx < dwReturn;idx++, pAttrInfo++ )
    {
        if ( (_wcsicmp(pAttrInfo->pszAttrName,pAttrName) == 0 ) &&
            (pAttrInfo->dwADsType == ADSTYPE_CASE_IGNORE_STRING))
        {
            wcscpy_s(pwReturn,
                    pAttrInfo->pADsValues->CaseIgnoreString);
            break;
        }
    }
    FreeADsMem( pAttrInfo );
}
return pwReturn;
}

```

The following Visual Basic code example creates a group with only the essential properties explicitly set (**cn**, **sAMAccountType**, **groupType**) and containing no members.

```
' Put the required attributes.  
oNewObject.Put "sAMAccountName", sSAMAcctName  
oNewObject.Put "GroupType", iGroupType  
  
' Commit the new group.  
oNewObject.SetInfo  
  
' Print group vitals.  
DisplayMessage ">>> Created new GROUP with a groupType of " & Str(iGroupType)  
PrintIADSOBJECT oNewObject  
  
Set oDirObjectRet = oNewObject  
Set oNewObject = Nothing  
Set oIadsContDirObj = Nothing  
Exit Sub  
  
CleanUp:  
    Set oNewObject = Nothing  
    Set oIadsContDirObj = Nothing  
    Set oDirObjectRet = Nothing  
  
End Sub
```

Adding Members to Groups in a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

A group can contain any number of users, contacts, or other groups as members. The following list lists the attributes of the group object that control group membership.

ATTRIBUTE	DESCRIPTION
member	The member attribute contains the distinguished names for the objects that are members of the group.
memberOf	The memberOf attribute contains the distinguished names of groups that contain the group as a direct member. The memberOf attribute does not contain any inherited group membership data. For example, if GroupA is a member of GroupB and GroupB is a member of GroupC, the memberOf attribute for GroupA will contain GroupB, but not GroupC. The Active Directory server maintains this property. When a distinguished name is added to the member property of another group, that other group's distinguished name is added to this group's memberOf property.

Each of the following methods can be used to add a member to a group. You can add a member by using the distinguished name of the member or binding to the member object and then adding the member object to the group object.

To add a member that belongs to a downlevel domain to a group in an uplevel domain, use the bindable form of the SID string for the distinguished name. For more information and a code example that shows how to convert an **objectSid** into a bindable string, see the **GetLDAPSidBindStringFromVariantSID** example function in [Example Code for Converting an objectSid into a Bindable String](#).

Adding Members to a Group by Using IADsGroup

The **IADsGroup** interface can be used to add members to a group by using the **IADsGroup.Add** method. Bind to and obtain the **IADsGroup** interface for the group object. Then the **IADsGroup.Add** method can be used to add members to the group.

Adding Members to a Group by Using IDirectoryObject

The **IDirectoryObject** interface can be used to add members to a group by using the **IDirectoryObject::SetObjectAttributes** method to modify the **member** attribute for the group. Bind to and obtain the **IDirectoryObject** interface for the group object. Then use the **IDirectoryObject::SetObjectAttributes** method to modify the **member** attribute.

NOTE

Because the **member** attribute has multiple values, ensure that you use the **ADS_ATTR_APPEND** control code to add a distinguished name to the **member** attribute. Using the **ADS_ATTR_UPDATE** control code will cause the existing **member** values to be overwritten.

The **IDirectoryObject** interface can also be used to add members to a group when the group is created by specifying the members in the *pAttributeEntries* parameter of the **IDirectoryObject::CreateDSObject**

method.

Adding Members to a Group by Using System.DirectoryServices

You can use the [System.DirectoryServices](#) namespace to add members to a group by using the **PropertyValueCollection.Add** method on the **member** property of the group object. For more information, see [Setting Properties on Directory Objects](#).

Adding Members to a Group by Using the LDAP API

You can use the [Lightweight Directory Access Protocol](#) API to add members to a group by using one of the **ldap_modify*** functions. For more information, see [Modifying a Directory Entry](#).

Example Code for Adding a Member to a Group

6/3/2022 • 4 minutes to read • [Edit Online](#)

This topic contains code examples that add a member to a group. The Visual Basic Scripting Edition (VBScript) and C++ examples add a member by adding the **IADs** object that represents the member to the **IADsGroup** object that represents the group. The Visual Basic .NET and C# examples modify the member property of the **DirectoryEntry** object that represents the group.

The following C# code examples add an existing member to a group. The function takes the ADsPath of the group container and the distinguished name of the member to be added to the group. The ADsPath is used to create a **DirectoryEntry** object that represents the group. The **PropertyValueCollection.Add** method adds to the group the member whose distinguished name was passed to the function. The function then uses the **DirectoryEntry.CommitChanges** method to write the new member information to the database.

Call the function with the following parameters:

- *bindString*: a valid ADsPath for a group container, such as
"LDAP://fabrikam.com/CN=TestGroup,OU=TestOU,DC=fabrikam,DC=com"
- *newMember*: the distinguished name of the member to be added to the group, such as
"CN=JeffSmith,OU=TestOU,DC=fabrikam,DC=com"

```
private void AddMemberToGroup(
    string bindString,
    string newMember
)
{
    try
    {
        DirectoryEntry ent = new DirectoryEntry( bindString );
        ent.Properties["member"].Add(newMember);
        ent.CommitChanges();
    }
    catch (Exception e)
    {
        Console.WriteLine( "An error occurred." );
        Console.WriteLine( "{0}", e.Message );
        return;
    }
}
```

The following Visual Basic .NET code examples add an existing member to a group. The function takes the ADsPath of the group container and the distinguished name of the member to be added to the group. The

ADsPath is used to create a [DirectoryEntry](#) object that represents the group. The [PropertyValueCollection.Add](#) method adds to the group the member whose distinguished name was passed to the function. The function then uses the [DirectoryEntry.CommitChanges](#) method to write the new member information to the database.

Call the function with the following parameters:

- *bindString*: a valid ADsPath for a group container, such as "LDAP://fabrikam.com/CN=TestGroup,OU=TestOU,DC=fabrikam,DC=com"
- *newMember*: the distinguished name of the member to be added to the group, such as "CN=JeffSmith,OU=TestOU,DC=fabrikam,DC=com"

```
Private Sub AddMemberToGroup(ByVal bindString As String,
                            ByVal newMember As String)

    Try

        Dim ent As New DirectoryEntry(bindString)

        ent.Properties("member").Add(newMember)

        ent.CommitChanges()

    Catch e As Exception

        Console.WriteLine("An error occurred.")

        Console.WriteLine("{0}", e.Message)

    Return

End Try

End Sub
```

The following VBScript example adds an existing member to a group. The script adds the user, Jeff Smith, to the TestGroup group.

```
Option Explicit

On Error Resume Next

Dim scriptResult      ' Script success or failure

Dim groupPath         ' ADsPath to the group container

Dim group             ' Group object

Dim memberPath        ' ADsPath to the member

Dim member             ' Member object

Dim groupMemberList   ' Used to display group members

Dim errorText          ' Error handing text

scriptResult = False

groupPath = "LDAP://fabrikam.com/CN=TestGroup,OU=TestOU,DC=fabrikam,DC=com"

memberPath = "LDAP://CN=JeffSmith,OU=TestOU,DC=fabrikam,DC=com"

WScript.Echo("Retrieving group object")

Set group = GetObject(groupPath)
```

```

If Err.number <> vbEmpty then

    Call ErrorHandler("Could not create group object.")

End If

Call ShowMembers(groupPath)      'Optional function call

WScript.Echo("Retrieving new member object")

Set member = GetObject(memberPath)

If Err.number <> vbEmpty then

    Call ErrorHandler("Could not get new member object.")

End If

WScript.Echo("Adding member to group.")

group.Add(member.ADPath)

If Err.number <> vbEmpty then

    Call ErrorHandler("Could not add member to group.")

End If

Call ShowMembers(groupPath)      ' Optional function call

scriptResult = True

Call FinalResult(scriptResult)

'*****
' This function displays the members of a group. The function
' takes the ADsPath of the group.
'*****

Sub ShowMembers(groupPath)

    Dim groupMember

    Dim groupMemberList

    Dim groupObject

    Set groupObject = GetObject(groupPath)

    Set groupMemberList = groupObject.Members

    Select Case groupMemberList.Count

        Case 1

            WScript.Echo vbCrLf & "The group has one member."

        Case 0

            WScript.Echo vbCrLf & "The group has no members."

        Case Else

            WScript.Echo vbCrLf & "The group has " & groupMemberList.Count & " members."

    End Select

    If groupMemberList.Count > 0 then

```

```

    If groupMemberList.Count > 0 Then

        WScript.Echo vbCrLf & "Here is a member list."

        For Each groupMember In groupMemberList

            WScript.Echo groupMember.Name

        Next

        WScript.Echo vbCrLf

    End If

    Set groupObject = Nothing

    Set groupMemberList = Nothing

End Sub

'*****
' This function shows if the script succeeded or failed. The
' function processed the scriptResult variable.
'*****


Sub FinalResult(scriptResult)

    WScript.Echo vbCrLf

    If scriptResult = False Then

        WScript.Echo "Script failed."

    Else

        WScript.Echo("Script successfully completed.")

    End If

    WScript.Quit

End Sub

'*****
' This function handles errors that occur in the script.
'*****


Sub ErrorHandler( errorText )

    WScript.Echo(vbCrLf & errorText)

    WScript.Echo("Error number: " & Err.number)

    WScript.Echo("Error Description: " & Err.Description)

    Err.Clear

    Call FinalResult(scriptResult)

End Sub

```

The following C++ code example adds an existing member to a group.

```

///////////
/* AddMemberToGroup() - Adds the passed directory object as a member
   of passed group

Parameters

IADsGroup * pGroup    - Group to hold the new IDirectoryObject.
IADs* pIADsNewMember - Object which will become a member
                      of the group. Object can be a user,
                      contact, or group.

*/
HRESULT AddMemberToGroup(IADsGroup * pGroup, IADs* pIADsNewMember)
{
    HRESULT hr = E_INVALIDARG;
    if ((!pGroup)||(!pIADsNewMember))
        return hr;

    // Use the IADs::get_ADsPath() member to get the ADsPath.
    // When the ADsPath string is returned, to add the new
    // member to the group, call the IADsGroup::Add() member,
    // passing in the ADsPath string.
    // Query the new member for its AdsPath.
    // This is a fully qualified LDAP path to the object to add.
    BSTR bsNewMemberPath;
    hr = pIADsNewMember->get_ADsPath(&bsNewMemberPath);
    if (SUCCEEDED(hr))
    {

        // Use the IADsGroup interface to add the new member.
        // Pass the LDAP path to the
        // new member to the IADsGroup::Add() member
        hr = pGroup->Add(bsNewMemberPath);

        // Free the string returned from IADs::get_ADsPath()
        SysFreeString(bsNewMemberPath);
    }
    return hr;
}

```

Related topics

[Adding Members to Groups in a Domain](#)

[DirectoryEntry](#)

[DirectoryEntry.CommitChanges](#)

[IADs](#)

[IADsGroup](#)

[PropertyValueCollection.Add](#)

Example Code for Converting an objectSid into a Bindable String

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic contains a code example that converts an **objectSid** to a bindable string to add a member that belongs to a down-level domain to a group in an up-level domain.

The following C++ code example shows how to convert an **objectSid** into a bindable string.

```
HRESULT VariantArrayToBytes(VARIANT Variant,
                            LPBYTE *ppBytes,
                            DWORD *pdwBytes);

/************************************/

GetLDAPsidBindStringFromVariantSID()

Converts a SID in VARIANT form, such as an objectSid value,
and converts it into a bindale string in the form:

LDAP://<SID=xxxxxx...>

The returned string is allocated with AllocADsMem and must
be freed by the caller with FreADsMem.

************************************/

LPWSTR GetLDAPsidBindStringFromVariantSID(VARIANT vSID)
{
    LPWSTR pwszReturn = NULL;

    if(vSID.vt != VT_EMPTY)
    {
        HRESULT hr;
        LPBYTE pByte;
        DWORD dwBytes = 0;

        hr = VariantArrayToBytes(vSID, &pByte, &dwBytes);
        if(S_OK == hr)
        {
            // Convert the BYTE array into a string of
            // hex characters.
            CComBSTR sbstrTemp = "LDAP://<SID=";

            for(DWORD i = 0; i < dwBytes; i++)
            {
                WCHAR wszByte[3];

                swprintf_s(wszByte, L"%02x", pByte[i]);
                sbstrTemp += wszByte;
            }

            sbstrTemp += ">";
            pwszReturn =
                (LPWSTR)AllocADsMem((sbstrTemp.Length() + 1) *
                                    sizeof(WCHAR));
            if(pwszReturn)
            {
                wcscpy_s(pwszReturn, sbstrTemp.m_str);
            }
        }
    }
}
```

```

        }

        FreeADsMem(pByte);
    }

}

return pwszReturn;
}

//****************************************************************************

VariantArrayToBytes()

This function converts a VARIANT array into an array of BYTES.
This function allocates the buffer using AllocADsMem. The caller
must free this memory with FreeADsMem when it is no longer
required.

*****
```

HRESULT VariantArrayToBytes(VARIANT Variant,
 LPBYTE *ppBytes,
 DWORD *pdwBytes)

{

if(!(Variant.vt & VT_ARRAY) ||
 !Variant.parray ||
 !ppBytes ||
 !pdwBytes)
 {
 return E_INVALIDARG;
 }

*ppBytes = NULL;
 *pdwBytes = 0;

HRESULT hr = E_FAIL;
 SAFEARRAY *pArrayVal = NULL;
 CHAR HUGEPE *pArray = NULL;

// Retrieve the safe array.
 pArrayVal = Variant.parray;
 DWORD dwBytes = pArrayVal->rgsabound[0].cElements;
 *ppBytes = (LPBYTE)AllocADsMem(dwBytes);
 if(NULL == *ppBytes)
 {
 return E_OUTOFMEMORY;
 }

hr = SafeArrayAccessData(pArrayVal, (void HUGEPE * FAR *) &pArray);
 if(SUCCEEDED(hr))
 {
 // Copy the bytes to the safe array.
 CopyMemory(*ppBytes, pArray, dwBytes);
 SafeArrayUnaccessData(pArrayVal);
 *pdwBytes = dwBytes;
 }

return hr;
}

Removing Members from Groups in a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

You can remove users, groups, or contacts from groups. The **member** attribute of the **group** object contains all direct members of the group.

The simplest way to remove a member from a group is to call the **IADsGroup.Remove** method on the **IADsGroup** interface of the group object you want to remove members from.

Example Code for Removing a Member from a Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example contains a function that removes a member from a group.

```
///////////  
/* RemoveMemberFromGroup() - Removes the passed directory object from the membership of passed  
group  
  
Parameters  
  
    IADsGroup * pGroup      - Group to remove the member from  
    IADs* pIADsNewMember   - Object to remove.  
                           Object can be a user, contact, or group.  
*/  
HRESULT RemoveMemberFromGroup(IADsGroup * pGroup, IADs* pIADsNewMember)  
{  
    HRESULT hr = E_INVALIDARG;  
    if ((!pGroup) || (!pIADsNewMember))  
        return hr;  
  
    // Use the IADs::get_ADsPath() member to get the ADsPath  
    // When the ADsPath string is returned, all that is required to  
    // remove the member from the group, is to call the  
    // IADsGroup::Remove() method, passing in the ADsPath string.  
    // Query the member for its AdsPath  
    // This is a fully qualified LDAP path to the object to remove.  
    BSTR bsNewMemberPath;  
    hr = pIADsNewMember->get_ADsPath(&bsNewMemberPath);  
    if (SUCCEEDED(hr))  
    {  
  
        // Use the IADsGroup interface to remove the member.  
        // Pass the LDAP path to the  
        // member to the IADsGroup::Remove() method  
        hr = pGroup->Remove(bsNewMemberPath);  
  
        // Free the string returned from IADs::get_ADsPath()  
        SysFreeString(bsNewMemberPath);  
    }  
    return hr;  
}
```

Nesting a Group in Another Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

Adding a group as a member of another group is called *nesting*. For distribution groups, nesting is supported in both mixed mode and native mode. For security groups, nesting is supported only for domains running in native mode.

To nest a group in another group, use the same techniques described in [Adding Members to Groups in a Domain](#). Be aware that depending on the scope of the group, the group can contain only specific types and scopes of other groups.

The nesting options also depend on whether the domain is in mixed mode or native mode. For more information about groups in different domain modes, see [Nesting in Native Mode](#) and [Nesting in Mixed Mode](#).

For more information about nesting of groups, see [Common Errors](#).

Nesting in Native Mode

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following list describes what can be contained in a group that exists in a native-mode domain. This same list also applies to distribution groups in mixed-mode domains. The scope of the group determines these containment rules.

- A universal group can contain other universal groups, global groups and accounts from any domain within the forest in which this Universal Group resides.
- A global group can contain other global groups and accounts from the same domain that the group belongs to. A global group cannot contain any universal groups, or any global group or account from another domain.
- A domain local group can contain universal groups, global groups and accounts from any domain or forest. A domain local group can also contain other domain local groups from the same domain that the group belongs to. A domain local group cannot contain other domain local groups from any other domain or forest.

Nesting in Mixed Mode

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic lists the restrictions of security groups in a mixed-mode domain.

Security groups in a mixed-mode domain have the following restrictions:

- Universal groups cannot be created in mixed-mode domains because the universal scope is supported only in Windows 2000 native-mode domains.
- A global group can contain accounts from the same domain to which the group belongs. A global group cannot contain any universal groups, any global group, or an account from another domain.
- A domain local group can contain global groups and accounts from any domain or forest. A domain local group cannot contain any other domain local group.

Be aware that distribution groups can be nested according to the nesting rules for native mode.

Common Errors (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following table contains a list of common errors that can occur based on the scope of the group being nested.

HRESULT	DESCRIPTION
0x8007001F	General failure. This error occurs if you attempt to: <ul style="list-style-type: none">• Add a domain local group to a global or universal group or a domain local group in another domain. Domain local groups can only be added as members to other domain local groups in the same domain.• Add a universal group to a global group. Universal groups can be added to universal and domain local groups, but not global groups.
0x8007202F	ERROR_DS_CONSTRAINT_VIOLATION. This error occurs if you attempt to add a security group to another group in a domain running in mixed mode. Security groups cannot be nested in mixed mode; security groups can only be nested in native mode.

Determining a User's or Group's Membership in a Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **IADsGroup.IsMember** method can be used to determine if an object is a member of a group. This method returns TRUE if the specified object is a direct member of the group, that is, the group's member property contains the specified object.

A group can contain other groups. The **IADsGroup.IsMember** method does not recursively verify the members of groups in its member property, groups within those groups, and so on. To recursively verify that an object is a member of a group, enumerate the groups in the member property, verify the members of those groups to see if the object is a member, and if those groups contain other groups, check their members, and so on.

NOTE

Since groups can be nested, group membership may have loops. Any script that enumerates over many groups should keep an internal list of groups to end recursion if a group has already been visited.

Example Code for Checking for Membership in a Group

6/3/2022 • 4 minutes to read • [Edit Online](#)

The following code example verifies absolute membership of an object by recursively verifying that an object is a member of a group or any groups nested in that group.

```

/*
 * RecursiveIsMember()

- Recursively scans the members of passed IADsGroup pointer
and any groups that belong to the passed pointer - for
membership of Passed group.

Returns TRUE if the member is found in the passed group,
or if the passed member is a member of any group which is
a member of the passed group.

Parameters

IADsGroup * pADsGroup           - Group from which to
                                 verify members.
LPWSTR     pwszMember           - LDAP path for object
                                 to verify membership.
BOOL       bVerbose              - IF TRUE, will output verbose
                                 data for the scan.

OPTIONAL Parameters

LPOLESTR   pwszUser             - User Name and Password, if the
                                 parameters are not passed,
LPOLESTER  pwszPassword         - binding will use ADsGetObject;
                                 if the parameters are specified,
                                 ADsOpenObject is used, passing
                                 user name and password.

*/
BOOL RecursiveIsMember(IADsGroup * pADsGroup,
                      LPWSTR pwszMemberGUID,
                      LPWSTR pwszMemberPath,
                      BOOL bVerbose,
                      LPOLESTR pwszUser,
                      LPOLESTR pwszPassword)

{
    HRESULT      hr          = S_OK; // COM Result Code
    IADsMembers * pADsMembers = NULL; // Pointer to Members
                                      // of the IADsGroup
    BOOL         fContinue    = TRUE; // Looping Variable
    IEnumVARIANT * pEnumVariant = NULL; // Pointer to the Enum
                                         // variant
    IUnknown *   pUnknown    = NULL; // IUnknown for getting
                                      // the ENUM initially
    VARIANT      VariantArray[FETCH_NUM]; // Variant array for tem-
                                         // holding returned data
    ULONG        ulElementsFetched = NULL; // Number of elements
                                         // retrieved
    BSTR         bsGroupPath  = NULL;
    BOOL         bRet         = FALSE;

    if(!IADsGroup || !pwszMemberGUID || !pwszMemberPath)
    {
        bRet = FALSE;
    }
    else
    {
        if(bVerbose)
        {
            printf("RecursiveIsMember: Verbose mode enabled.\n");
        }

        if(pwszUser & pwszPassword)
        {
            if(ADsOpenObject(&pADsGroup, pwszUser, pwszPassword))
            {
                if(ADsGetObject(pADsGroup, pwszMemberGUID))
                {
                    if(ADsIsMember(pADsGroup, pwszMemberPath))
                    {
                        bRet = TRUE;
                    }
                }
            }
        }
        else
        {
            if(ADsGetObject(pADsGroup, pwszMemberGUID))
            {
                if(ADsIsMember(pADsGroup, pwszMemberPath))
                {
                    bRet = TRUE;
                }
            }
        }
    }
}

```

```

        return FALSE;
    }

    // Get the path of the object passed in
    hr = pADsGroup->get_ADsPath(&bsGroupPath);

    if (!SUCCEEDED(hr))
        return hr;

    if (bVerbose)
    {
        WCHAR pwszOutput[2048];
        swprintf_s(pwszOutput,
                   L"Checking the Group: %s\n\n for the member: %s\n\n",
                   bsGroupPath,
                   pwszMemberPath);
        PrintBanner(pwszOutput);
    }

    // Get an interface pointer to the IADsCollection of members
    hr = pADsGroup->Members(&pADsMembers);

    if (SUCCEEDED(hr))
    {
        // Query the IADsCollection of members for a new ENUM
        // Interface. Be aware that the enum comes back as an
        // IUnknown *
        hr = pADsMembers->get__NewEnum(&pUnknown);

        if (SUCCEEDED(hr))
        {
            // Call the QueryInterface method for the
            // IUnknown * for an IEnumVARIANT interface.
            hr = pUnknown->QueryInterface(IID_IEnumVARIANT,
                                            (void **)&pEnumVariant);

            if (SUCCEEDED(hr))
            {
                // While no hit errors or end of data...
                while (fContinue)
                {
                    ulElementsFetched = 0;
                    // Get a "batch" number of group members-number
                    // of rows specified by FETCH_NUM
                    hr = ADsEnumerateNext(pEnumVariant,
                                          FETCH_NUM,
                                          VariantArray,
                                          &ulElementsFetched);

                    if (ulElementsFetched )
                    {
                        // Loop through the current batch,
                        // printing the path for each member.
                        for (ULONG i = 0; i < ulElementsFetched; i++ )
                        {
                            // Pointer for holding dispatch of element:
                            IDispatch * pDispatch      = NULL;

                            // Holds path of object:
                            BSTR          bstrCurrentPath  = NULL;

                            // Holds the GUID of object:
                            BSTR          bstrGuidCurrent   = NULL;

                            // Holds the current object:
                            IDirectoryObject * pIDOCurrent = NULL;

                            // Get the dispatch pointer for the variant
                            pDispatch = VariantArray[i].pdispVal;

```

```

assert(HAS_BIT_STYLE(VariantArray[i].vt,
                     VT_DISPATCH));

// Get the IADs interface for the
// "member" of this group
hr = pDispatch->QueryInterface(IID_IDirectoryObject,
                                 (VOID **) &pIDOCurrent);

if (SUCCEEDED(hr))
{
    // Get the GUID for the current object
    hr = GetObjectGuid(pIDOCurrent,bstrGuidCurrent);

    if (FAILED(hr))
        return hr;

    IADs * pIADsCurrent = NULL;

    // Get the IADs Interface for
    // the current object
    hr = pIDOCurrent->QueryInterface(IID_IADs,
                                       (void**) &pIADsCurrent);
    if (FAILED(hr))
        return hr;

    // Get the ADsPath property for this member
    hr = pIADsCurrent->get_ADsPath(&bstrCurrentPath);

    if (SUCCEEDED(hr))
    {
        if (bVerbose)
            wprintf(L"Comparing:\n%s\nWITH:\n%s\n\n",
                   bstrGuidCurrent,
                   pwszMemberGUID);

        // Verify that the member of this group is
        // Equal to passed.
        if (_wcsicmp(bstrGuidCurrent,
                      pwszMemberGUID)==0)
        {
            if (bVerbose)
                wprintf(L"Object: %s is a member of %s\n\n",
                       pwszMemberPath,
                       bstrGuidCurrent);

            bRet = TRUE;
            break;
        }
        else // Otherwise, bind to this and
              // verify that it is a group.
        {
            // If is it a group then the call to
            // the QueryInterface method to
            // IADsGroup succeeds.

            IADsGroup * pIADsGroupAsMember = NULL;

            if (pwszUser)
                hr = ADsOpenObject(bstrCurrentPath,
                                   pwszUser,
                                   pwszPassword,
                                   ADS_SECURE_AUTHENTICATION,
                                   IID_IADsGroup,
                                   (void**) &pIADsGroupAsMember);
            else
                hr = ADsGetObject(bstrCurrentPath,
                                  IID_IADsGroup,
                                  (void **) &pIADsGroupAsMember);

            // If bind was completed, then this is a group.
        }
    }
}

```

```

        if (SUCCEEDED(hr))
        {
            // Recursively call this group to
            // verify this group.
            BOOL bRetRecurse;
            bRetRecurse =
                RecursiveIsMember(pIADsGroupAsMember,
                                  pwszMemberGUID,
                                  pwszMemberPath,
                                  bVerbose,
                                  pwszUser,
                                  pwszPassword );

            if (bRetRecurse)
            {
                bRet = TRUE;
                break;
            }
            pIADsGroupAsMember->Release();
            pIADsGroupAsMember = NULL;
        }
    }

    SysFreeString(bstrCurrentPath);
    bstrCurrentPath = NULL;

    SysFreeString(bstrGuidCurrent);
    bstrGuidCurrent = NULL;
}

// Release.
pIDOCurrent->Release();
pIDOCurrent = NULL;
if (pIADsCurrent)
{
    pIADsCurrent->Release();
    pIADsCurrent = NULL;
}
}

// Clear the variant array.
memset(VariantArray, 0, sizeof(VARIANT)*FETCH_NUM);
}

else
fContinue = FALSE;
}

pEnumVariant->Release();
pEnumVariant = NULL;
}

pUnknown->Release();
pUnknown = NULL;
}

pADsMembers ->Release();
pADsMembers = NULL;
}

// Free the group path if retrieved.
if (bsGroupPath)
{
    SysFreeString(bsGroupPath);
    bsGroupPath = NULL;
}
return bRet;
}

/*
 * GetObjectGuid()      - Gets the GUID, in string form, from
 *                         the passed Directory Object.
 *                         Returns S_OK on success.
 */

```

Parameters


```
_putws(pwszBanner);
_putws(L"////////////////////////////\\n");
}
```

Enumerating Groups (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

This section contains the following information:

- [Enumerating Groups in a Domain](#)
- [Searching for Groups by Scope or Type in a Domain](#)
- [Enumerating Members in a Group](#)
- [Enumerating Groups That Contain Many Members](#)

Enumerating Groups in a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

Groups can be placed in any container or organizational unit (OU) in a domain as well as the root of the domain. This means that groups can be in numerous locations in the directory hierarchy. Therefore, you have two choices for enumerating groups:

1. Enumerate the groups directly contained in a container, OU, or at the root of the domain.

Explicitly bind to the container object that contains the groups to enumerate, set a filter that contains "groups" as the class using the **IADsContainer.Filter** property, and use the **IADsContainer::get_NewEnum** method to enumerate the group objects.

This technique enumerates groups contained directly in a container or OU object. If the container contains other containers that can potentially contain other groups, you must bind to those containers and recursively enumerate the groups on those containers. To manipulate the group objects and read only specific properties, use the deep search described in Option 2.

Because enumeration returns pointers to ADSI COM objects representing each group object, you can call **QueryInterface** to get **IADs**, **IADsGroup**, and **IADsPropertyList** interface pointers to the group object; that is, you can get interface pointers to each enumerated group object in a container without having to explicitly bind to each group object. To perform operations on all the groups directly within a container, enumeration does not require binding to each group in order to call **IADs** or **IADsGroup** methods. To retrieve specific properties from groups, use **IDirectorySearch** as described in the second option.

An exception to this occurs when you attempt to enumerate a group that contains members that are wellKnown security principals, such as Everyone, Authenticated users, BATCH, and so on. Because you cannot bind to these types of objects, they are not listed when you enumerate groups in the WinNT scope, even though it may appear to bind, because certain IADs methods such as Class, ADsPath, and Name return correct results when invoked on enumerated members.

2. Perform a deep search for "objectCategory=group" to find all groups in a tree.

First, bind to the container object where to begin the search. For example, to find all groups in a domain, bind to root of the domain; to find all groups in the forest, bind to global catalog and search from the root of the GC.

Then use **IDirectorySearch** to query using a search filter that contains (objectCategory=group) and search preference of **ADS_SCOPE_SUBTREE**.

NOTE

You can perform a search with a search preference of **ADS_SCOPE_ONELEVEL** to limit the search to the direct contents of the container object that you bound to.

IDirectorySearch retrieves only the values of specific properties from groups. To retrieve values, use **IDirectorySearch**. To manipulate the group objects returned from a search, that is, to use **IADs** or **IADsGroup** methods, explicitly bind to them. To do this, specify **distinguishedName** as one of the properties to return from the search and use the returned distinguished names to bind to each group returned in the search.

Only specific properties are retrieved. You cannot retrieve all attributes without explicitly specifying every possible attribute of the group class.

Searching for Groups by Scope or Type in a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Windows 2000 domains, there is single class called **group** for all group scopes (Domain Local, Global, Universal) and types (security, distribution). The **groupType** attribute of the group object specifies the group type and scope.

To use type or scope to search for groups on Windows 2000 domains, use a filter that contains a matching rule for the **groupType** attribute. For more information about matching rules, see [Search Filter Syntax](#).

For more information and a code example that shows how to search for groups in a domain, see [Example Code for Searching for Groups in a Domain](#).

Example LDAP Query Strings

The following query string examples show how to construct an LDAP query string used to search for or filter specific group types.

The following query string will search for security groups. This example uses "-2147483648" as the decimal equivalent of the **ADS_GROUP_TYPE_SECURITY_ENABLED** flag.

```
(&(objectCategory=group)(groupType:1.2.840.113556.1.4.803:=-2147483648))
```

The following query string will search for universal distribution groups; that is, groups that contain the **ADS_GROUP_TYPE_UNIVERSAL_GROUP** flag and do not contain the **ADS_GROUP_TYPE_SECURITY_ENABLED** flag. This example uses "8" as the decimal equivalent of **ADS_GROUP_TYPE_UNIVERSAL_GROUP** and "-2147483648" as the decimal equivalent of the **ADS_GROUP_TYPE_SECURITY_ENABLED** flag.

```
(&(objectCategory=group)((&(groupType:1.2.840.113556.1.4.803:=8)(!(groupType:1.2.840.113556.1.4.803:=-2147483648))))
```

Enumerating Members in a Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

The members of a group are stored in a multi-value attribute called **member**. For groups with a small to medium-sized membership, use the **IADsGroup.Members** method to get a pointer to an **IADsMembers** object that contains the list of all members. Then use the **IADsMembers::get__NewEnum** to get an enumerator object that you can use to enumerate the members.

If the anticipated group membership will be 1000 or more members, use ranging to retrieve users one range at a time. For more information about using ranging to enumerate members, see [Enumerating Groups That Contain Many Members](#).

Example Code for Displaying Members of a Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes code examples that display members of a group.

The following C++ code example uses the **IADsGroup** and **IADsMembers** classes to display the members of a group.

```
///////////
/* PrintGroupObjectMembers()
- Prints the members of the group that the pADsGroup passes.

Parameters
    IADsGroup * pADsGroup      - Group from which to list members.

*/
HRESULT PrintGroupObjectMembers(IADsGroup * pADsGroup)
{
    HRESULT          hr           = S_OK;          // COM Result Code
    IADsMembers *   pADsMembers  = NULL;          // Pointer to Members of the IADsGroup
    BOOL            fContinue    = TRUE;          // Looping Variable
    IEnumVARIANT * pEnumVariant = NULL;          // Pointer to the ENUM variant
    IUnknown *      pUnknown     = NULL;          // IUnknown for getting the ENUM initially
    VARIANT         VariantArray[FETCH_NUM];        // Variant array for temp holding returned data
    ULONG           ulElementsFetched = NULL;       // Number of elements retrieved

    // Get an interface pointer to the IADsCollection of members.
    hr = pADsGroup->Members(&pADsMembers);

    if (SUCCEEDED(hr))
    {

        // Query the IADsCollection of members for a new ENUM Interface.
        // Be aware that the enum comes back as an IUnknown *
        hr = pADsMembers->get__NewEnum(&pUnknown);

        if (SUCCEEDED(hr))
        {

            // Call the QueryInterface method for the IUnknown * for a IEnumVARIANT interface.
            hr = pUnknown->QueryInterface(IID_IEnumVARIANT, (void **)&pEnumVariant);

            if (SUCCEEDED(hr))
            {

                // While no errors or end of data...
                while (fContinue)
                {
                    ulElementsFetched = 0;

                    // Get a "batch" number of group members - number of rows that FETCH_NUM specifies
                    hr = ADsEnumerateNext(pEnumVariant, FETCH_NUM, VariantArray, &ulElementsFetched);

                    if (ulElementsFetched )//SUCCEEDED(hr) && hr != S_FALSE)
                    {

                        // Loop through the current batch, printing
                        // the path for each member.
                        for (ULONG i = 0; i < ulElementsFetched; i++)
                        {
                            IDispatch * pDispatch      = NULL;
                            // Pointer for holding dispath of element.
                            IADs      * pIADsGroupMember = NULL;
                            // IADs pointer to group member.
                        }
                    }
                }
            }
        }
    }
}
```

```

        BSTR          bstrPath      = NULL;
        // Contains the path of the object.

        // Get the dispatch pointer for the variant.
        pDispatch = VariantArray[i].pdspVal;
        assert(HAS_BIT_STYLE(VariantArray[i].vt,VT_DISPATCH));

        // Get the IADs interface for the "member" of this group.
        hr = pDispatch->QueryInterface(IID_IADs,
                                         (VOID **) &pIADsGroupMember) ;

        if (SUCCEEDED(hr))
        {

            // Get the ADsPath property for this member.
            hr = pIADsGroupMember->get_ADsPath(&bstrPath) ;

            if (SUCCEEDED(hr))
            {

                // Print the ADsPath of the group member.
                wprintf(L"\tMember Object: %ws\n", bstrPath);
                SysFreeString(bstrPath);
            }
            pIADsGroupMember->Release();
            pIADsGroupMember = NULL;
        }
    }

    // Clear the variant array.
    memset(VariantArray, 0, sizeof(VARIANT)*FETCH_NUM);
}
else
{
    fContinue = FALSE;
}
pEnumVariant->Release();
pEnumVariant = NULL;
}
pUnknown->Release();
pUnknown = NULL;
}
pADsMembers ->Release();
pADsMembers = NULL;
}

// If all completed normally, all data
// was printed, and an S_FALSE, indicating
// no more data, was received. If so,
// return S_OK.
if (hr == S_FALSE)
    hr = S_OK;

return hr;
}

```

The following Visual Basic code example uses the [IADs](#) class to display the group members.

```
Sub DisplayGroupMembers(oGroup As IADsGroup)

    Dim oChild As IADs
    Dim sMembers As String

    On Error GoTo CleanUp

    MsgBox "Member Count: " & oGroup.Members.Count

    sMembers = "Names:"

    For Each oChild In oGroup.Members
        sMembers = sMembers & vbCrLf & oChild.Get("name")
    Next oChild

    MsgBox sMembers

    Exit Sub

CleanUp:
    MsgBox ("An error has occurred... " & Err.Number & vbCrLf & Err.Description)
    Set oChild = Nothing

End Sub
```

Enumerating Groups That Contain Many Members

6/3/2022 • 2 minutes to read • [Edit Online](#)

The members of a group are stored in a multi-value attribute called **member**. The **member** attribute can contain a large number of values. Enumerating members can be inefficient when the number of values in a multi-valued attribute becomes large. The server will also limit the maximum number of values that can be retrieved in a single query. This means that if a group may have more members than can be supplied by the server, the only way to enumerate all members is to use incremental retrieval of data, known as *range retrieval*.

Range retrieval involves requesting a limited number of attribute values in a single query. The number of values requested must be less than, or equal to, the maximum number of values supported by the server. To reduce the number of times the query must contact the server, the number of values requested should be as close, as possible, to this maximum. To enable an application to work correctly with all servers, a maximum number of 1000 should be used.

The version of the server that supplies the requested data determines the maximum number of values that can be retrieved in a single query. The following table lists the server version and the maximum number of values that can be retrieved in a single query.

SERVER OPERATING SYSTEM VERSION	MAXIMUM VALUES RETRIEVED
Windows 2000	1000
Windows Server 2003	1500

For more information about retrieving ranges of attribute values with ADSI, see [Attribute Range Retrieval](#).

For more information about retrieving ranges of attribute values with [System.DirectoryServices](#), see [Enumerating Members in a Large Group](#).

Querying for Groups in a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

Groups can be placed in any container or organizational unit in a domain as well as the root of the domain. Groups may not always be in one container. Therefore, it is necessary to search the entire domain to find all groups in the domain.

To search for all groups in a domain, set the search start point to the root of the domain, set the search scope to subtree and search for all objects that have an **objectClass** value of "group".

If groups that contain particular **ADS_GROUP_TYPE_ENUM** values must be found, the **LDAP_MATCHING_RULE_BIT_AND** matching rule operator can be used to search for groups that have particular bits set in the **groupType** attribute. For more information about using matching rules, see [Search Filter Syntax](#).

For more information and a code example that shows how to search for groups in a domain, see [Example Code for Searching for Groups in a Domain](#).

Example Code for Searching for Groups in a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C/C++ code example searches the subtree of the specified container for all group objects that have the specified group type.

```
*****  
PrintGroupsInContainer()  
  
    Searches the entire subtree for all groups objects that contain the  
    specified group type bits from the ADS_GROUP_TYPE_ENUM enumeration.  
*****  
  
HRESULT PrintGroupsInContainer(LPCWSTR pwszContainerDN, DWORD type)  
{  
    HRESULT hr = E_FAIL;  
  
    // Construct the ADsPath to bind to the search root.  
    CComBSTR sbstrADsPath = "LDAP://";  
    sbstrADsPath += pwszContainerDN;  
  
    // Bind to the container.  
    CComPtr<IDirectorySearch> spSearch;  
    hr = AdsGetObject(sbstrADsPath, IID_IDirectorySearch, (LPVOID*)&spSearch);  
    if(SUCCEEDED(hr))  
    {  
        ADS_SEARCHPREF_INFO SearchPref[1];  
  
        // Set the scope of the search to be all of the subtree.  
        SearchPref[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;  
        SearchPref[0].vValue.dwType = ADSTYPE_INTEGER;  
        SearchPref[0].vValue.Integer = ADS_SCOPE_SUBTREE;  
  
        hr = spSearch->SetSearchPreference(SearchPref, sizeof(SearchPref)/sizeof(ADS_SEARCHPREF_INFO));  
        if(FAILED(hr))  
        {  
            return hr;  
        }  
  
        ADS_SEARCH_HANDLE hSearch = NULL;  
        LPWSTR pwszDN = L"distinguishedName";  
        LPWSTR pwszAttributes[1] = {pwszDN};  
  
        // Convert the group type to search for into a string.  
        WCHAR wszGroupType[30]; // Plenty large enough to handle the biggest 32-bit number.  
        swprintf_s(wszGroupType, L"%d", type);  
  
        // Construct the search filter.  
        CComBSTR sbstrSearchFilter;  
        sbstrSearchFilter = "(&(objectClass=group)(groupType:";  
        sbstrSearchFilter += LDAP_MATCHING_RULE_BIT_AND;  
        sbstrSearchFilter += ":=";  
        sbstrSearchFilter += wszGroupType;  
        sbstrSearchFilter += "))";  
  
        // Execute the search.  
        hr = spSearch->ExecuteSearch(sbstrSearchFilter,  
                                     pwszAttributes,
```

```

pwszAttributes,
sizeof(pwszAttributes)/sizeof(LPWSTR),
&hSearch);

if(FAILED(hr))
{
    return hr;
}

// Get the first result row. There should never be more than one result.
hr = spSearch->GetFirstRow(hSearch);
while(S_OK == hr)
{
    ADS_SEARCH_COLUMN col;

    // Get the distinguished name for the current result.
    hr = spSearch->GetColumn(hSearch, pwszDN, &col);
    if(SUCCEEDED(hr))
    {
        if(ADSTYPE_DN_STRING == col.dwADsType)
        {
            wprintf(col.pADsValues[0].DNString);
            wprintf(L"\n\n");
        }

        // Free the column.
        spSearch->FreeColumn(&col);
    }

    hr = spSearch->GetNextRow(hSearch);
}

// Close the search handle to cleanup.
hr = spSearch->CloseSearchHandle(hSearch);
}

return hr;
}

```

The following Visual Basic code example searches the subtree of the specified container for all group objects that have the specified group type.

```

'.....  

' PrintGroupsInContainer()  

'  

' Searches the entire subtree for all groups objects that contain the  

' specified group type bits from the ADS_GROUP_TYPE_ENUM enumeration.  

'  

'.....  

Public Sub PrintGroupsInContainer(ContainerDN As String, GroupType As Long)
    Const ADS_GROUP_TYPE_GLOBAL_GROUP = 2
    Const ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP = 4
    Const ADS_GROUP_TYPE_LOCAL_GROUP = 4
    Const ADS_GROUP_TYPE_UNIVERSAL_GROUP = 8
    Const ADS_GROUP_TYPE_SECURITY_ENABLED = &H80000000

    Const ADS_SECURE_AUTHENTICATION = 1

    Const LDAP_MATCHING_RULE_BIT_AND = "1.2.840.113556.1.4.803"
    Const LDAP_MATCHING_RULE_BIT_OR = "1.2.840.113556.1.4.804"

    Set oConn = CreateObject("ADODB.Connection")
    Set oComm = CreateObject("ADODB.Command")

    oConn.Provider = "ADsDSOObject"
    oConn.Properties("ADSI Flag") = ADS_SECURE_AUTHENTICATION

    oConn.Open
    oComm.ActiveConnection = oConn

    oComm.CommandText = "<LDAP://" + ContainerDN + ">(&(objectClass=group)(groupType:" +  

    LDAP_MATCHING_RULE_BIT_AND + ":=" + str(GroupType) + "));distinguishedName;subtree"

    ' Execute the query.
    Set oRS = oComm.Execute

    ' Print the results.
    oRS.MoveFirst
    While Not oRS.EOF
        List.AddItem oRS.Fields(0)

        oRS.MoveNext
    Wend
End Sub

```

Changing a Group's Scope or Type

6/3/2022 • 2 minutes to read • [Edit Online](#)

Changing a group scope or type is not allowed in mixed-mode domains. However, the following conversions are allowed in native-mode domains:

Global group to universal group: This is only allowed if the global group is not a member of another global group.

Domain local group to universal group: The domain local group being converted cannot contain another domain local group.

Universal group to global or domain local group: For conversion to global group, the universal group being converted cannot contain users or global groups from another domain. For conversion to domain local group, the universal group being converted cannot be a member of any universal group or a domain local group from another domain.

In native mode, a group type can be converted freely between security groups and distribution groups.

Be aware that if a group is used to set access control, changing the scope or type can affect the access control entries (ACEs) that contain that group. The security system will ignore ACEs that contain groups that are not security groups.

Example Code for Changing the Scope of a Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ code example changes the scope of a group.

```
WCHAR *pwszLDAPPath =
L"LDAP://CN=mygroup,OU=myou,DC=Fabrikam,DC=com";

HRESULT hr;
IAcDsGroup * pGroup = NULL;

// Initialize COM.
CoInitialize(0);

// Bind to the passed container.
hr = ADsGetObject( pwszLDAPPath, IID_IAcDsGroup,(void **) &pGroup);

if (SUCCEEDED(hr))
{
    VARIANT vValue;
    BSTR bsValue = SysAllocString(L"groupType");
    VariantInit(&vValue);
    // Set a new GroupType Value.
    vValue.vt = VT_I4;
    vValue.lVal = ADS_GROUP_TYPE_GLOBAL_GROUP ;

    hr = pGroup->Put(bsValue,vValue);
    hr = pGroup->SetInfo();
    pGroup->Release();
    pGroup= NULL;
    SysFreeString(bsValue);
}

CoUninitialize();
```

The following Visual Basic code example changes the scope of a group.

```
Dim x as IAcDs
On Error GoTo CleanUp
Set x = GetObject("LDAP://CN=mygroup,OU=myou,DC=Fabrikam,DC=com")
x.Put "groupType",
      ADS_GROUP_TYPE_UNIVERSAL_GROUP|ADS_GROUP_TYPE_SECURITY_ENABLED
Exit Sub

CleanUp:
    MsgBox("An error has occurred.")
    x = Nothing
```

Getting the Domain Account-Style Name of a Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

Users, groups, computers, and other security principals can be represented in domain account form. Domain account (the logon name used in earlier versions of Windows NT) has the following form:

```
<domain>\<account>
```

Where "<domain>" is the name of the Windows NT domain that contains the user and "<account>" is the **SamAccountName** property of the specified user. For example: "Fabrikam\jeffsmith".

The domain account form can specify the trustee in an ACE in a security descriptor. It is also used for the logon name on computers running Windows version NT 4.0 and earlier.

```
// Need to include the following headers to use DsGetDcName.
// #include <LMCONS.H>
// #include <Dsgetdc.h>
// #include <Lmapibuf.h>
// This function returns the previous version name of the security principal
// specified by the distinguished name specified by szDN.
// The szDomain parameter should be NULL to use the current domain
// to get the name translation. Otherwise, specify the domain to use as the
// domain name (such as liliput)
// or in dotted format (such as lilliput.Fabrikam.com).
HRESULT GetDownlevelName(LPOLESTR szDomainName, LPOLESTR szDN, LPOLESTR *ppNameString)
{
    HRESULT hr = E_FAIL;
    IADSNameTranslate *pNameTr = NULL;
    IADS *pObject = NULL;
    CComBSTR sbstrInitDomain = "";

    if (( !szDN) || (!ppNameString))
    {
        return hr;
    }

    // Use the current domain if none is specified.
    if (!szDomainName)
    {
        // Call DsGetDcName to get the name of this computer's domain.
        PDIRECTORY_CONTROLLER_INFO DomainControllerInfo = NULL;
        DWORD dReturn = 0L;
        dReturn = DsGetDcName( NULL,
                              NULL,
                              NULL,
                              NULL,
                              DS_DIRECTORY_SERVICE_REQUIRED,
                              &DomainControllerInfo
        );
        if (dReturn==NO_ERROR)
        {
            sbstrInitDomain = DomainControllerInfo->DomainName;
            hr = S_OK;
        }
    }

    // Free the buffer.
    if (DomainControllerInfo)
```

```

        NetApiBufferFree(DomainControllerInfo);
    }
else
{
    sbstrInitDomain = szDomainName;
    hr = S_OK;
}

if (SUCCEEDED(hr))
{
    // Create the COM object for the IADsNameTranslate object.
    hr = CoCreateInstance(
                    CLSID_NameTranslate,
                    NULL,
                    CLSCTX_INPROC_SERVER,
                    IID_IADsNameTranslate,
                    (void **) &pNameTr
                );
    if (SUCCEEDED(hr))
    {
        // Initialize for the specified domain.
        hr = pNameTr->Init(ADS_NAME_INITTYPE_DOMAIN, sbstrInitDomain);
        if (SUCCEEDED(hr))
        {
            CComBSTR sbstrNameTr;

            hr = pNameTr->Set(ADS_NAME_TYPE_1779, CComBSTR(szDN));
            hr = pNameTr->Get(ADS_NAME_TYPE_NT4, &sbstrNameTr);
            if (SUCCEEDED(hr))
            {
                *ppNameString = (OLECHAR *)CoTaskMemAlloc (sizeof(OLECHAR)*(sbstrNameTr.Length() + 1));
                if (*ppNameString)
                    wcscpy_s(*ppNameString, sbstrNameTr);
                else
                    hr=E_FAIL;
            }
        }

        pNameTr->Release();
    }
}

// Caller must call CoTaskMemFree to free ppNameString.
return hr;
}

```

Groups on Member Servers and Windows 2000 Professional

6/3/2022 • 2 minutes to read • [Edit Online](#)

On member servers and computers running on Windows 2000 Professional, there is a local security database. This local security database can contain its own local user and local groups whose scope is only the particular computer where they are created. When managing these types of users and groups on member servers and computers running on Windows NT Workstation or Windows 2000 Professional, use the WinNT provider.

When a member server or a computer running on Windows 2000 Professional or Windows 2000 Professional is a member of a Windows 2000 domain, the groups or users in the domain can be used in the local security database to grant rights to that group on that particular computer.

When managing groups on a Windows 2000 domain using ADSI, use the LDAP provider. When managing groups on member servers and computers running on Windows NT Workstation or Windows 2000 Professional, use the WinNT provider.

You must bind, at least one instance, to each provider: 1) Bind to the LDAP provider to retrieve the ADsPath to the group or user you want to add to a group in the local database and 2) Bind to the WinNT provider to add that user or group to a local group.

Enumerating Local Groups

6/3/2022 • 2 minutes to read • [Edit Online](#)

On member servers and computers running on Windows 2000 Professional, you can enumerate all local groups.

Only local groups can be created on member servers and Windows 2000 Professional. However, those local groups can contain:

- Universal and Global groups from the forest that contains the domain to which the computer is a member.
- Domain local groups from that computer's domain.
- Users from any domain in the forest.

To enumerate the local groups on a member server or computer running Windows 2000 Professional

1. Bind to the computer using the following rules:
 - a. Use an account with sufficient rights to access that computer.
 - b. Use the following binding string format using the WinNT provider, computer name, and an extra parameter to instruct ADSI that it is binding to a computer: "WinNT://<computer name>, <computer>".

"The <computer name>" parameter is the name of the computer group to access. This parameter instruct ADSI that it is binding to a computer and allows the WinNT provider's parser to skip some ambiguity-resolution queries to determine what type of object you are binding to.

 - c. Bind to the **IADsContainer** interface.
2. Set a filter that contains "groups" using the **IADsContainer.Filter** property. This enables you to enumerate the container and retrieve only groups.
3. Enumerate the group objects, using the **IADsContainer::get__NewEnum** method.
4. For each the group object, use the **IADsGroup** interface to read the name and members of the group.

Example Code for Enumerating Local Groups

6/3/2022 • 4 minutes to read • [Edit Online](#)

This topic includes a code example that enumerates all objects of a specified class.

The following C++ code example enumerates all objects of a specified class using ADSI.

```
//////////  
/* ListMembersWithWinNtProvider()      - Uses the WinNT provider to list children based on a filter  
                                         Returns S_OK on success  
  
Parameters  
  
LPWSTR pwszComputer      - Computer to list  
LPWSTR pwszClass          - Filter for listing  
LPWSTR pwszUSER = NULL    - User Name for ADsOpenObject() binding- If not passed - Bind Though  
ADsGetObject()  
LPWSTR pwszPASS = NULL    - Password for ADsOpenObject() binding- If not passed - Bind Though  
ADsGetObject()  
  
*/  
HRESULT ListMembersWithWinNtProvider(LPWSTR pwszComputer,LPWSTR pwszClass, LPWSTR pwszUSER = NULL, LPWSTR  
pwszPASS = NULL)  
{  
    HRESULT hr;  
    LPWSTR pwszBindingString = NULL;  
  
    IADsContainer * pIADsCont = NULL;  
  
    // Allocate a String for Binding. This should be large enough.  
    pwszBindingString = new WCHAR[(wcslen(gbsComputer) *2) + 20];  
  
    swprintf_s(pwszBindingString,L"WinNT://%s,computer",pwszComputer);  
  
    // Ensure either no user is passed - or both user and password are passed.  
    assert(!pwszUSER || (pwszUSER && pwszPASS));  
  
    // Bind to the container passed.  
    // If USER and PASS passed in, use ADsOpenObject()  
    if (pwszUSER)  
        hr = ADsOpenObject( pwszBindingString,  
                            pwszUSER,  
                            pwszPASS,  
                            ADS_SECURE_AUTHENTICATION,  
                            IID_IADsContainer,  
                            (void**) &pIADsCont);  
    else  
        hr = ADsGetObject( pwszBindingString, IID_IADsContainer,(void **)&pIADsCont);  
  
    if (SUCCEEDED(hr))  
    {  
        VARIANT vFilter;  
        VariantInit(&vFilter);  
        LPWSTR pwszFilter = pwszClass;  
  
        // Build a Variant of array type, using the filter passed.  
        hr = ADsBuildVarArrayStr(&pwszFilter, 1, &vFilter);  
  
        if (SUCCEEDED(hr))  
        {  
            // Set the filter for the results of the Enum.  
            hr = pIADsCont->put_Filter(vFilter);
```

```

if (SUCCEEDED(hr))
{
    IEnumVARIANT * pEnumVariant = NULL; // Pointer to the IEnumVariant Interface
    VARIANT Variant; // Variant for retrieving data
    ULONG ulElementsFetched; // Number of elements fetched

    // Builds an enumerator interface - this will be used
    // to enumerate the objects contained in the IADsContainer.
    hr = ADsBuildEnumerator(pIADsCont,&pEnumVariant);
    // While no errors, loop through and print the data.
    while (SUCCEEDED(hr) && hr != S_FALSE)
    {

        // Object comes back as a VARIANT holding an IDispatch *
        hr = ADsEnumerateNext(pEnumVariant,1,&Variant,&ulElementsFetched);

        if (hr != S_FALSE)
        {
            assert(HAS_BIT_STYLE(Variant.vt,VT_DISPATCH));

            IDispatch *pDispatch = NULL;
            IADs *pIADs= NULL;
            pDispatch = Variant.pdispVal;

            // Call the QueryInterface method for the Variant IDispatch * for the IADs
            interface.
            hr = pDispatch->QueryInterface(IID_IADs,(VOID **) &pIADs) ;

            if (SUCCEEDED(hr))
            {
                // Print data about the object.
                BSTR bsResult;

                pIADs->get_Name(&bsResult);
                wprintf(L" NAME: %s\n",(LPOLESTR) bsResult);
                SysFreeString(bsResult);

                pIADs->get_ADsPath(&bsResult);
                wprintf(L" ADSPATH: %s\n",(LPOLESTR) bsResult);
                SysFreeString(bsResult);

                puts("-----");
                pIADs->Release();
                pIADs = NULL;
            }
        }
    }

    // Because the hr from iteration was lost, free
    // the interface if the pointer is != NULL
    if (pEnumVariant)
    {
        pEnumVariant->Release();
        pEnumVariant = NULL;
    }
    VariantClear(&Variant);
}

VariantClear(&vFilter);
}

delete [] pwszBindingString;
pwszBindingString = NULL;

return hr;
}

```

The following Visual Basic code example enumerates local groups using the [IADsContainer](#) and [IADsGroup](#) interfaces.

```
' Example: Enumerating all local groups on member server or Windows NT Workstation or Windows 2000 Professional
Dim IADsCont As IADsContainer
Dim Group As IADsGroup

On Error GoTo CleanUp

sComputer = InputBox("This script lists the groups on a member server or workstation." & vbCrLf & vbCrLf & "Specify the computer name:")

If sComputer = "" Then
    MsgBox "No computer name was specified. Specify a computer name."
    Exit Sub
End If

.....
' Bind to the computer
.....
' Be aware that this sample uses the caller's security context.
' To specify a user account other than the user account under which
' which your application is running, use IADsOpenDSObject.
Set IADsCont = GetObject("WinNT://" & sComputer & ",computer")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If

.....
' Filter to view only group objects
.....
IADsCont.Filter = Array("group")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADsContainer::Filter method"
End If

strText = ""
intIndex = 0
intNumDisplay = 0
cmember = 0
'Maximum number of groups to list on a msgbox.
MAX_DISPLAY = 10

.....
' Get each group and display its name and its members
.....
For Each Group In IADsCont
    intIndex = intIndex + 1
    ' Get the name.
    strText = strText & vbCrLf & Right(" " & intIndex, 4) & " " & Group.Name

    intNumDisplay = intNumDisplay + 1
    ' Get the members object.
    Set memberList = Group.Members
    If (Err.Number <> 0) Then
        BailOnFailure Err.Number, "on IADsGroup::members method"
    End If

    ' Get the enumerate the members of the group from the members object
    For Each member In memberList
        If cmember = 0 Then
            strText = strText & vbCrLf & " " & "Members:"
        End If
        strText = strText & vbCrLf & " " & member.Name & " (" & member.Class & ")"
        cmember = cmember + 1
    Next
End If
```

```

.....
If cmember = 0 Then
    strText = strText & vbCrLf & "      " & "No members"
End If
' Display in msgbox if there are MAX_DISPLAY groups to display
If intNumDisplay >= MAX_DISPLAY Then
    Call show_groups(strText, sComputer)
    strText = ""
    intNumDisplay = 0
End If
' Reset the count of members within the current group.
cmember = 0
Next
Call show_groups(strText, sComputer)

Exit Sub

CleanUp:
MsgBox ("An error has occurred... " & Err.Number & vbCrLf & Err.Description)
Set IADsCont = Nothing
Set Group = Nothing

.....
' Display subroutines
.....
Sub show_groups(strText, strName)
    MsgBox strText, vbInformation, "Groups on " & strName
End Sub

Sub BailOnFailure(ErrNum, ErrText)    strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub

```

Creating Local Groups

6/3/2022 • 2 minutes to read • [Edit Online](#)

Only local groups can be created for member servers and Windows 2000 Professional.

To create a local group for a member server or computer running Windows 2000 Professional

1. Bind to the computer using the following rules:

- a. Use an account with sufficient rights to access that computer.
- b. Use the following binding string format using the WinNT provider, computer name, and an extra parameter to instruct ADSI that it is binding to a computer: "WinNT://<computer name>, <computer>".

The "<computer name>" parameter is the name of the computer groups to access.

In the binding string, the "<computer>" parameter instructs ADSI that it is binding to a computer. ADSI makes this data available to the WinNT provider's parser so that it can skip some ambiguity-resolution queries to determine what type of object you are binding to.

- c. Bind to the **IADsContainer** interface.

2. Specify "localGroup" as the class using **IADsContainer.Create** to add the group.

NOTE

If you specify "group" as the class, ADSI uses "localGroup". Do not specify the class as "globalGroup". Groups of class "globalGroup" cannot be created on member servers or a computer running Windows NT Workstation/Windows 2000 Professional. If you specify "globalGroup", **IADsContainer.Create** creates the group in the property cache, but **IADs.SetInfo** does not write the group to the security database and it does not return an error.

3. Write the group to the computer security database using **IADs.SetInfo**.

Example Code for Creating a Local Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ code example creates a local group on a member server or a computer running on Windows NT Workstation or Windows 2000 Professional.

```
*****  
CreateMachineLocalGroup()  
  
Creates a local group on the specified computer and adds the  
specified objects to the group.  
  
Parameters:  
  
pwszComputer - A null-terminated string that contains the name  
of the computer on which to create the local group.  
  
pwszGroupName - A null-terminated string that contains the name  
of the group to create.  
  
rgpwszObjectsToAdd - Contains an array of null-terminated strings  
that specify the objects to add to the new group. The objects in  
this array must belong to the computer specified in pwszComputer.  
dwObjectsToAdd contains the number of elements in this array.  
This parameter is ignored if dwObjectsToAdd contains zero.  
  
dwObjectsToAdd - Contains the number of elements in the  
rgpwszObjectsToAdd array. Pass zero for this parameter to not  
add any objects to the group.  
*****/  
  
HRESULT CreateMachineLocalGroup(  
    LPCWSTR pwszComputer,  
    LPCWSTR pwszGroupName,  
    LPCWSTR *rgpwszObjectsToAdd,  
    DWORD dwObjectsToAdd)  
{  
    if(!pwszComputer || !pwszGroupName || !rgpwszObjectsToAdd)  
    {  
        return E_POINTER;  
    }  
  
    HRESULT hr;  
  
    // Because the WinNT provider is used, bind to the  
    // specific computer name.  
    CComBSTR sbstrADsPath = L"WinNT://";  
    sbstrADsPath += pwszComputer;  
    sbstrADsPath += ",computer";  
  
    // Bind to the container.  
    CComPtr<IADsContainer> spContainer;  
    hr = ADsGetObject(sbstrADsPath,  
                      IID_IADsContainer,  
                      (void**) &spContainer);  
    if(FAILED(hr))  
    {  
        return hr;  
    }  
}
```

```

/*
Create the group. This only creates the group in memory.
The group is not actually created until IADs.SetInfo is called.
*/
CComPtr<IDispatch> spDisp;
hr = spContainer->Create(CComBSTR("group"),
                           CComBSTR(pwszGroupName),
                           &spDisp);

if(FAILED(hr))
{
    return hr;
}

// Get the IADsGroup interface.
CComPtr<IADsGroup> spGroup;
hr = spDisp->QueryInterface(IID_IADsGroup, (void**)&spGroup);
if(FAILED(hr))
{
    return hr;
}

// Commit the group to the directory.
hr = spGroup->SetInfo();
if(HRESULT_FROM_WIN32(ERROR_ALIAS_EXISTS) == hr)
{
    /*
    This occurs if the group exists. Should the function add
    the members to the exiting group or fail?
    */
}
else if(FAILED(hr))
{
    return hr;
}

// Add the specified objects to the group.
for(DWORD i = 0; i < dwObjectsToAdd; i++)
{
    CComBSTR sbstrObj = "WinNT://";
    sbstrObj += pwszComputer;
    sbstrObj += "/";
    sbstrObj += rgpszObjectsToAdd[i];

    hr = spGroup->Add(sbstrObj);
    if(HRESULT_FROM_WIN32(ERROR_MEMBER_IN_ALIAS) == hr)
    {
        /*
        This will occur if the member already
        exists in the group.
        */
    }
    else if(FAILED(hr))
    {
        /*
        The object cannot be added, but this is
        not a catastrophic error. Retry to add
        additional objects.
        */
        hr = S_OK;
        continue;
    }
}

return hr;
}

```

The following Visual Basic code example creates a local group on a member server or a computer running on

Windows NT Workstation or Windows 2000 Professional.

```
Public Sub CreateMachineLocalGroup(Computer As String, _
    GroupName As String, _
    ObjectToAdd As String)

    Dim oComputer As IADsContainer

    ' Bind to the computer.
    Set oComputer = GetObject("WinNT://" & Computer & ",computer")

    ' Create the group. This only creates the group in memory.
    ' The group is not actually created until IADs.SetInfo is called.
    Dim oGroup As IADsGroup
    Set oGroup = oComputer.Create("group", GroupName)

    ' Ignore errors for the next operation.
    On Error Resume Next
    ' Commit the group to the directory.
    oGroup.SetInfo

    ' Stop ignoring errors.
    On Error GoTo 0

    ' Add the objects to the group.
    oGroup.Add "WinNT://" & Computer & "/" & ObjectToAdd
End Sub
```

Deleting Local Groups

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic shows how to delete a local group from a member server or computer running on Windows 2000 Professional.

To delete a local group

1. Bind to the computer using the following rules:
 - a. Use an account with sufficient rights to access that computer.
 - b. Use the following binding string format using the WinNT provider, computer name, and an extra parameter to instruct ADSI that it is binding to a computer: "WinNT://<computer name>, <computer>". The "<computer name>" parameter is the name of the computer group to access. This parameter instruct ADSI that it is binding to a computer and allows the WinNT provider's parser to skip some ambiguity-resolution queries to determine what type of object you are binding to.
 - c. Bind to the **IADsContainer** interface.
2. Specify "group" as the class using **IADsContainer.Delete** to delete the group.

You do not need to call **IADs.SetInfo** to commit the change to the container. The **IADsContainer.Delete** call commits the deletion of the group directly to the directory.

Example Code for Deleting a Local Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ code example deletes a local group on a member server or a computer running Windows 2000 Professional or Windows NT Workstation.

```
*****  
DeleteADObject()  
*****  
  
HRESULT DeleteADObject(LPOLESTR pwszAdsPath,  
                      LPWSTR pwszUsername,  
                      LPWSTR pwszPassword)  
{  
    HRESULT hr;  
    IADs *pIADsToDelete = NULL;  
  
    // Bind to the object to be deleted.  
  
    // If a username and password are passed, use ADsOpenObject(),  
    // otherwise use ADsGetObject()  
    if (!pwszUsername || 0 == *pwszUsername) // No user password --  
                                              // use ADsOpenObject.  
    {  
        hr = ADsGetObject( pwszAdsPath,  
                           IID_IADs,  
                           (void **) & pIADsToDelete);  
    }  
    else  
    {  
        hr = ADsOpenObject( pwszAdsPath,  
                           pwszUsername,  
                           pwszPassword,  
                           ADS_SECURE_AUTHENTICATION,  
                           IID_IADs,  
                           (void**) & pIADsToDelete);  
    }  
  
    if (SUCCEEDED(hr))  
    {  
        BSTR bsParentPath;  
  
        // Get the parent path.  
        hr = pIADsToDelete->get_Parent(&bsParentPath);  
        if(SUCCEEDED(hr))  
        {  
            VARIANT vCNToDelete;  
  
            VariantInit(&vCNToDelete);  
  
            // Get the CN property for the object to delete.  
            hr = pIADsToDelete->Get(CComBSTR("cn"), &vCNToDelete);  
            if (SUCCEEDED(hr))  
            {  
                IDirectoryObject *pIDirObjectParent = NULL;  
  
                // *****  
                // Bind to the parent.  
                // If a username and password are passed,  
                // use ADsOpenObject()
```

```

        // otherwise use ADsGetObject()
        if (!pwszUsername || 0 == *pwszUsername)
            // No user password passed - use ADsOpenObject.
        {
            hr = ADsGetObject(bsParentPath,
                               IID_IDirectoryObject,
                               (void **) &pIDirObjectParent);
        }
        else
        {
            hr = ADsOpenObject(bsParentPath,
                               pwszUsername,
                               pwszPassword,
                               ADS_SECURE_AUTHENTICATION,
                               IID_IDirectoryObject,
                               (void **) &pIDirObjectParent);
        }
    if (SUCCEEDED(hr))
    {
        // Release the object to delete.
        pIADsToDelete->Release();
        pIADsToDelete = NULL;

        LPWSTR pwszPrefix = L"CN=";
        LPWSTR pwszRDN =
            new WCHAR[1strlenW(pwszPrefix) +
                      1strlenW(vCNTToDelete.bstrVal) + 1];
        if (pwszRDN)
        {
            wcscpy_s(pwszRDN, pwszPrefix);
            wcscat_s(pwszRDN, vCNTToDelete.bstrVal);

            // Instruct the parent to
            // delete the child object.
            hr = pIDirObjectParent->DeleteDSObject(pwszRDN);

            delete pwszRDN;
        }
        else
        {
            hr = E_OUTOFMEMORY;
        }

        // Release the Parent Object.
        pIDirObjectParent->Release();
        pIDirObjectParent = NULL;
    }

    VariantClear(&vCNTToDelete);
}

SysFreeString(bsParentPath);
}
}

// If a IADsObject is held, release it.
if ( pIADsToDelete)
{
    // Release the object to delete.
    pIADsToDelete->Release();
    pIADsToDelete = NULL;
}

return hr;
}

```

The following Visual Basic Scripting Edition code example deletes a local group on a member server or a

computer running Windows 2000 Professional or Windows NT Workstation.

```
' Example: Deleting a local group on a member server or Windows NT
'   Workstation or Windows 2000 Professional

.....
' Parse the arguments.
.....
On Error Resume Next

Set oArgs = WScript.Arguments
If oArgs.Count < 2 Then
    sComputer = InputBox("This script deletes a group from a member server or workstation." & vbCrLf &
    vbCrLf &"Specify

the computer name:")
    sGroup = InputBox("Specify the group name:")
Else
    sComputer = oArgs.item(0)
    sGroup = oArgs.item(1)
End If

If sComputer = "" Then
    WScript.Echo "No computer name was specified. You must specify a computer name."
    WScript.Quit(1)
End If
If sGroup = "" Then
    WScript.Echo "No group name was specified. You must specify a group name."
    WScript.Quit(1)
End If

.....
' Bind to the computer.
.....
Set cont= GetObject("WinNT://" & sComputer & ",computer")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
End If

.....
' Delete the group.
.....
' It is not necessary to specify localGroup.
' To specify the group is acceptable.
Set oGroup = cont.Delete("group", sGroup)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on IADsContainer::Delete method"
End If

strText = "The group " & sGroup & " was deleted on computer " & sComputer & "."

Call show_groups(strText, sComputer)

.....
' Display subroutines.
.....
Sub show_groups(strText, strName)
    MsgBox strText, vbInformation, "Create group on " & strName
End Sub
```

Adding Domain Objects to Local Groups

6/3/2022 • 2 minutes to read • [Edit Online](#)

When a member server or a computer running on Windows 2000 Professional is a member of a Windows 2000 domain, the users and groups that belong to the domain can be added to groups on the local computer to grant rights to the domain user or group on that particular computer.

When managing domain local groups on a Windows 2000 domain using ADSI, the LDAP provider is normally used. When managing local groups on member servers and a computer running Windows 2000 Professional, however, the WinNT provider must be used.

Only local groups can be created on member servers and Windows 2000 Professional. However, the local groups can contain:

- Universal and global groups from the forest that contains the domain that the computer is a member.
- Domain local groups from that computer domain.
- Users from any domain in the forest.

To add a domain object to a local group

1. Bind to the **IADsContainer** interface of the computer that contains the local group to add a member to. The binding must be performed using an account that has sufficient rights to access that computer. The binding string must take the form "WinNT://<computer name>,computer" where "<computer name>" is the name of the computer group to add a member to. The ",computer" parameter instructs ADSI that it is binding to a computer. ADSI exposes this data to the WinNT provider's parser so that it can skip some ambiguity-resolution queries to determine what type of object you are binding to. This can save the user a 5 to 20 second wait for the ambiguity to be resolved.
2. Use the **IADsContainer.GetObject** method with "group" as the class and the local group name as the name of the object to bind to the group.
3. Bind to the **IADsGroup** interface of the local group to which a member will be added.
4. Construct the ADsPath of the object to add to the local group in the form "WinNT://<domain>/<name>", where "<domain>" is the name of the domain that contains the object to add and "<name>" is the name of the object to add.
5. Add the user or group to the local group with the **IADsGroup.Add** method, passing the ADsPath constructed in Step 4.

For more information and a code example that shows how to add a domain user or group object to a local group, see [Example Code for Adding a Domain User or Group to a Local Group](#).

Example Code for Adding a Domain User or Group to a Local Group

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following C++ code example adds a domain user or group to a local group on a member server or a computer running on Windows NT Workstation or Windows 2000 Professional.

```
#include <stdio.h>
#include <atlbase.h>
#include <activeds.h>
#include <ntldap.h>

*****  
  
AddDomainUserToLocalGroup()  
  
    Adds a user from a domain to a local group using the WinNT provider.  
  
    Parameters:  
  
    pwszComputerName - Contains the name of the computer that the group belongs to.  
  
    pwszGroupName - Contains the name of the group to add a member to. This group must exist on the target computer.  
  
    pwszDomainName - Contains the name of domain that contains the user to add to the group.  
  
    pwszUserToAdd - Contains the name of the user in the domain to add to the group.  
  
*****/  
  
HRESULT AddDomainUserToLocalGroup(LPCWSTR pwszComputerName,
                                    LPCWSTR pwszGroupName,
                                    LPCWSTR pwszDomainName,
                                    LPCWSTR pwszUserToAdd)
{
    HRESULT hr = E_FAIL;
    CComPtr<IADsContainer> spComputer;

    // Build the binding string.
    CComBSTR sbstrBindingString;
    sbstrBindingString = "WinNT://";
    sbstrBindingString += pwszComputerName;
    sbstrBindingString += ",computer";

    // Bind to the computer that contains the group.
    hr = ADsOpenObject( sbstrBindingString,
                        NULL,
                        NULL,
                        ADS_SECURE_AUTHENTICATION,
                        IID_IADsContainer,
                        (void**)&spComputer);

    if(FAILED(hr))
    {
        return hr;
    }
}
```

```
}

// Bind to the group object.
CComPtr spDisp;
hr = spComputer->GetObject(CComBSTR("group"),
                            CComBSTR(pwszGroupName),
                            &spDisp);
if(FAILED(hr))
{
    return hr;
}

CComPtr<IADsGroup> spGroup;
hr = spDisp->QueryInterface(IID_IADs, (LPVOID*)&spGroup);
if(FAILED(hr))
{
    return hr;
}

// Bind to the member to add.
sbstrBindingString = "WinNT://";
sbstrBindingString += pwszDomainName;
sbstrBindingString += "/";
sbstrBindingString += pwszUserToAdd;

hr = spGroup->Add(sbstrBindingString);

return hr;
}
```

The following Visual Basic code example adds a domain user or group to a local group on a member server or a computer running on Windows NT Workstation or Windows 2000 Professional.

```
.....  
  
' AddDomainUserToLocalGroup  
  
' Adds a user from a domain to a local group using the WinNT  
' provider.  
  
' Parameters:  
  
' ComputerName - Contains the name of the computer that the group  
' belongs to.  
  
' GroupName - Contains the name of the group to add a member to.  
' This group must exist on the target computer.  
  
' DomainName - Contains the name of domain that contains the user  
' to add to the group.  
  
' UserName - Contains the name of the user in the domain to add  
' to the group.  
  
.....  
  
Public Sub AddDomainUserToLocalGroup(ComputerName As String, _  
                                     GroupName As String, _  
                                     DomainName As String, _  
                                     UserName As String)  
    Dim GroupComputer As IADsContainer  
    Dim Group As IADsGroup  
  
    ' Bind to the computer that contains the group.  
    Set GroupComputer = GetObject("WinNT://" &  
                                 ComputerName &  
                                 ",computer")  
  
    ' Bind to the group object.  
    Set Group = GroupComputer.GetObject("group", GroupName)  
  
    ' Add the user to the group.  
    Group.Add ("WinNT://" & DomainName & "/" & UserName)  
End Sub
```

What Application and Service Developers Need to Know About Groups

6/3/2022 • 3 minutes to read • [Edit Online](#)

When developing an application or service, you can use groups to delegate administration or control access to the application or service as a whole or part. For example, an application can control who can perform various administrative operations. To do this, create ACEs that grant specified access rights to the appropriate groups. It is good programming practice to have an ACE that grants access to a group than to have several ACEs that grant access to individual users or computers.

It is recommended that applications use the following guidelines when using groups:

- Do not create dependencies on hard-coded groups, which can create complications if the group is deleted or moved.
- Avoid using built-in groups unless the function of the group provides the specific needs for the application. For example, do not require that a user be a member of the Administrators group just to provide the user with permission to create a computer account. Doing this provides the user with permissions and privileges that they may not need, which can cause security vulnerability. Instead, you should create a new security group that has the specific permissions required and add the user to this new group.
- When creating new groups, keep in mind the following recommendations:
 - Protect the group by setting ACEs that control who can add or remove members.
 - Use global groups if they are used for access control on objects in Active Directory Domain Services.
 - Use universal groups only if necessary (member data is required globally using global catalog; group can contain any user/group). If you do use universal groups, place global groups in the universal group and add/remove users from the global group. Avoid excess change to universal groups for replication efficiency.
- Do not use group membership for access control. Instead, use an ACE and control access by having the system perform an access check.

To control access to operations that do not fit within the predefined access rights for objects in Active Directory Domain services, use the control access rights feature of access control in Windows 2000. For more information, see [ADS_RIGHTS_ENUM](#).

To use a control access right to control the right to perform an operation

1. Create a control access right that defines the type of access to the application or service. For more information, see [Control Access Rights](#).
2. Create an object in Active Directory Domain Services that represents the application, service, or resource.
3. Add object ACEs to the DACL in the object security descriptor to grant or deny users or groups the control access right on that object. For more information, see [Setting Access Rights on an Object](#).
4. When a user attempts to perform the protected operation, your application or service uses the [AccessCheckByTypeResultList](#) function to determine whether the control access right is granted to the user. For more information, see [Checking a Control Access Right in an Object's ACL](#).
5. Based on the result of the access check on the object, your application or service can grant or deny the user access to the application or service.

A service application could also create a group whose members would be the various service instances. For example, a service with instances installed on multiple computers throughout an enterprise might have a common log file that all service instances write to. The service installation program creates the log file and uses

a DACL to grant access only to members of a group. The group members would be the user accounts under which the various service instances are running, or if the services run under the LocalSystem account, the members would be the computer accounts of the host servers.

Tracking Changes

6/3/2022 • 2 minutes to read • [Edit Online](#)

Some applications must maintain consistency between specific data stored in the Active Directory directory service and other data. The other data might be stored in the directory, in a SQL Server table, in a file, or in the registry. When data stored in the directory changes, the other data may be required to change in order to remain consistent. Applications that have this requirement include:

This section does not cover mechanisms used by monitoring applications. These are applications that monitor directory changes not for the purpose of maintaining consistent data between separate stores, but as a system management tool. Although monitoring applications can use the same mechanisms that support change-tracking applications, the following mechanisms are specifically designed for monitoring applications:

- Security auditing. By modifying the system access-control list (SACL) portion of an object security descriptor, you can cause accesses to the object on a given domain controller to generate audit records in the security event log on that DC. You can audit reads, writes, or both; you can audit the entire object or specific attributes. For more information, see [Retrieving an Object's SACL](#) and [Audit Generation](#).
- Event logging. By modifying registry settings on a given domain controller, you can change the kinds of events logged to the directory service event log. Specifically, to log all modifications, set the **8 Directory Access** value under the following registry key to 4.

```
HKEY_LOCAL_MACHINE  
  SYSTEM  
    CurrentControlSet  
      Services  
        NTDS  
          Diagnostics
```

For more information, see [Event Logging](#).

- Event tracing. Windows 2000 introduced an Event Tracing API for tracing and logging interesting events in software or hardware. The Windows operating system, and Active Directory Domain Services in particular, support the use of event tracing for capacity planning and detailed performance analysis. For more information, see [Event Tracing](#) and [Event Tracing in ADSI](#).

This section includes the following topics:

- [Overview of Change Tracking Techniques](#)
- [Change Notifications in Active Directory Domain Services](#)
- [Polling for Changes Using the DirSync Control](#)
- [Polling for Changes Using USNChanged](#)

Overview of Change Tracking Techniques

6/3/2022 • 4 minutes to read • [Edit Online](#)

There are several ways that change tracking mechanisms can differ:

- Scope for tracking changes: An application can track changes to a single attribute of a single object, to all objects in a domain, and so on. If the mechanism matches the requirements of the application, the application receives a minimum of irrelevant data, and thus this enhances performance.
- Timeliness: An application is notified of every change as it happens, or can be notified of the net effect of changes over a period of minutes or hours.

Processing less timely data may be more efficient, because several changes may be collapsed into one. For example, if an attribute changes three times within a one hour interval, an application notified of changes accumulated over an hour will be notified of just one attribute change, not three.

When thinking about timeliness, consider the effect of replication latency. An update that originates on one domain controller does not replicate to another domain controller instantly. Requiring change-tracking timeliness much better than the expected replication latency often gives no real benefit to the application.

- Polling versus notification: With polling, an application periodically makes a request to a domain controller to receive change tracking data. With notification the domain controller sends changes to the application only when changes occur.

The overhead of polling is obvious: The application may request change tracking data when nothing significant has occurred. The overhead of notification is more subtle. The server must maintain data about notification requests and must consult this data to decide whether or not to send a notification. This can add overhead to normal update requests.

- Expressing the application's knowledge: persistent versus temporary: Every change tracking mechanism must include some method for the server holding the data tracked to understand the application's state of knowledge, so that the idea of "change" is well defined. For example, the application's state of knowledge might be expressed as "Updated with all changes that occurred on DC d before time t." A mechanism based on this way of expressing an application's state of knowledge would provide an efficient way for the application to obtain changes that have occurred later than a specified time.

If the expression of the application's knowledge can be persisted, that is, stored recoverably, as in a file or database, application restart is less resource intensive than if it cannot. In the example above, the expression of the application's knowledge can be persisted by recording the DC d and the time t. Some change notification mechanisms do not allow this data to be persisted. The server and application must synchronize with some other mechanism when the application starts. This is resource-intensive if multiple objects are involved, and can involve complex programming.

Use the following techniques to track changes in Active Directory Domain Services:

- Use the change notification control to initiate a persistent asynchronous search for changes that match a specified filter. For more information, see [Change Notifications in Active Directory Domain Services](#).
- Use a directory synchronization (DirSync) search to retrieve changes that have occurred since the previous DirSync search. For more information, see [Polling for Changes Using the DirSync Control](#).
- Use the USNChanged attribute to search for objects that have changed since the previous search. For more information, see [Polling for Changes Using USNChanged](#).

The change notification control is designed for applications or services that require reasonably prompt notification of infrequent changes. An example is a service or program that stores configuration data on the Active Directory server and must be notified promptly when a change occurs. Be aware that there are limitations of the notification control.

- The promptness of notifications depends on replication latency and where the change occurred. You may be notified promptly when a change replicates into the replica you are monitoring, but the change may have originated much earlier on some other replica.
- The control is restricted to monitoring a single object or the immediate children of a container. Applications that must monitor multiple containers or unrelated objects can register up to five notification requests.
- If too many clients are listening for changes that occur frequently, it will impact the performance of the server. In general, applications should limit their use of this control for performance reasons on the server. If you do not need to know about changes immediately, it may be best to periodically poll for changes instead of using change notification.

The DirSync and USNChanged search techniques are designed for applications that maintain consistency between data on the Active Directory server and corresponding data in some other storage. These techniques are used by applications that periodically poll for changes. The DirSync technique is based on an LDAP server control that you can use through ADSI or LDAP APIs. The disadvantages of the DirSync control are that it can only be used by a highly privileged account, such as a domain administrator. The following is a list of limitations of DirSync control:

- The DirSync control can only be used by a highly privileged account, such as a domain administrator.
- The DirSync control can only monitor an entire naming context. You cannot limit the scope of a DirSync search to monitor only a specific subtree, container, or object in a naming context.

The USNChanged technique does not have these limitations, although it is somewhat more complicated to use than DirSync.

Change Notifications in Active Directory Domain Services

6/3/2022 • 3 minutes to read • [Edit Online](#)

Active Directory Domain Services provide a mechanism for a client application to register with a domain controller to receive change notifications. To do this, the client specifies the LDAP change notification control in an asynchronous LDAP search operation. The client also specifies the following search parameters.

PARAMETER	DESCRIPTION
Scope	Specify either LDAP_SCOPE_BASE to monitor just the object, or LDAP_SCOPE_ONELEVEL to monitor the immediate children of the object, not including the object itself. Do not specify LDAP_SCOPE_SUBTREE . Although the subtree scope is supported if the base object is the root of a naming context, its use can severely impact server performance, because it generates an LDAP search result message every time an object in the naming context is modified. You cannot specify LDAP_SCOPE_SUBTREE for an arbitrary subtree.
Filter	Specify a search filter of "(objectclass=*)", which means you receive notifications for changes to any object in the specified scope.
Attributes	Specify a list of attributes to return when a change occurs. Be aware that you receive notifications when any attribute is modified, not just the specified attributes.

You can register up to five notification requests on a single LDAP connection. You must have a dedicated thread that waits for the notifications and processes them quickly. When you call the `ldap_search_ext` function to register a notification request, the function returns a message identifier that identifies that request. You then use the `ldap_result` function to wait for change notifications. When a change occurs, the server sends you an LDAP message that contains the message identifier for the notification request that generated the notification. This causes the `ldap_result` function to return with search results that identify the object that changed.

The client application must determine the initial state of the object monitored. To do this, you must first register the notification request and then read the current state.

The client application must also determine the cause of the change. For a base level search, a notification occurs when any attribute changes, or when the object is deleted or moved. For a one-level search, a notification occurs when a child object is created, deleted, moved, or modified. Be aware that moving or renaming an object in the hierarchy above a target object does not generate a notification even though the distinguished name of the target changed as a result. For example, if monitoring changes to the child objects in a container, you will not receive notifications if the container itself is moved or renamed.

When the client processes the search results, it can use the `ldap_get_dn` function to get the distinguished name of the object that changed. Do not rely on distinguished names to identify the objects tracked, because distinguished names can change. Instead, include the `objectGUID` attribute in the list of attributes to retrieve. Each object's `objectGUID` remains unchanged regardless of where the object is moved within the enterprise forest.

If an object within the search scope is deleted, the client receives a change notification and the **isDeleted** attribute of the object is set to **TRUE**. In this case, the search results report the new distinguished name of the object in the Deleted Objects container of its partition. It is not necessary to specify the tombstone control ([LDAP_SERVER_SHOW_DELETED_OID](#)) to get notifications of object deletions. For more information, see [Retrieving Deleted Objects](#).

When a client has registered a notification request, the client continues to receive notifications until the connection is broken or the client abandons the search by calling the `ldap_abandon` function. If the client or server disconnects, for example, if the server fails, the notification request is terminated. When the client reconnects, it must register for notifications again, and then read the current state of the objects of interest in case there were changes while the client was disconnected.

The client can use the value of an object's **uSNChanged** attribute to determine whether the current state of the object on the server reflects the latest changes that the client has received. The system increases an object's **uSNChanged** attribute when the object is moved or modified. For example, if the server fails and the directory partition is restored from a backup, the server's replica of an object may not reflect changes previously reported to the client, in which case the **uSNChanged** value on the server will be lower than the value stored by the client.

For more information and a code example that uses the LDAP change notification control in an asynchronous LDAP search operation, see [Example Code for Receiving Change Notifications](#).

For more information about when to use the LDAP change notification control, see [Overview of Change Tracking Techniques](#).

Example Code for Receiving Change Notifications

6/3/2022 • 9 minutes to read • [Edit Online](#)

The following code example shows how to use the LDAP change notification control to receive notifications of changes to an object in Active Directory Domain Services. The example registers for notifications, reads the initial state of the object, and then uses a loop to wait for and process changes to the object.

First, the code example calls the [ldap_search_ext](#) function, which is an asynchronous search operation that returns after registering a notification request. Second, after setting up the notification request, the example calls the [ldap_search_s](#) function, which is a synchronous search operation that reads the current state of the object. Third, the example, uses a loop that calls [ldap_result](#) to wait for results of the asynchronous search operation. When the [ldap_result](#) function returns, the example processes the search results and repeats the loop.

Be aware that if the example read the current state and then set up the notification request, there would be a period of time during which changes could occur before the notification request was registered. By reading the object after setting up the notification request, the window works in reverse—you could receive notifications of changes that occurred before reading the initial state. To handle this, the example caches the object [uSNChanged](#) attribute value when it reads the object's initial state. Then, when [ldap_result](#) returns with a change notification, the example compares the cached [uSNChanged](#) attribute value with the value reported by [ldap_result](#). If the new [uSNChanged](#) attribute value is less than or equal to the cached value, the example discards the results because they indicate a change that occurred prior to the initial read operation.

This example performs a base level search that monitors a single object. You can specify the [LDAP_SCOPE_ONELEVEL](#) scope to monitor all child objects of the specified object. You can also modify the code to register up to five notification requests and then use the [ldap_result](#) function to wait for notifications from any of the requests. Remember that change notification requests impact the performance of the server. For more information about how to improve performance, see [Change Notifications in Active Directory Domain Services](#).

```
// Add Wldap32.lib to the project

#include <stdafx.h>
#include <windows.h>
#include <winldap.h>
#include <ntldap.h>
#include <stdio.h>
#include <rpcdce.h>
#include <wchar.h>

// Forward declarations
VOID BuildGUIDString(WCHAR *szGUID, LPBYTE pGUID);
BOOL ProcessResult(LDAP *pldapConnection, LDAPMessage *message, __int64 *piUSNChanged);

// GetChangeNotifications Function
//
// Description:
//   The function binds to an LDAP server,
//   registers for change notifications,
//   retrieves the current state, and
//   processes change notifications.
//
// Input Parameters:
//   szSearchBaseDN  The distinguished name of the object monitored.
//
// Return Values
//   0   The function was successful.
```

```

// 1 An error occurred.
INT GetChangeNotifications (LPWSTR szSearchBaseDN)
{
    INT          err = LDAP_OPERATIONS_ERROR; // Return value for LDAP operations.
    INT          iRetVal = 1;                  // Return value of this function.
    INT          inumberOfChanges = 3;        // The number of changes to monitor.
    BOOL         bSuccess = FALSE;           // Results from processing search results.
    ULONG        version = LDAP_VERSION3;   // LDAP version 3.
    LDAP*        pldapConnection = NULL;     // Pointer to the connection handle.
    LDAPControl  simpleControl;            // Structure used to represent both
                                            // client and server controls.
    PLDAPControl controlArray[2];          // Pointer to the LDAPControl.
    LONG         msgId;                   // Used to identify the message.
    LDAPMessage* presults = NULL;         // Pointer to LDAPMessage data structure
                                            // used to contain the search results.
    LDAPMessage* pmassage = NULL;         // Pointer to an LDAPMessage data
                                            // structure used to contain each
                                            // entry of the search results.
    LPWSTR       szAttrs[] = {             // Array of attributes whose value will
                                            // be displayed when the attribute
                                            // changes to a different value.

        {L"telephoneNumber"},           // L"telephoneNumber"
        {L"isDeleted"},                // L"isDeleted"
        {L"objectGUID"},               // L"objectGUID"
        {L"uSNChanged"},               // L"uSNChanged"
        NULL
    };
    __int64 iUSNChanged = 0;              // Stores the latest USNChanged value for the object.

    // Connect to the default LDAP server. If successful, the function returns a handle to the
    // connection.
    pldapConnection = ldap_open(NULL, // Connect to the default LDAP server.
                               0); // TCP port number.

    // If the connect function fails, show an error message and go to the error section.
    if (pldapConnection == NULL)
    {
        wprintf(L"ldap_open failed to connect. Error: 0x%x.\n", GetLastError());
        goto FatalExit0;
    }

    // The function succeeded in opening a connection to an LDAP server.
    wprintf(L"Connected to server.\n");

    // Set the options on connection blocks to specify LDAP version 3.
    pldapConnection->ld_lberoptions = 0;

    ldap_set_option(pldapConnection, LDAP_OPT_VERSION, &version);

    // Asynchronously authenticate a client to the LDAP server using the current credentials.
    err = ldap_bind_s(pldapConnection,           // Connection handle
                     NULL,                  // Null pointer
                     NULL,                  // Use the current credential.
                     LDAP_AUTH_NEGOTIATE); // Use the most appropriate authentication method.

    // If the bind function fails, show an error message and go to the error section.
    if (LDAP_SUCCESS != err)
    {
        wprintf(L"Bind failed: 0x%x\n", err);
        goto FatalExit0;
    }

    // The function succeeded in binding to the default LDAP server.
    wprintf(L"Successful bind.\n");

    // Set up the control to search the directory for objects that have
    // changed from a previous state.
    simpleControl.ldctl_oid = LDAP_SERVER_NOTIFICATION_OID_W;
}

```

```

simpleControl.ldctl_iscritical = TRUE;

// Set up the control to specify a BER-encoded sequence of parameters to the
// length of the binary data and to initialize a pointer.
simpleControl.ldctl_value.bv_len = 0;
simpleControl.ldctl_value.bv_val = NULL;

// Initialize the control array values.
controlArray[0] = &simpleControl;
controlArray[1] = NULL;

// Start a persistent asynchronous search.
err = ldap_search_ext(pldapConnection,
                      (PWCHAR) szSearchBaseDN,           // Connection handle.
                      LDAP_SCOPE_BASE,                 // Distinguished name of the object to monitor.
                      L"ObjectClass=*",                // Monitor the specified object.
                      szAttrs,                         // The search filter.
                      0,                               // Attributes to retrieve.
                      0,                               // Retrieve attributes and values.
                      (PLDAPControl *) &controlArray, // Server control.
                      NULL,                            // Client controls.
                      0,                               // No timeout.
                      0,                               // No size limit.
                      (PULONG)&msgId);              // Receives identifier for results.

// If the search function fails, show an error message and go to the error section.
if (LDAP_SUCCESS != err)
{
    wprintf(L" The asynch search failed. Error: 0x%x \n", err);
    goto FatalExit0;
}

// The asynchronous search function succeeded.
wprintf(L"Registered for change notifications on %s.\n", szSearchBaseDN);
wprintf(L"Message identifier is %d.\n", msgId);

// After starting the persistent search, perform a synchronous search
// to retrieve the current state of the monitored object.
err = ldap_search_s(pldapConnection,           // Connection handle.
                    (PWCHAR) szSearchBaseDN, // Distinguished name of the object to monitor.
                    LDAP_SCOPE_BASE,        // Monitor the specified object.
                    L"ObjectClass=*",       // The search filter.
                    szAttrs,               // List of attributes to retrieve.
                    0,                     // Retrieve attributes and values.
                    &presults);            // Receives the search results.

// If the synchronous search function fails, show an error message and go to
// the error section.
if (LDAP_SUCCESS != err)
{
    wprintf(L"ldap_search_s error: 0x%x\n", err);
    goto FatalExit0;
}

// The synchronous search function succeeded.
wprintf(L"\nGot current state\n");

// Process the search results by retrieving the first entry of a message
// and calling the function that processes the message. As long as the
// message contains an entry that is not null, retrieve and process the
// the next entry.
pmessage = ldap_first_entry(pldapConnection, presults);

while (pmessage != NULL)
{
    bSuccess = ProcessResult(pldapConnection, pmessage, &iUSNChanged);
    pmessage = ldap_next_entry(pldapConnection, pmessage);
}

// After processing the message, frees the results of the synchronous search routine.

```

```

ldap_msgfree(presults);
presults = NULL;

// Begin checking for changes on the specified object.
wprintf(L"Waiting for change notifications...\n");

// This loop monitors for changes on the specified object. The number of
// returned changes is specified.
for (INT icounter = 0;                                // Lower limit.
     icounter < inumberOfChanges;                      // Upper limit.
     icounter++)                                         // Increment the counter after each loop.
{
    // Wait for the results of the asynchronous search.
    presults = NULL;
    err = ldap_result(pldapConnection, // Connection handle.
                      LDAP_RES_ANY,   // Message identifier.
                      LDAP_MSG_ONE,   // Retrieve one message at a time.
                      NULL,           // No timeout.
                      &presults);      // Receives the search results.

    if ((err == (ULONG) -1) || (presults) == NULL)
    {
        wprintf(L"ldap_result error: 0x%lx\n", pldapConnection->ld_errno);
        break;
    }

    wprintf(L"\nGot a notification. Message ID: %d\n", presults->lmmsgid);

    // Process the search results by retrieving the first entry of a message
    // and calling the function that processes the message. As long as the
    // message contains an entry that is not null, retrieve and process the
    // the next entry.
    pmessage = ldap_first_entry(pldapConnection, presults);

    while (pmessage != NULL)
    {
        bSuccess = ProcessResult(pldapConnection, pmessage, &iUSNChanged);
        pmessage = ldap_next_entry(pldapConnection, pmessage);
    }

    // Free the resources used by the results.
    ldap_msgfree(presults);

    // Set the pointer to null.
    presults = NULL;
}

iRetVal = 0;
wprintf(L"The program returned the last %d changes.\n", inumberOfChanges);

// Error Section.
FatalExit0:

// If a connection succeeds, frees resources associated with an LDAP session.
if (pldapConnection) ldap_unbind(pldapConnection);

// Free resources used by the results.
if (presults) ldap_msgfree(presults);

return iRetVal;
}

// BuildGUIDString
//
// Description:
// The function turns the GUID into a string.
//
// Input parameters:

```

```

// szGUID  Pointer to a null-terminated string that will contain the
//          the string version of the GUID.
// pGUID   Pointer to the GUID to be converted to a string.
VOID BuildGUIDString(WCHAR* szGUID,LPBYTE pGUID)
{
    DWORD i = 0;
    DWORD dwlen = sizeof(GUID);
    WCHAR buf[4];
    wcscpy_s(szGUID,dwlen, L"");
    for (i; i < dwlen; i++)
    {
        swprintf_s(buf, L"%02x", pGUID[i]);
        wcscat_s(szGUID,dwlen,buf);
    }

}

// ProcessResult
//
// Description:
// This function processes and displays the search results.
//
// Input Parameters:
// pldapConnection     Pointer to the connection handle.
// pmessage            Pointer to the result entry to process.
// piUSNChanged        Pointer to the latest USNChanged value.
//
// Return Values:
// True    The function succeeded.
// False   The function failed.
BOOL ProcessResult(LDAP* pldapConnection,
                   LDAPMessage* pmessage,
                   __int64* piUSNChanged)
{
    PWCHAR* value = NULL;
    __int64 iNewUSNChanged;
    PWCHAR dn = NULL, attribute = NULL;
    BerElement *opaque = NULL;
    berval **pbvGUID=NULL;
    WCHAR szGUID[40];      // String version of the objectGUID attribute.
    ULONG count, total;

    // Get the uSNChanged attribute to determine if this result is new data.
    value = ldap_get_values(pldapConnection, pmessage, L"uSNChanged");

    // The attempt to get the attribute value failed so the function fails.
    if (!value)
    {
        wprintf(L"ldap_get_values error\n");
        return FALSE;
    }

    // Convert the attribute value from a string to an integer.
    iNewUSNChanged = _wtoi64(value[0]);

    // If this attribute value is less than or equal to the previous value, the result
    // contains out-of-date data. Therefore, discard the attribute value.
    if (iNewUSNChanged <= *piUSNChanged)
    {
        wprintf(L"Discarding outdated search results.\n");
        ldap_value_free(value);

        return TRUE;
    }
    else
    {
        *piUSNChanged = iNewUSNChanged;
        ldap_value_free(value);
    }
}

```

```

    }

    // The search results are newer than the previous state; process
    // the results.
    dn = ldap_get_dn(pldapConnection, pmessage);

    // If the function does not return a distinguished name, then an
    // error occurred.
    if (!dn)
    {
        wprintf(L"ldap_get_dn error\n");
        return FALSE;
    }

    // Display the distinguished name of the object.
    wprintf(L"    Distinguished Name is : %s\n", dn);

    // Free the resources used to get the distinguished name.
    ldap_memfree(dn);

    // Display the new values of the specified attributes
    attribute = ldap_first_attribute(pldapConnection, pmessage, &opaque);

    while (attribute != NULL)
    {
        // Handle objectGUID as a binary value.
        if (_wcsicmp(L"objectGUID", attribute)==0)
        {
            wprintf(L"    %s: ", attribute);
            pbvGUID = ldap_get_values_len (pldapConnection, pmessage, attribute);
            if (pbvGUID)
            {
                BuildGUIDString(szGUID, (LPBYTE) pbvGUID[0]->bv_val);
                wprintf(L"%s\n", szGUID);
            }
            ldap_value_free_len(pbvGUID);
        }
        else
        {
            // Handle other attributes as string values.
            value = ldap_get_values(pldapConnection, pmessage, attribute);
            wprintf(L"    %s: ", attribute);
            if (total = ldap_count_values(value) > 1)
            {
                for (count = 0; count < total; count++)
                    wprintf(L"        %s\n", value[count]);
            }
            else
            {
                wprintf(L"%s\n", value[0]);
                ldap_value_free(value);
            }
        }
        ldap_memfree(attribute);
        attribute = ldap_next_attribute(pldapConnection, pmessage, opaque);
    }

    return TRUE;
}

// Entry point for the application.
int wmain(int cArgs,WCHAR *pArgs[])
{
    PWCHAR szSearchBaseDN = NULL;
    wprintf(L"\n");
    if (cArgs < 2)
    {
        wprintf(L"Missing the distinguished name of the search base.\n");
        wprintf(L"Usage: <changes> <distinguished name of search base>\n");

```

```
    // API function usage.  GetChanges distinguishes between the search base and the current working directory.
    return 0;
}
szSearchBaseDN = (PWCHAR) pArgs[1];
return GetChangeNotifications(szSearchBaseDN);
```

Polling for Changes Using the DirSync Control

6/3/2022 • 6 minutes to read • [Edit Online](#)

Active Directory directory synchronization (DirSync) control is an LDAP server extension that enables an application to search an directory partition for objects that have changed since a previous state.

Use the DirSync control through ADSI by specifying the `ADS_SEARCHPREF_DIRSYNC` search preference when using [IDirectorySearch](#). For more information and a code example, see [Example Code Using ADS_SEARCHPREF_DIRSYNC](#). You can also perform a DirSync search using the LDAP API. The following describes the ADSI implementation, most of which also applies to using LDAP directly, except as discussed at the end of this topic.

When you perform a DirSync search, you pass in a provider-specific data element (cookie) that identifies the directory state at the time of the previous DirSync search. For the first search, you pass in a null cookie, and the search returns all objects that match the filter. The search also returns a valid cookie. Store the cookie in the same storage that you are synchronizing with the Active Directory server. On subsequent searches, get the cookie from storage and pass it with the search request. The search results now include only the objects and attributes that have changed since the previous state identified by the cookie. The search also returns a new cookie to store for the next search.

The following table lists search parameters that the client search request can specify.

PARAMETER	DESCRIPTION
Base of the search	The base of a DirSync search must be the root of a directory partition, which can be a domain partition, the configuration partition, or the schema partition.
Scope	The scope of a DirSync search must be <code>ADS_SCOPE_SUBTREE</code> , that is, the entire subtree of the partition. Be aware that for a search of a domain partition, the subtree includes the heads, but not the contents, of the configuration and schema partitions. To poll for changes in a smaller scope, use the <code>USNChanged</code> technique instead of DirSync.
Filter	You can specify any valid search filter. For an initial search with a null cookie, the results include all objects that match the filter. For subsequent searches with a valid cookie, the search results include data only for objects that match the filter and have changed since the state indicated by the cookie. For more information about search filters, see Creating a Query Filter .

PARAMETER	DESCRIPTION
Attributes	<p>You can specify a list of attributes to be returned when a change occurs. For each object, the initial results include all the requested attributes set on the object. Subsequent search results include only the specified attributes that have changed. Unchanged attributes are not included in the search results. In the ADSI implementation, the search results always include the objectGUID and instanceType of each object. Also, the specified attribute list acts as an additional filter: the initial search results include only objects that have at least one of the specified attributes set; subsequent searches include only objects on which one or more of the attributes have changed (values added or deleted).</p>

Also, be aware that:

- For incremental searches, the best practice is to bind to the same domain controller (DC) used in the previous search, that is, the DC that generated the cookie. If the same DC is unavailable, either wait until it is, or bind to a new DC and perform a full synchronization. Store the DNS name of the DC in the secondary storage with the cookie.

You can pass a cookie generated by one DC to a different DC hosting a replica of the same directory partition. There is no chance that a client will miss changes by using a cookie from one DC on another DC. However, it is possible that the search results from the new DC may include reported changes by the old DC; in some cases, the new DC may return all objects and attributes, as with a full synchronization. The client should just make its database consistent with reported search results for any given DirSync call, that is, handle all incremental results as if they were the latest state. It does not matter whether you have seen the change before or are even going back to a previous state because repeated incremental synchronizations will converge on consistency.

- When an object is renamed or moved, its child objects, if any, are not included in the search results, even though the distinguished names of the child objects have changed. Similarly, when an inheritable ACE is modified in an object security-descriptor, the child objects of the object are not included in the search results, even though the security-descriptors of the child objects have changed.
- Use the **objectGUID** attribute to identify the tracked objects. The **objectGUID** of each object remains unchanged regardless of where the object is moved within the forest.
- Be aware that the search results of a DirSync search indicate the state of the objects on a replica of the directory partition at the time of the search. This means that changes made on other DCs will not be included if they have not been replicated to the target DC. It also means that an object's attributes may have changed several times since the previous DirSync search, but the search will show only the final state, not the sequence of changes.
- In the ADSI implementation, the application must handle the cookie as opaque and not make any assumptions about its internal organization or value.
- Be aware that the client stores the cookie, cookie length, and DNS name of the DC in the same storage that contains the synchronized object data. This ensures that the cookie and other parameters remain in sync with the object data if the storage is ever restored from a backup.
- To retrieve the **parentGUID** attribute, which is constructed for the DirSync control, it is also necessary to request the **name** attribute.

To use the DirSync control, caller must have the "directory get changes" right assigned on the root of the partition being monitored. By default, this right is assigned to the Administrator and LocalSystem accounts on domain controllers. The caller must also have the [DS-Replication-Get-Changes](#) extended control access right. For more information about implementing a change-tracking mechanism for applications that must run under an account that does not have this right, see [Polling for Changes Using USNChanged](#). For more information about privileges, see [Privileges](#).

Retrieving Deleted Objects With a DirSync Search

The `ADS_SEARCHPREF_DIRSYNC` search results automatically include deleted objects (tombstones) that match the specified search filter. However, a search filter that will match an object when it is live may not match the object after it is deleted. This is because tombstones retain only a subset of the attributes present on the original object. For example, you would typically use the following filter for user objects.

```
(&(objectClass=user)(objectCategory=person))
```

The `objectCategory` attribute is removed when an object is deleted, so the filter above would not match any tombstone objects. Conversely, the `objectClass` attribute is retained on tombstone objects, so a filter of "`(objectClass=user)`" would match deleted user objects.

The attribute list that you specify with a DirSync search also acts as a filter; search results include only objects on which one or more of the specified attributes have changed since the previous DirSync search. If the attribute list does not include any attributes that are retained on tombstones, the search results will not include tombstones. To handle this, request all attributes by specifying a null attribute list; or you can request the `isDeleted` attribute, set to `TRUE` on all tombstones. Tombstone attributes have the `0x8` bit set in the `searchFlags` attribute of the `attributeSchema` definition.

For more information, see [Retrieving Deleted Objects](#).

LDAP Implementation of the DirSync Control

You can also perform a DirSync search by using the LDAP API with the `LDAP_SERVER_DIRSYNC_OID` control. If you use the LDAP API, also specify the `LDAP_SERVER_EXTENDED_DN_OID` and `LDAP_SERVER_SHOW_DELETED_OID` controls. The `LDAP_SERVER_EXTENDED_DN_OID` control causes an LDAP search to return an extended form of the distinguished name that includes the `objectGUID` and `objectSID` for security principal objects such as users, groups, and computers. The `LDAP_SERVER_SHOW_DELETED_OID` control causes the search results to include data for deleted objects. Be aware that these controls are automatically included in the ADSI implementation.

Example Code Using ADS_SEARCHPREF_DIRSYNC

6/3/2022 • 7 minutes to read • [Edit Online](#)

The following code example uses the ADSI implementation of the directory synchronization (DirSync) control to search the local domain partition of an Active Directory server for user objects changed since the previous call.

The code example uses the [IDirectorySearch](#) interface to search from the root of the domain partition. Before calling the [ExecuteSearch](#) method, the example calls the [SetSearchPreference](#) method to specify the [ADS_SEARCHPREF_SEARCH_SCOPE](#), [ADS_SEARCHPREF_DIRSYNC](#), and [ADS_SEARCHPREF_TOMBSTONE](#) search preferences. The scope must specify a subtree search. With the [ADS_SEARCHPREF_DIRSYNC](#) search preference, specify an [ADS_PROV_SPECIFIC](#) structure that contains the length of the cookie and a pointer to it.

The first time this application is called, it specifies a null cookie and a zero length. This causes the search operation to perform a full read, returning all the requested attributes for all objects that match the filter. Along with the search results, the server returns a valid cookie and the cookie length. On subsequent runs, the program retrieves the cached cookie and length and uses them to retrieve changes since the previous run.

Be aware that this sample simply caches the cookie and cookie length in the registry. In a real synchronization application, store the parameters in the same storage that you are keeping consistent with the Active Directory server. This ensures that the parameters and object data remain in sync if your database is ever restored from a backup.

```
// Add adsiid.lib to project
// Add activeds.lib to project

#include "stdafx.h"
#include "windows.h"
#include "stdio.h"
#include "activeds.h"
#include "adshlp.h"
#include "atldbase.h"
#include "iads.h"
#include "tchar.h"
#include "string.h"
#include "mbstring.h"
#include "stdlib.h"

typedef struct {WCHAR objectGUID[40];
                WCHAR ADsPath[MAX_PATH];
                WCHAR phoneNumber[32];
                BOOL isDeleted;
            } MyUserData;

// Forward declaration.
VOID BuildGUIDString(WCHAR *szGUID, LPBYTE pGUID);
VOID WriteObjectDataToStorage(MyUserData *userdata, BOOL bUpdate);

// DoDirSyncSearch
HRESULT DoDirSyncSearch(
    LPWSTR pszSearchFilter, // Search filter.
    LPWSTR *pAttributeNames, // Attributes to retrieve.
    DWORD dwAttributes, // Number of attributes.
    PUCHAR *ppCookie, // Pointer to previous cookie.
    PULONG pulCookieLength, // Length of previous cookie.
    LPWSTR szDC) // Name of DC to bind to.
{
    IADs *pRootDSE = NULL;
```

```

IDirectorySearch *pSearch = NULL;
ADS_SEARCH_HANDLE hSearch = NULL;
ADS_SEARCHPREF_INFO arSearchPrefs[3];
ADS_PROVIDERSPECIFIC dirsSync;
ADS_SEARCH_COLUMN col;
HRESULT hr = S_OK;
VARIANT var;
MyUserData userdata;
BOOL bUpdate = FALSE;
DWORD dwCount = 0;

// Validate input parameters.
if (!pu1CookieLength || !ppCookie || !szDC)
{
    wprintf(L"Invalid parameter.\n");
    return E_FAIL;
}

LPOLESTR szDSPPath = new OLECHAR[MAX_PATH];
LPOLESTR szServerPath = new OLECHAR[MAX_PATH];
VariantInit(&var);

// If cookie is non-NULL, this is an update. Otherwise, it is a full-read.
if (*ppCookie)
    bUpdate = TRUE;
CoInitialize(NULL);

// If there is a DC name from the previous USN sync,
// include it in the binding string.
if (szDC[0])
{
    wcsncpy_s(szServerPath,MAX_PATH,L"LDAP://",MAX_PATH);
    wcsncat_s(szServerPath, MAX_PATH,szDC[MAX_PATH-wcslen(szServerPath)]);
    wcsncat_s(szServerPath, MAX_PATH,L"/",MAX_PATH-wcslen(szServerPath));
}
else
    wcsncpy_s(szServerPath, MAX_PATH,L"LDAP://",MAX_PATH);

// Bind to root DSE.
wcsncpy_s(szDSPPath,MAX_PATH,szServerPath,MAX_PATH);
wcsncat_s(szDSPPath, MAX_PATH,L"rootDSE",MAX_PATH-wcslen(szDSPPath));
wprintf(L"RootDSE binding string: %s\n", szDSPPath);
hr = AdsGetObject(szDSPPath,
                  IID_IADS,
                  (void**)&pRootDSE);
if (FAILED(hr))
{
    wprintf(L"failed to bind to rootDSE: 0x%x\n", hr);
    goto cleanup;
}

// Save the name of the DC connected to in order to connect to
// the same DC on the next dirsSync operation.
if (! szDC[0])
{
    hr = pRootDSE->Get(CComBSTR("DnsHostName"), &var);
    wcsncpy_s(szServerPath,MAX_PATH,L"LDAP://",MAX_PATH);
    wcsncat_s(szServerPath, MAX_PATH,var.bstrVal, MAX_PATH-wcslen(szServerPath));
    wcsncat_s(szServerPath, MAX_PATH,L"/", MAX_PATH-wcslen(szServerPath));
}

// Get an IDirectorySearch pointer to the root of the domain partition.
hr = pRootDSE->Get(CComBSTR("defaultNamingContext"), &var);
wcsncpy_s(szDSPPath,MAX_PATH,szServerPath,MAX_PATH);
wcsncat_s(szDSPPath, MAX_PATH,var.bstrVal, MAX_PATH - wcslen(szDSPPath));
hr = AdsGetObject(szDSPPath, IID_IDirectorySearch, (void**) &pSearch);
if (FAILED(hr))
{
    wprintf(L"failed to get IDirectorySearch: 0x%x\n", hr);
}

```

```

        goto cleanup;
    }

    // Initialize the structure to pass in the cookie.
    // On the first call, the cookie is NULL and the length is zero.
    // On later calls, the cookie and length are the values returned by
    // the previous call.
    dirsync.dwLength = *pulCookieLength;
    dirsync.lpValue = *ppCookie;
    arSearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    arSearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
    arSearchPrefs[0].vValue.Integer = ADS_SCOPE_SUBTREE;
    arSearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_DIRSYNC;
    arSearchPrefs[1].vValue.dwType = ADSTYPE_PROV_SPECIFIC;
    arSearchPrefs[1].vValue.ProviderSpecific = dirsync;
    hr = pSearch->SetSearchPreference(arSearchPrefs, 2);
    if (FAILED(hr))
    {
        wprintf(L"failed to set search prefs: 0x%x\n", hr);
        goto cleanup;
    }

    // Search for the objects indicated by the search filter.
    hr = pSearch->ExecuteSearch(pszSearchFilter,
                                pAttributeName, dwAttributes, &hSearch );
    if (FAILED(hr))
    {
        wprintf(L"failed to set execute search: 0x%x\n", hr);
        goto cleanup;
    }

    // Loop through the rows of the search result
    // Each row is an object that has changed since the previous call.
    hr = pSearch->GetNextRow( hSearch);
    while ( SUCCEEDED(hr) && hr != S_ADS_NOMORE_ROWS )
    {
        ZeroMemory(&userdata, sizeof(MyUserData) );

        // Retrieve the attributes for each object.
        // With a DirSync search, a GetColumn call can return success even
        // though no values are set for the specified attribute. If this
        // happens, col.pADsValues is NULL; the following code checks
        // col.pADsValues before trying to access it.
        // Get the telephone number.
        hr = pSearch->GetColumn( hSearch, L"telephoneNumber", &col );
        if ( SUCCEEDED(hr) )
        {
            if (col.dwADsType == ADSTYPE_CASE_IGNORE_STRING && col.pADsValues)
                wcscpy_s(userdata.phoneNumber, col.pADsValues->CaseIgnoreString);
            pSearch->FreeColumn( &col );
        }

        // Get the isDeleted attribute.
        hr = pSearch->GetColumn( hSearch, L"isDeleted", &col );
        if ( SUCCEEDED(hr) )
        {
            if (col.dwADsType == ADSTYPE_BOOLEAN && col.pADsValues)
                userdata.isDeleted = col.pADsValues->Boolean;
            pSearch->FreeColumn( &col );
        }

        // Get the ADsPath.
        hr = pSearch->GetColumn( hSearch, L"ADsPath", &col );
        if ( SUCCEEDED(hr) )
        {
            if (col.dwADsType == ADSTYPE_CASE_IGNORE_STRING && col.pADsValues)
                wcscpy_s(userdata.ADspath, col.pADsValues->CaseIgnoreString);
            pSearch->FreeColumn( &col );
        }
    }
}

```

```

// Get the objectGUID number.
hr = pSearch->GetColumn( hSearch, L"objectGUID", &col );
if ( SUCCEEDED(hr) )
{
    WCHAR szGUID[40]; // string version of the objectGUID attribute
    if ((col.dwADsType == ADSTYPE_OCTET_STRING) && col.pADsValues &&
        (col.pADsValues->OctetString.lpValue))
    {
        BuildGUIDString(szGUID, (LPBYTE) col.pADsValues->OctetString.lpValue);
        wcscpy_s(userdata.objectGUID, szGUID);
    }
    pSearch->FreeColumn( &col );
}

WriteObjectDataToStorage(&userdata, bUpdate);
dwCount++;
hr = pSearch->GetNextRow( hSearch );
}
wprintf(L"dwCount: %d\n", dwCount);

// After looping through the results, get the cookie.
if (hr == S_ADS_NOMORE_ROWS )
{
    hr = pSearch->GetColumn( hSearch, ADS_DIRSYNC_COOKIE, &col );
    if ( SUCCEEDED(hr) ) {
        if (col.dwADsType == ADSTYPE_PROV_SPECIFIC && col.pADsValues)
        {
            wprintf(L"Got cookie\n");
            *pulCookieLength = col.pADsValues->ProviderSpecific.dwLength;
            *ppCookie = (P UCHAR) AllocADsMem (*pulCookieLength);
            memcpy(*ppCookie, col.pADsValues->ProviderSpecific.lpValue,
                   *pulCookieLength);
        }
        pSearch->FreeColumn( &col );
    } else
        wprintf(L"no cookie: 0x%llx\n", hr);
}

cleanup:
if (pRootDSE)
    pRootDSE->Release();
if (hSearch)
    pSearch->CloseSearchHandle(hSearch);
if (pSearch)
    pSearch->Release();

VariantClear(&var);
CoUninitialize();
delete [] szServerPath;
delete [] szDSPath;

return hr;
}

// WriteObjectDataToStorage routine
VOID WriteObjectDataToStorage(MyUserData *userdata, BOOL bUpdate)
{
    if (bUpdate)
        wprintf(L"UPDATE:\n");
    else
        wprintf(L"INITIAL DATA:\n");
    wprintf(L"    objectGUID: %s\n", userdata->objectGUID);
    wprintf(L"    ADsPath: %s\n", userdata->ADsPath);
    wprintf(L"    phoneNumber: %s\n", userdata->phoneNumber);
    if (userdata->isDeleted)
        wprintf(L"    DELETED OBJECT\n");
    wprintf(L"-----\n");
    return;
}

```

```

}

// WriteCookieAndDCtoStorage routine
// This example caches the cookie in the registry. In a real
// synchronization application, store these parameters in the
// same storage that you are keeping consistent with Active Directory.
// This ensures that the parameters and object data remain in sync if
// the storage is ever restored from a backup.
DWORD WriteCookieAndDCtoStorage(UCHAR *pCookie,
                                ULONG ulLength,
                                WCHAR *pszDCName)
{
    HKEY hReg = NULL;
    DWORD dwStat = NO_ERROR;

    // Create a registry key under
    // HKEY_CURRENT_USER\SOFTWARE\Vendor\Product.
    dwStat = RegCreateKeyExW(HKEY_CURRENT_USER,
                           L"Software\\Microsoft\\Windows 2000 AD-Synchro-DirSync",
                           0,
                           NULL,
                           REG_OPTION_NON_VOLATILE,
                           KEY_ALL_ACCESS,
                           NULL,
                           &hReg,
                           NULL);

    if (dwStat != NO_ERROR)
    {
        wprintf(L"RegCreateKeyEx failed: 0x%x\n", dwStat);
        return dwStat;
    }

    // Cache the cookie as a value under the registry key.
    dwStat = RegSetValueExW(hReg, L"Cookie", 0, REG_BINARY,
                           (const BYTE *)pCookie, ulLength);

    if (dwStat != NO_ERROR)
        wprintf(L"RegSetValueEx for cookie failed: 0x%x\n", dwStat);

    // Cache the cookie length as a value under the registry key.
    dwStat = RegSetValueExW(hReg, L"Cookie Length", 0, REG_DWORD,
                           (const BYTE *)&ulLength, sizeof(DWORD));
    if (dwStat != NO_ERROR)
        wprintf(L"RegSetValueEx for cookie length failed: 0x%x\n", dwStat);

    // Cache the DC name as a value under the registry key.
    dwStat = RegSetValueExW(hReg, L"DC name", 0, REG_SZ,
                           (const BYTE *)pszDCName, 2*(wcslen(pszDCName)) );
    if (dwStat != NO_ERROR)
        wprintf(L"RegSetValueEx for DC name failed: 0x%x\n", dwStat);

    RegCloseKey(hReg);
    return dwStat;
}

// GetCookieAndDCfromStorage routine
DWORD GetCookieAndDCfromStorage(PUCHAR *ppCookie,           // Receives pointer to cookie.
                                PULONG pulCookieLength, // Receives length of cookie.
                                WCHAR *pszDCName)      // Receives name of DC to bind to.
{
    HKEY hReg = NULL;
    DWORD dwStat;
    DWORD dwLen;

    // Open the registry key.
    dwStat = RegOpenKeyExW(HKEY_CURRENT_USER,
                           L"Software\\Microsoft\\Windows 2000 AD-Synchro-DirSync",
                           0,
                           KEY_QUERY_VALUE,
                           &hReg);

```

```

        ...
    }

    if (dwStat != NO_ERROR)
    {
        wprintf(L"RegOpenKeyEx failed: 0x%x\n", dwStat);
        return dwStat;
    }

    // Get the length of the cookie from the registry.
    dwLen = sizeof(DWORD);
    dwStat = RegQueryValueExW(hReg, L"Cookie Length", NULL, NULL,
                             (LPBYTE)pulCookieLength, &dwLen );
    if (dwStat != NO_ERROR) {
        wprintf(L"RegQueryValueEx failed to get length: 0x%x\n", dwStat);
        return dwStat;
    }

    // Allocate a buffer for the cookie value.
    *ppCookie = (P UCHAR) GlobalAlloc(GPTR, *pulCookieLength);
    if (!*ppCookie)
    {
        wprintf(L"GlobalAlloc failed: %u\n", GetLastError() );
        return dwStat;
    }

    // Get the cookie from the registry.
    dwStat = RegQueryValueExW(hReg, L"Cookie", NULL, NULL,
                             (LPBYTE)*ppCookie, pulCookieLength );
    if (dwStat != NO_ERROR) {
        wprintf(L"RegQueryValueEx failed to get cookie: 0x%x\n", dwStat);
        return dwStat;
    }

    // Get the DC name from the registry.
    dwLen = MAX_PATH;
    dwStat = RegQueryValueExW(hReg, L"DC name", NULL, NULL,
                             (LPBYTE)pszDCName, &dwLen );
    if (dwStat != NO_ERROR) {
        wprintf(L"RegQueryValueEx failed to get DC name: 0x%x\n", dwStat);
        return dwStat;
    }
    else
        pszDCName[dwLen] = 0;
    RegCloseKey(hReg);
    return NO_ERROR;
}

// BuildGUIDString
// Routine that makes the GUID into a string in directory service bind form
VOID
BuildGUIDString(WCHAR *szGUID, LPBYTE pGUID)
{
    DWORD i = 0x0;
    DWORD dwlen = sizeof(GUID);
    WCHAR buf[4];

    wcscpy_s(szGUID,MAX_PATH,L"");

    for (i;i<dwlen;i++)
    {
        swprintf_s(buf, L"%02x", pGUID[i]);
        wcscat_s(szGUID, MAX_PATH,buf);
    }
}

// Entry point for application
int main(int argc, char* argv[])
{
    DWORD dwStat;
    ULONG ulLength;
    ...
}

```

```

UCHAR *pCookie;
WCHAR szDCName[MAX_PATH];
HRESULT hr;

if (argc==1)
{
    // Perform a full synchronization.
    // Initialize the synchronization parameters to zero or NULL.
    wprintf(L"Performing a full sync.\n");
    szDCName[0] = '\0';
    ulLength = 0;
    pCookie = NULL;
}
else
{
    // Perform an incremental synchronization.
    // Initialize synchronization parameters from storage.
    wprintf(L"Retrieving changes only.\n");
    dwStat = GetCookieAndDCfromStorage(&pCookie, &ulLength, szDCName);
    if (dwStat != NO_ERROR)
    {
        wprintf(L"Could not get the cookie: %u\n", dwStat);
        goto cleanup;
    }
}

// Perform the search and update the synchronization parameters.
hr = DoDirSyncSearch(L"(objectClass=user)",
                      NULL,           // Retrieve all attributes.
                      -1,
                      &pCookie,
                      &ulLength,
                      szDCName);

if (FAILED(hr))
{
    wprintf(L"DoDirSyncSearch failed: 0x%x\n", hr);
    goto cleanup;
}

// Cache the returned synchronization parameters in storage.
wprintf(L"Caching the synchronization parameters.\n");
dwStat = WriteCookieAndDCtoStorage(pCookie, ulLength, szDCName);
if (dwStat != NO_ERROR)
{
    wprintf(L"Could not cache the cookie: %u\n", dwStat);
    goto cleanup;
}

cleanup:
    if (pCookie)
        GlobalFree (pCookie);
    return 1;
}

```

Polling for Changes Using USNChanged

6/3/2022 • 5 minutes to read • [Edit Online](#)

The DirSync control is powerful and efficient, but has two significant limitations:

- Only for highly privileged applications: To use the DirSync control, an application must run under an account that has the `SE_SYNC_AGENT_NAME` privilege on the domain controller. Few accounts are so highly privileged, so an application that uses the DirSync control cannot be run by ordinary users.
- No subtree scoping: The DirSync control returns all changes that occur within a naming context. An application interested only in changes that occur in a small subtree of a naming context must wade through many irrelevant changes, which is inefficient both for the application and for the domain controller.

Changes from Active Directory can also be obtained by querying the `uSNChanged` attribute, which avoids the limitations of the DirSync control. This alternative is not better than the DirSync control in all respects because it involves transmitting all attributes when any attribute changes and it requires more work from the application developer to handle certain failure scenarios correctly. It is, currently, the best way to write certain change-tracking applications.

When a domain controller modifies an object it sets that object's `uSNChanged` attribute to a value that is larger than the previous value of the `uSNChanged` attribute for that object, and larger than the current value of the `uSNChanged` attribute for all other objects held on that domain controller. As a consequence, an application can find the most recently changed object on a domain controller by finding the object with the largest `uSNChanged` value. The second most recently changed object on a domain controller will have the second largest `uSNChanged` value, and so on.

The `uSNChanged` attribute is not replicated, therefore reading an object's `uSNChanged` attribute at two different domain controllers will typically give different values.

For example, the `uSNChanged` attribute can be used to track changes in a subtree S. First, perform a "full sync" of the subtree S. Suppose the largest `uSNChanged` value for any object in S is U. Periodically query for all objects in subtree S whose `uSNChanged` value is greater than U. The query will return all objects that have changed since the full sync. Set U to the largest `uSNChanged` among these changed objects, and you are ready to poll again.

The subtleties of implementing a `uSNChanged` synchronization application include:

- Use the `highestCommittedUSN` attribute of rootDSE to bound your `uSNChanged` filters. That is, before starting a full sync, read the `highestCommittedUSN` of your affiliated domain controller. Then, perform a full synchronization query (using paged results) to initialize the database. When this is complete, store the `highestCommittedUSN` value read before the full sync query; to use as the lower bounds of the `uSNChanged` attribute for the next synchronization. Later, to perform an incremental synchronization, reread the `highestCommittedUSN` rootDSE attribute. Then query for relevant objects, using paged results, whose `uSNChanged` is greater than the lower bounds of the `uSNChanged` attribute value saved from the previous synchronization. Update the database using this information. When complete, update the lower bounds of the `uSNChanged` attribute from the `highestCommittedUSN` value read before the incremental synchronization query. Always store the lower bounds of the `uSNChanged` attribute value in the same storage that the application is synchronizing with the domain controller content.

Following this procedure, rather than that based on `uSNChanged` values on retrieved objects, avoids making the server reexamine updated objects that fall outside the set that is applicable to the application.

- Because **uSNChanged** is a non-replicated attribute, the application must bind to the same domain controller every time it runs. If it cannot bind to that domain controller it must either wait until it can do so, or affiliate with some new domain controller and perform a full synchronization with that domain controller. When the application affiliates with a domain controller it records the DNS name of that domain controller in stable storage, which is the same storage it is keeping consistent with the domain controller content. Then it uses the stored DNS name to bind to the same domain controller for subsequent synchronizations.
- The application must detect when the domain controller it is currently affiliated with has been restored from backup, because this can cause inconsistency. When the application affiliates with a domain controller it caches the "invocation id" of that domain controller in stable storage, that is, the same storage it is keeping consistent with the domain controller content. The "invocation id" of a domain controller is a GUID stored in the **invocationID** attribute of the domain controller's service object. To get the distinguished name of a domain controller's service object, read the **dsServiceName** attribute of the rootDSE.

Be aware that when the application's stable storage is restored from backup there are no consistency problems because the domain controller name, invocation id, and lower bounds of the **uSNChanged** attribute value are stored with the data synchronized with the domain controller content.

- Use paging when querying the server, both full and incremental synchronizations, to avoid the possibility of retrieving large result sets simultaneously. For more information, see [Specifying Other Search Options](#).
- Perform index-based queries to avoid forcing the server to store large intermediate results when using paged results. For more information, see [Indexed Attributes](#).
- In general, do not use server-side sorting of search results, which can force the server to store and sort large intermediate results. This applies to both full and incremental synchronizations. For more information, see [Specifying Other Search Options](#).
- Handle no parent conditions gracefully. The application may recognize an object before it has recognized its parent. Depending upon the application, this may or may not be a problem. The application can always read the current state of the parent from the directory.
- To handle moved or deleted objects, store the **objectGUID** attribute of each tracked object. An object's **objectGUID** attribute remains unchanged regardless of where it is moved throughout the forest.
- To handle moved objects, either perform periodic full synchronizations or increase the search scope and filter out uninteresting changes at the client end.
- To handle deleted objects, either perform periodic full synchronizations or perform a separate search for deleted objects when you perform an incremental synchronization. When you query for deleted objects, retrieve the **objectGUID** of the deleted objects to determine the objects to delete from your database. For more information, see [Retrieving Deleted Objects](#).
- Be aware that the search results include only the objects and attributes that the caller has permission to read (based on the security descriptors and DACLs on the various objects). For more information, see [Effects of Security on Queries](#).

For more information, and a code example that shows the basics of a USNChanged synchronization application, see [Example Code to Retrieve Changes Using USNChanged](#).

Example Code to Retrieve Changes Using USNChanged

6/3/2022 • 12 minutes to read • [Edit Online](#)

The following code example uses the **uSNChanged** attribute of an object in Active Directory Domain Services to retrieve changes since a previous query. The code example can perform either a full synchronization or an incremental update. For a full synchronization, the sample application connects to the rootDSE of a domain controller and reads the following parameters that it stores to be used in the next incremental synchronization:

- The DNS name of the DC. Incremental synchronizations must be performed on the same DC as the previous synchronization.
- The **invocationID** GUID of the DC. The code example uses this value to detect that the DC has been restored from a backup, in which case, the sample must perform a full synchronization.
- The **highestCommittedUSN**. This value becomes the lower bound for the **uSNChanged** filter on the next incremental synchronization.

The code example uses the **IDirectorySearch** interface, specifying the distinguished name of the base of the search, a search scope, and a filter. There are no restrictions on the search base or scope. In addition to specifying the objects of interest, the filter must also specify a **uSNChanged** comparison, such as "**uSNChanged >= <lower bound USN>**". For a full synchronization, "**<lower bound USN>**" is zero. For an incremental synchronization, it is the 1 plus the **highestCommittedUSN** value from the previous search.

Be aware that this sample application is intended only to show how to use **uSNChanged** to retrieve changes from the Active Directory server. It prints the changes and does not actually synchronize the data in a secondary storage. Consequently, it does not show how to handle issues like moved objects or "no parent" conditions. It does show how to retrieve deleted objects, but it does not show how an application uses the **objectGUID** of the deleted objects to determine the corresponding object to delete in the storage.

Also, the sample caches the DC name, **invocationID**, and **highestCommittedUSN** in the registry. In a real synchronization application, you must store the parameters in the same storage that are kept consistent with the Active Directory server. This ensures that the parameters and object data remain synchronized if your database is ever restored from a backup.

```
#include <windows.h>
#include <stdio.h>
#include <activeds.h>
#include <ntdsapi.h>
#include <atldbbase.h>

#pragma comment(lib, "activeds")
#pragma comment(lib, "adsiid")

typedef struct _MYUSERDATA
{
    WCHAR objectGUID[40];
    WCHAR distinguishedName[MAX_PATH];
    WCHAR phoneNumber[32];
} MYUSERDATA, *PMYUSERDATA;

#define ARRAYSIZE(__buf__) (sizeof(__buf__)/sizeof(__buf__[0]))

// Forward declaration
VOID BuildGUIDString(WCHAR *szGUID, LPBYTE pGUID); VOID WriteObjectDataToStorage(PMYUSERDATA pUserData, BOOL bUpdate); VOID DeleteObjectDataFromStorage(PMYUSERDATA pUserData);
```

```

//*****
// DoUSNSyncSearch
//*****

HRESULT DoUSNSyncSearch(
    LPWSTR pszSearchBaseDN,          // Distinguished name of search base
    ULONG ulScope,                  // Scope of the search
    LPWSTR *pAttributeNames,        // Attributes to retrieve
    DWORD dwAttributes,             // Number of attributes
    LPWSTR pszPrevInvocationID,     // GUID string for DC's invocationID
    LPWSTR pszPrevHighUSN,          // Highest USN from previous sync
    LPWSTR pszDC)                  // Name of DC to bind to
{
    LPWSTR pszServerPath = NULL;
    LPWSTR pszDSPath = NULL;
    IADs *pRootDSE = NULL;
    IADs *pDCService = NULL;
    IADs *pDeletedObj = NULL;
    IDirectorySearch *pSearch = NULL;
    ADS_SEARCH_HANDLE hSearch = NULL;
    ADS_SEARCHPREF_INFO arSearchPrefs[3];
    WCHAR szSearchFilter[256];      // Search filter
    ADS_SEARCH_COLUMN col;
    MYUSERDATA userdata;
    void HUGEP *pArray;
    WCHAR szGUID[40];
    INT64 iLowerBoundUSN;
    HRESULT hr = E_FAIL;
    DWORD dwCount = 0;
    VARIANT var;
    BOOL bUpdate = TRUE;

    // Validate input parameters.
    if (!pszPrevInvocationID || !pszPrevHighUSN || !pszDC)
    {
        wprintf(L"Invalid parameter.\n");
        return E_INVALIDARG;
    }

    VariantInit(&var);

    // Allocate the server path string buffer.
    pszServerPath = new WCHAR[7 + wcslen(pszDC) + 1 + 1];
    if(!pszServerPath)
    {
        wprintf(L"failed to allocate memory");
        hr = E_OUTOFMEMORY;
        goto cleanup;
    }

    // If there exists a DC name from the previous USN synchronization,
    // include it in the binding string.
    if (pszDC[0])
    {
        wcscpy_s(pszServerPath, L"LDAP://");
        wcscat_s(pszServerPath, pszDC);
        wcscat_s(pszServerPath, L"/");
    }
    else
    {
        wcscpy_s(pszServerPath, L"LDAP://");
    }

    // Allocate the DS path string buffer.
    pszDSPath = new WCHAR[wcslen(pszServerPath) + 7 + 1];
    if(!pszDSPath)
    {
        wprintf(L"failed to allocate memory");
        hr = E_OUTOFMEMORY;
    }
}

```

```

        goto cleanup;
    }

    // Bind to the root DSE.
    wcscpy_s(pszDSPPath, pszServerPath);
    wcscat_s(pszDSPPath, L"rootDSE");
    hr = ADsOpenObject(pszDSPPath,
                       NULL,
                       NULL,
                       ADS_SECURE_AUTHENTICATION,
                       IID_IADs,
                       (void**)&pRootDSE);
    if (FAILED(hr))
    {
        wprintf(L"failed to bind to root: 0x%x\n", hr);
        goto cleanup;
    }

    // Get the name of the DC connected to.
    hr = pRootDSE->Get(CComBSTR("DnsHostName"), &var);
    if (FAILED(hr))
    {
        wprintf(L"failed to get DnsHostName: 0x%x\n", hr);
        goto cleanup;
    }

    // Compare it to the DC name from the previous USN sync operation.
    // If not the same, perform a full synchronization.
    if (_wcsicmp(pszDC, var.bstrVal) != 0)
    {
        bUpdate = FALSE;
        // This prevents a full sync being performed
        // EVERY time
        wcscpy_s( pszDC, var.bstrVal );

        // Reallocate the string buffer.
        delete[] pszServerPath;
        pszServerPath = new WCHAR[7 + wcslen(var.bstrVal) + 1 + 1];
        if(!pszServerPath)
        {
            wprintf(L"failed to allocate memory");
            hr = E_OUTOFMEMORY;
            goto cleanup;
        }

        // Use the DC name in the bind string prefix.
        wcscpy_s(pszServerPath, L"LDAP://");
        wcscat_s(pszServerPath, var.bstrVal);
        wcscat_s(pszServerPath, L"/");
    }

    // Bind to the DC service object to get the invocationID.
    // The dsServiceName property of root DSE contains the distinguished
    // name of this DC service object.
    VariantClear(&var);
    hr = pRootDSE->Get(CComBSTR("dsServiceName"), &var);
    if (FAILED(hr))
    {
        wprintf(L"failed to get \"dsServiceName\"\n");
        goto cleanup;
    }

    // Reallocate the DS path buffer.
    delete[] pszDSPPath;
    pszDSPPath = new WCHAR[wcslen(pszServerPath) + wcslen(var.bstrVal) + 1];
    if(!pszDSPPath)
    {
        wprintf(L"failed to allocate memory");
        hr = E_OUTOFMEMORY;
    }
}

```

```

        goto cleanup;
    }

wcscpy_s(pszDSPPath, pszServerPath);
wcscat_s(pszDSPPath, var.bstrVal);
hr = ADsOpenObject(pszDSPPath,
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION,
                    IID_IADs,
                    (void**)&pDCService);
VariantClear(&var);
if (FAILED(hr))
{
    wprintf(L"failed to bind to the DC service object: 0x%x\n", hr);
    goto cleanup;
}

// Get the invocationID GUID from the service object.
hr = pDCService->Get(CComBSTR("invocationID"), &var);
if (FAILED(hr))
{
    wprintf(L"failed to get \"invocationID\"\n");
    goto cleanup;
}

hr = SafeArrayAccessData((SAFEARRAY*)(var.parray), (void *HUGECP* FAR*)&pArray);
if (FAILED(hr))
{
    wprintf(L"failed to get hugecp: 0x%x\n", hr);
    goto cleanup;
}

BuildGUIDString(szGUID, (LPBYTE)pArray);
VariantClear(&var);

// Compare the invocationID GUID to the GUID string from the previous
// synchronization. If not the same, this is a different DC or the DC
// was restored from backup; perform a full synchronization.
if (_wcsicmp(szGUID, pszPrevInvocationID)!=0)
{
    bUpdate = FALSE;
    wcscpy_s(pszPrevInvocationID, szGUID); // Save the invocationID GUID.
}

// If previous high USN is an empty string, handle this as a full synchronization.
bUpdate = (pszPrevHighUSN[0] != '\0');

// Set the lower bound USN to zero if this is a full synchronization.
// Otherwise, set it to the previous high USN plus one.
if (bUpdate == FALSE)
{
    iLowerBoundUSN = 0;
}
else
{
    iLowerBoundUSN = _wtoi64(pszPrevHighUSN) + 1; // Convert the string to an integer.
}

// Get and save the current high USN.
hr = pRootDSE->Get(CComBSTR("highestCommittedUSN"), &var);
if (FAILED(hr))
{
    wprintf(L"failed to get \"highestCommittedUSN\"\n");
    goto cleanup;
}

wcscpy_s(pszPrevHighUSN, var.bstrVal);
wprintf(L"current highestCommittedUSN: %s\n". pszPrevHighUSN);

```

```

VariantClear(&var);

// Reallocate the DS path buffer.
delete[] pszDSPPath;
pszDSPPath = new WCHAR[wcslen(pszServerPath) + wcslen(pszSearchBaseDN) + 1];
if(!pszDSPPath)
{
    wprintf(L"failed to allocate memory");
    hr = E_OUTOFMEMORY;
    goto cleanup;
}

// Get an IDirectorySearch pointer to the base of the search.
wcscpy_s(pszDSPPath, pszServerPath);
wcscat_s(pszDSPPath, pszSearchBaseDN);
hr = ADsOpenObject(pszDSPPath,
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION,
                    IID_IDirectorySearch,
                    (void**)&pSearch);
if (FAILED(hr))
{
    wprintf(L"failed to get IDirectorySearch: 0x%x\n", hr);
    goto cleanup;
}

// Set up the scope and page size search preferences.
arSearchPrefs [0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
arSearchPrefs [0].vValue.dwType = ADSTYPE_INTEGER;
arSearchPrefs [0].vValue.Integer = ulScope;

arSearchPrefs [1].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;
arSearchPrefs [1].vValue.dwType = ADSTYPE_INTEGER;
arSearchPrefs [1].vValue.Integer = 100;

hr = pSearch->SetSearchPreference(arSearchPrefs, 2);
if (FAILED(hr))
{
    wprintf(L"failed to set search prefs: 0x%x\n", hr);
    goto cleanup;
}

// The search filter specifies the objects to monitor
// and the USNChanged value to exceed.
swprintf_s(szSearchFilter,
L"(&(objectClass=user)(objectCategory=person)(uSNChanged>=%I64d))",
iLowerBoundUSN );

// Search for the objects indicated by the search filter.
hr = pSearch->ExecuteSearch(szSearchFilter,
                             pAttributeName, dwAttributes, &hSearch );
if (FAILED(hr))
{
    wprintf(L"failed to set execute search: 0x%x\n", hr);
    goto cleanup;
}

// Loop through the rows of the search result. Each row is an object
// with USNChanged greater than or equal to the specified value.
hr = pSearch->GetNextRow(hSearch);
while ( SUCCEEDED(hr) && hr != S_AMS_NOMORE_ROWS )
{
    ZeroMemory(&userdata, sizeof(MYUSERDATA));

    // Get the distinguishedName.
    hr = pSearch->GetColumn(hSearch, L"distinguishedName", &col);
    if ( SUCCEEDED(hr) )

```

```

        if ( SUCCEEDED(hr) )
    {
        if ( col.dwADSType == ADSTYPE_DN_STRING && col.pADSValues)
        {
            wcscpy_s(userdata.distinguishedName,
col.pADSValues->DNString);
        }

        pSearch->FreeColumn( &col );
    }

    // Get the telephone number.
    hr = pSearch->GetColumn( hSearch, L"telephoneNumber", &col );
    if ( SUCCEEDED(hr) )
    {
        if ( col.dwADSType == ADSTYPE_CASE_IGNORE_STRING &&
col.pADSValues)
        {
            wcscpy_s(userdata.phoneNumber, col.pADSValues->CaseIgnoreString);
        }

        pSearch->FreeColumn( &col );
    }

    // Get the objectGUID.
    hr = pSearch->GetColumn( hSearch, L"objectGUID", &col );
    if ( SUCCEEDED(hr) )
    {
        if ((col.dwADSType == ADSTYPE_OCTET_STRING) && col.pADSValues &&
            (col.pADSValues->OctetString.lpValue))
        {
            BuildGUIDString(szGUID, (LPBYTE) col.pADSValues->OctetString.lpValue);
            wcscpy_s(userdata.objectGUID, szGUID);
        }

        pSearch->FreeColumn( &col );
    }

    // Write the data from Active Directory to the secondary storage.
    WriteObjectDataToStorage(&userdata, bUpdate);
    dwCount++;
    hr = pSearch->GetNextRow( hSearch );
}

wprintf(L"dwCount: %d\n", dwCount);

// If this is a full synchronization, the operation is complete.
if (!bUpdate)
{
    goto cleanup;
}

// If it is an update, search for deleted objects.

// Release the search handle and pointer to reuse them.
wprintf(L"Searching for deleted objects\n");
if (hSearch)
{
    pSearch->CloseSearchHandle(hSearch);
    hSearch = NULL;
}
if (pSearch)
{
    pSearch->Release();
    pSearch = NULL;
}

// Bind to the Deleted Objects container.
hr = pRootDSE->Get(CComBSTR("defaultNamingContext"), &var);

```

```

if (FAILED(hr))
{
    wprintf(L"failed to get \\\"defaultNamingContext\\\"\n");
    goto cleanup;
}

LPWSTR pwszFilter = L"%s<WKGUID=%s,%s>";

// Reallocate the DS path buffer.
delete[] pszDSPath;
pszDSPath = new WCHAR[ wcslen(pwszFilter) +
                    wcslen(pszServerPath) +
                    wcslen(GUID_DELETED_OBJECTS_CONTAINER_W) +
                    wcslen(var.bstrVal) +
                    1];
if(!pszDSPath)
{
    wprintf(L"failed to allocate memory");
    hr = E_OUTOFMEMORY;
    goto cleanup;
}

swprintf_s(  pszDSPath, pwszFilter,
            pszServerPath,
            GUID_DELETED_OBJECTS_CONTAINER_W,
            var.bstrVal);
VariantClear(&var);

hr = ADsOpenObject(pszDSPath,
                   NULL,
                   NULL,
                   ADS_SECURE_AUTHENTICATION | ADS_FAST_BIND,
                   IID_IDirectorySearch,
                   (void**)&pSearch);
if (FAILED(hr))
{
    wprintf(L"failed to get IDirectorySearch: 0x%x\n", hr);
    goto cleanup;
}

// Specify the scope, pagesize, and tombstone search preferences.
arSearchPrefs[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
arSearchPrefs[0].vValue.dwType = ADSTYPE_INTEGER;
arSearchPrefs[0].vValue.Integer = ADS_SCOPE_SUBTREE;

arSearchPrefs[1].dwSearchPref = ADS_SEARCHPREF_PAGESIZE;
arSearchPrefs[1].vValue.dwType = ADSTYPE_INTEGER;
arSearchPrefs[1].vValue.Integer = 100;

arSearchPrefs[2].dwSearchPref = ADS_SEARCHPREF_TOMBSTONE;
arSearchPrefs[2].vValue.dwType = ADSTYPE_BOOLEAN;
arSearchPrefs[2].vValue.Boolean = TRUE;

hr = pSearch->SetSearchPreference(arSearchPrefs, 3);
if (FAILED(hr))
{
    wprintf(L"failed to set search prefs: 0x%x\n", hr);
    goto cleanup;
}

// Set up the search filter.
swprintf_s(szSearchFilter,
          L"(&(isDeleted=TRUE)(uSNChanged>=%I64d))",
          iLowerBoundUSN );

// Execute the search.
hr = pSearch->ExecuteSearch(szSearchFilter,
                             pAttributeName, dwAttributes, &hSearch );
if (FAILED(hr))

```

```

    {
        wprintf(L"failed to set execute search: 0x%08X\n", hr);
        goto cleanup;
    }
    wprintf(L"Started search for deleted objects.\n");

    // Loop through the rows of the search result.
    // Each row is an object deleted since the previous call.
    dwCount = 0;
    hr = pSearch->GetNextRow(hSearch);
    while ( SUCCEEDED(hr) && hr != S_ADS_NOMORE_ROWS )
    {
        ZeroMemory(&userdata, sizeof(MYUSERDATA) );

        // Get the distinguishedName.
        hr = pSearch->GetColumn(hSearch, L"distinguishedName", &col);
        if ( SUCCEEDED(hr) )
        {
            if (col.dwADsType == ADSTYPE_DN_STRING && col.pADsValues)
            {
                wcscpy_s(userdata.distinguishedName,
                col.pADsValues->DNString);
            }

            pSearch->FreeColumn( &col );
        }

        // Get the objectGUID number.
        hr = pSearch->GetColumn( hSearch, L"objectGUID", &col );
        if ( SUCCEEDED(hr) )
        {
            if ((col.dwADsType == ADSTYPE_OCTET_STRING) && col.pADsValues &&
                (col.pADsValues->OctetString.lpValue))
            {
                BuildGUIDString(szGUID, (LPBYTE) col.pADsValues->OctetString.lpValue);
                wcscpy_s(userdata.objectGUID, szGUID);
            }

            pSearch->FreeColumn( &col );
        }

        // If the objectGUID of a deleted object matches an objectGUID in
        // the secondary storage, delete the object from storage.
        DeleteObjectDataFromStorage(&userdata);
        dwCount++;
        hr = pSearch->GetNextRow(hSearch);
    }

    wprintf(L"deleted dwCount: %d\n", dwCount);

cleanup:

    if (pszServerPath)
    {
        delete[] pszServerPath;
        pszServerPath = NULL;
    }
    if (pszDSPath)
    {
        delete[] pszDSPath;
        pszDSPath = NULL;
    }
    if (pRootDSE)
    {
        pRootDSE->Release();
        pRootDSE = NULL;
    }
    if (pDCService)
    {

```

```

        pDCService->Release();
        pDCService = NULL;
    }
    if (pDeletedObj)
    {
        pDeletedObj->Release();
        pDeletedObj = NULL;
    }
    if (hSearch)
    {
        pSearch->CloseSearchHandle(hSearch);
        hSearch = NULL;
    }
    if (pSearch)
    {
        pSearch->Release();
        pSearch = NULL;
    }
    VariantClear(&var);

    return hr;
}

//*****
// DeleteObjectDataFromStorage routine
//*****
VOID DeleteObjectDataFromStorage(PMYUSERDATA pUserData) {
    wprintf(L"DELETED OBJECT:\n");
    wprintf(L"    objectGUID: %s\n", pUserData->objectGUID);
    wprintf(L"    distinguishedName: %s\n", pUserData->distinguishedName);
    wprintf(L"-----\n");

    return;
}

//*****
// WriteObjectDataToStorage routine
//*****
VOID WriteObjectDataToStorage(PMYUSERDATA pUserData, BOOL bUpdate) {
    if (bUpdate)
    {
        wprintf(L"UPDATE:\n");
    }
    else
    {
        wprintf(L"INITIAL DATA:\n");
    }

    wprintf(L"    objectGUID: %s\n", pUserData->objectGUID);
    wprintf(L"    distinguishedName: %s\n", pUserData->distinguishedName);
    wprintf(L"    phoneNumber: %s\n", pUserData->phoneNumber);
    wprintf(L"-----\n");

    return;
}

//*****
// WriteSyncParamsToStorage routine
// This example caches the parameters in the registry. In a real // synchronization application, store the
// parameters in the same // storage that you are keeping consistent with Active Directory.
// This ensures that the parameters and object data remain synchronized // if the storage is ever restored
// from a backup.
//*****
DWORD WriteSyncParamsToStorage(
    LPWSTR pszPrevInvocationID, // Receives invocation ID
    LPWSTR pszPrevHighUSN,     // Receives previous high USN
    LPWSTR pszDCName)         // Receives name of DC to bind to
{
    HKEY hReg = NULL;
}

```

```

DWORD dwStat = NO_ERROR;

// Create a registry key under
// HKEY_CURRENT_USER\SOFTWARE\Vendor\Product.
dwStat = RegCreateKeyExW(HKEY_CURRENT_USER,
    L"Software\\Microsoft\\Windows 2000 AD-Synchro-USN",
    0,
    NULL,
    REG_OPTION_NON_VOLATILE,
    KEY_WRITE,
    NULL,
    &hReg,
    NULL);
if (NO_ERROR == dwStat)
{
    // Cache the invocationID as a value under the registry key.
    dwStat = RegSetValueExW(hReg,
        L"InvocationID",
        0,
        REG_SZ,
        (LPBYTE)pszPrevInvocationID,
        (lstrlenW(pszPrevInvocationID) + 1) * sizeof(WCHAR));

    // Cache the previous high USN as a value under the registry key.
    dwStat = RegSetValueExW(hReg,
        L"PreviousHighUSN",
        0,
        REG_SZ,
        (LPBYTE)pszPrevHighUSN,
        (lstrlenW(pszPrevHighUSN) + 1) * sizeof(WCHAR));

    // Cache the DC name as a value under the registry key.
    dwStat = RegSetValueExW(hReg,
        L"DC name",
        0,
        REG_SZ,
        (LPBYTE)pszDCName,
        (lstrlenW(pszDCName) + 1) * sizeof(WCHAR));

    RegCloseKey(hReg);
}

return dwStat;
}

//*****
// GetSyncParamsFromStorage routine
// This example reads the parameters from the registry. In a real // synchronization application, store the
parameters in the // same storage that you are keeping consistent with Active Directory.
//*****


DWORD GetSyncParamsFromStorage(
    LPWSTR pszPrevInvocationID, // Receives invocation ID
    DWORD dwPrevInvocationSize, // size of the buffer, in WCHARS
    LPWSTR pszPreviousHighUSN, // Receives previous high USN
    DWORD dwPrevHighUSNSize, // size of the buffer, in WCHARS
    LPWSTR pszDCName, // Receives name of DC to bind to
    DWORD dwDCNameSize) // size of the buffer, in WCHARS
{
    HKEY hReg = NULL;
    DWORD dwStat;

    // Open the registry key.
    dwStat = RegOpenKeyExW(
        HKEY_CURRENT_USER,
        L"Software\\Microsoft\\Windows 2000 AD-Synchro-USN",
        0,
        KEY_QUERY_VALUE,
        &hReg);
    if (NO_ERROR == dwStat)

```

```

    {
        // Get the previous invocationID from the registry.
        dwPrevInvocationSize = dwPrevInvocationSize * sizeof(WCHAR);
        dwStat = RegQueryValueExW(hReg,
            L"InvocationID",
            NULL,
            NULL,
            (LPBYTE)pszPrevInvocationID,
            &dwPrevInvocationSize);
        if (dwStat != NO_ERROR)
        {
            wprintf(L"RegQueryValueEx failed to get invocationID: 0x%x\n", dwStat);
        }

        // Get the previous high USN from the registry.
        dwPrevHighUSNSize = dwPrevHighUSNSize * sizeof(WCHAR);
        dwStat = RegQueryValueExW(hReg,
            L"PreviousHighUSN",
            NULL,
            NULL,
            (LPBYTE)pszPreviousHighUSN,
            &dwPrevHighUSNSize);
        if (dwStat != NO_ERROR)
        {
            wprintf(L"RegQueryValueEx failed to get previous high USN:
0x%x\n", dwStat);
        }

        // Get the DC name from the registry.
        dwDCNameSize = dwDCNameSize * sizeof(WCHAR);
        dwStat = RegQueryValueExW(hReg,
            L"DC name",
            NULL,
            NULL,
            (LPBYTE)pszDCName,
            &dwDCNameSize);
        if (dwStat != NO_ERROR)
        {
            wprintf(L"RegQueryValueEx failed to get DC name: 0x%x\n", dwStat);
        }

        RegCloseKey(hReg);
    }

    return dwStat;
}

//*****
// BuildGUIDString
// Routine that makes the GUID a string in directory service bind form.
//*****
VOID BuildGUIDString(WCHAR *szGUID, LPBYTE pGUID) {
    DWORD i;
    DWORD dwlen = sizeof(GUID);
    WCHAR buf[4];

    wcscpy_s(szGUID, L"");
    for(i = 0; i < dwlen; i++)
    {
        swprintf_s(buf, L"%02x", pGUID[i]);
        wcscat_s(szGUID, buf);
    }
}

//*****
// Main
//*****
int main(int argc, char* argv[])
{

```

```

    DWORD dwStat;
    HRESULT hr;

    // Attributes to retrieve
    LPWSTR szAttrs[] =
    {
        {L"telephoneNumber"},
        {L"distinguishedName"},
        {L"uSNChanged"},
        {L"objectGUID"},
        {L"isDeleted"}
    };
    LPWSTR *pszAttrs=szAttrs;
    DWORD dwAttrs = sizeof(szAttrs)/sizeof(LPWSTR);

    // DC properties to cache for next synchronization.
    WCHAR szPrevInvocationID[40];
    WCHAR szPrevHighUSN[40];
    WCHAR szDCName[MAX_PATH];

    CoInitialize(NULL);

    if (argc > 1)
    {
        // Perform a full synchronization.
        // Initialize synchronization parameters to empty strings.
        wprintf(L"Performing a full read.\n");
        szPrevInvocationID[0] = '\0';
        szPrevHighUSN[0] = '\0';
        szDCName[0] = '\0';
    }
    else
    {
        // Perform a synchronization update.
        // Initialize synchronization parameters from storage.
        wprintf(L"Retrieving changes only.\n");
        dwStat = GetSyncParamsFromStorage(
            szPrevInvocationID,
            ARRAYSIZE(szPrevInvocationID),
            szPrevHighUSN,
            ARRAYSIZE(szPrevHighUSN),
            szDCName,
            ARRAYSIZE(szDCName));
        if (dwStat != NO_ERROR)
        {
            wprintf(L"Could not get synchronization parameters: %u\n", dwStat);
            goto cleanup;
        }
    }

    // Perform the search and update the synchronization parameters.
    hr = DoUSNSyncSearch(L"OU=UK,DC=xapac,DC=xnet,DC=intra",
        ADS_SCOPE_SUBTREE,
        pszAttrs, dwAttrs,
        szPrevInvocationID, szPrevHighUSN, szDCName);

    if (FAILED(hr))
    {
        wprintf(L"DoUSNSyncSearch failed: 0x%x\n", hr);
        goto cleanup;
    }

    // Cache the synchronization parameters in storage for the next synchronization.
    wprintf(L"Caching the synchronization parameters.\n");
    dwStat = WriteSyncParamsToStorage(
        szPrevInvocationID, szPrevHighUSN, szDCName);
    if (dwStat != NO_ERROR)
    {
        wprintf(L"Error caching the synchronization parameters: %u\n", dwStat);
        goto cleanup;
    }
}

```

```
        goto cleanup;
    }

cleanup:

    CoUninitialize();

    return 1;
}
```

Service Publication

6/3/2022 • 2 minutes to read • [Edit Online](#)

Services advertise themselves using objects stored in Active Directory Domain Services. This is known as *service publication*. Clients query the directory to locate services. This is called *client-service rendezvous*. This section discusses the types of directory objects used for service publication and explains how they are used for client-service rendezvous.

This section discusses the following topics:

- [An overview of service publication](#)
- [Security issues for service publication](#)
- [Connection point objects](#)
- [Publishing with service connection points \(SCPs\)](#)
- [What information to store in a service connection point](#)
- [Where to create a service connection point](#)
- [How to publish replicable, host-based, and database services using service connection points](#)
- [Creating and maintaining a service connection point](#)
- [How a client queries for an SCP and uses it to bind to a service instance](#)
- [Using the RPC name service \(RpcNs\) APIs to publish an RPC service](#)
- [Windows Sockets registration and resolution \(RnR\) to publish a Windows Sockets service](#)
- [Publication of COM-based services in the COM+ class store](#)

For more information about how services and clients authenticate each other, see [Mutual Authentication Using Kerberos](#). For more information about service security contexts and logon accounts, see [Service Logon Accounts](#).

About Service Publication

6/3/2022 • 2 minutes to read • [Edit Online](#)

A service is an application that makes data or operations available to network clients. Often, a service is implemented as a formal Microsoft Win32-based service, but this is not required.

Service publication is the act of creating and maintaining data about one or more instances of a given service so that network clients can find and use the service. Publishing a service in Active Directory Domain Services enables clients and administrators to move from a computer-centric view of the distributed system to a service-centric view.

Microsoft Windows NT 3.51 and later operating systems: A distributed system was a group of computers running various services. To access a service, an application required data about which computers offered the service.

Windows 2000 Server, Windows 2000 Advanced Server and Windows 2000 Datacenter Server: Services publish their existence using Active Directory Domain Services objects. The objects contain binding information that client applications use to connect to instances of the service. To access a service, a client does not need to know about specific computers: the objects in an Active Directory server include this information. A client queries the Active Directory server for an object representing a service (called a connection point object) and uses the binding data from the object to connect to the service.

The following table shows examples of bindings.

SERVICE	BINDING
File Service	UNC Name for a share. For example "\\MyServer\MyshareName".
Web Service	URL. For example "https://www.fabrikam.com".
RPC Service	Remote procedure call (RPC) binding: special encoded information used to connect to the RPC server. RPC bindings can be converted to and from strings with the RPC APIs. For example: "ncacn_ip_tcp:server.fabrikam.com".

In a distributed system, the computers are engines, and the interesting entities are the services that are available. From the user perspective, the identity of the computer that provides a particular service is not important. What is important is accessing the service itself.

This is also the case with service management. The administrator of a given DNS zone is not interested in the computers running the DNS service; the administrator wants to administer DNS. There will likely be multiple instances of the DNS service, one of which is authoritative. The computers that support DNS service are not important to the DNS administrator. What is important is how to manage the service as a single distributed resource—not as individual processes running on different computers.

Security Issues for Service Publication

6/3/2022 • 2 minutes to read • [Edit Online](#)

The system restricts the ability to create, modify, or delete connection point objects. Be aware of, and handle, these restrictions when you publish a service.

Clients must be able to trust the data published in a connection point object in the directory. For this reason, permission to create a connection point object is typically restricted to privileged users, such as domain administrators. This prevents unauthorized users from deceiving clients by creating invalid connection points for well-known services.

Services must not run with domain administrator privileges. This means that a service typically cannot create its own connection point. Instead, you provide a service installation or configuration application that creates the connection point. This installer must be run by a user with the necessary privileges.

Although a service cannot typically create its connection point, it must be able to update the connection point properties at run time. The connection point properties contain the binding data used by clients to connect to the service. If the binding data changes, the service must update the connection point; otherwise, clients cannot use the service. This means that the installer must also modify the security descriptor on the connection point object to enable the service to read and write the appropriate properties at run time. For more information and a code example, see [Enabling Service Account to Access SCP Properties](#).

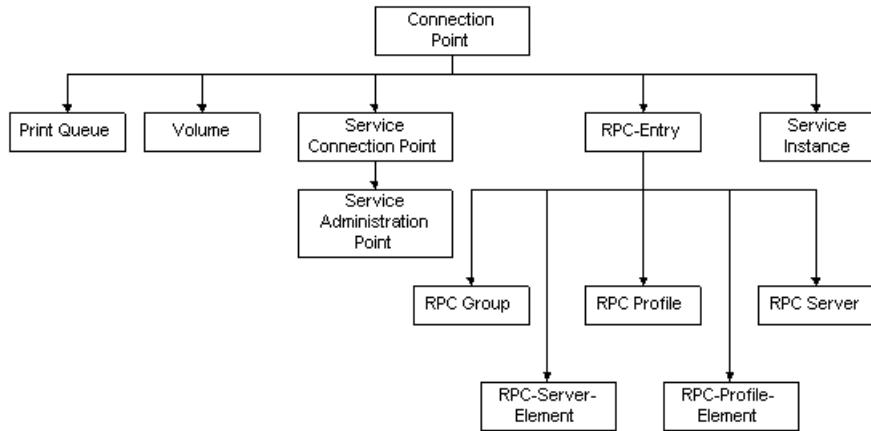
A service running under the LocalSystem account can create a connection point as a child object under its own computer object in the directory. Such a service is an exception to the rule of services not creating their own connection points. A LocalSystem service also has permission to modify the properties of connection point objects under its own computer object. Be aware that a service should run under the LocalSystem account only if absolutely necessary. For more information, see [Guidelines for Selecting a Service Logon Account](#).

The application that creates a connection point object, or any object, must have create child permissions for the object class to be created in the container where the object will be created. To remove an object, the process performing the operation must have delete child permissions for the object class to be deleted on the container holding the object, or have delete permissions on the object itself. To update a connection point the process performing the operation must have write-access to the properties to be updated on the object.

Connection Points

6/3/2022 • 2 minutes to read • [Edit Online](#)

A connection point object contains data about one or more instances of a service available on the network. The **connectionPoint** object class is the abstract base class from which objects representing connectable resources in Active Directory Domain Services are derived. The following illustration shows some of the object classes derived from the **connectionPoint** object class.



The following table lists immediate subclasses of the **connectionPoint** class.

OBJECT CLASS	DESCRIPTION
serviceConnectionPoint	Service connection point (SCP) objects for publishing data that client applications use to bind to a service. For more information, see Publishing with Service Connection Points (SCPs) .
rpcEntry	An abstract class whose subclasses are used by the RPC Name Service (Ns) accessed through the RpcNs* functions in the Win32 API. For more information, see Publishing with the RPC Name Service (RpcNs) .
serviceInstance	Connection point object used by the Windows Sockets Registration and Resolution (RnR) name service, accessed through the Windows Sockets WSA* APIs. For more information, see Publishing with Windows Sockets Registration and Resolution (RnR) .
printQueue	Connection point object used to publish network printers. For more information, see IADsPrintQueue .
volume	Connection point object used to publish file services.

Be aware that COM-based services do not use connection-point objects to advertise themselves. These services are published in the class store. The Windows 2000 class store is a directory-based repository for all COM-based applications, interfaces, and APIs that provide for application publishing and assigning. For more information, see [Publishing COM+ Services](#).

Publishing with Service Connection Points

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory Schema defines a **serviceConnectionPoint** (SCP) object class to make it easy for a service to publish service-specific data in the directory. Clients of the service use the data in an SCP to locate, connect to, and authenticate an instance of your service.

This section provides an overview of service connection points and code examples that show how a client/service application uses SCPs.

The code example follows these steps to implement service publication with SCPs.

For more information and a code example that performs these steps, see [Creating a Service Connection Point](#).

To create SCPs in the directory at service installation

1. Bind to the computer object for the host computer on which the service instance is installed.
2. Create an SCP object as a child of the computer object, specifying the initial values for the attributes of the SCP.
3. Set access control entries (ACEs) in the security descriptor of the SCP object to enable the service to modify SCP properties at run time.
4. Cache the **objectGUID** of the SCP in the registry on the service's host computer.

For more information and a code example that performs these steps, see [Updating a Service Connection Point](#).

To update the SCP attributes at service startup

1. Retrieve the **objectGUID** from the registry and use it to bind to the SCP.
2. Retrieve attributes, such as **serviceDNSName** and **serviceBindingInformation**, from the SCP. Compare these values to the current values and update the SCP if necessary.

For more information and a code example that performs these steps, see [How Clients Find and Use a Service Connection Point](#).

To find and use an SCP by a client application

1. Bind to the Global Catalog and search for objects with a **keywords** attribute that matches the service's product GUID. Each object found is an instance of the service. Select an instance and retrieve the distinguished name of the SCP.
2. Use the distinguished name to bind to the SCP.
3. Retrieve the values of various attributes from the SCP, such as **serviceDNSName** and **serviceBindingInformation**. Use these values to connect to and authenticate the service instance.

For more information about what roles can create and update an SCP, see [Security Issues for Service Publication](#).

For more information about where to create an SCP, see [Where to Create a Service Connection Point](#).

For more information about the kind of data to store in an SCP, see [Service Connection Point Properties](#).

For more information about how a service installer and the service work together to maintain current data in an SCP, see [Creating and Maintaining a Service Connection Point](#).

Where to Create a Service Connection Point

6/3/2022 • 3 minutes to read • [Edit Online](#)

When an instance of Active Directory Domain Services is installed, the service installer creates service connection point objects (SCP) in Active Directory Domain Services. The primary objective should be to minimize replication traffic and to enable efficient administration and maintenance of objects.

Be aware that client applications find SCPs by searching the directory for keywords in the SCP. The **keywords** attribute of an SCP is included in the Global Catalog; clients can search the Global Catalog to find SCPs in the forest. For this reason, the client does not influence where to publish SCPs.

Minimize Replication Traffic

To minimize replication traffic, create SCPs in the domain partition of the domain of the service host computer. For example, you can create SCPs as child objects of the computer object on which the service is installed. A domain partition of Active Directory Domain Services, sometimes called a domain naming context, contains domain-specific objects such as the objects for the users and computers of the domain. A full replica of all objects in the domain partition is replicated to every domain controller (DC) for the domain, but it is not replicated to DCs of other domains.

Do not create SCPs in the Configuration partition, also known as the configuration naming context, because changes to the Configuration partition are replicated to every DC in the forest. As noted above, clients throughout the forest can query the Global Catalog to find SCPs anywhere in the forest, so creating SCPs in the Configuration partition does not make them more visible to clients; it only generates more replication traffic.

Ease of Administration

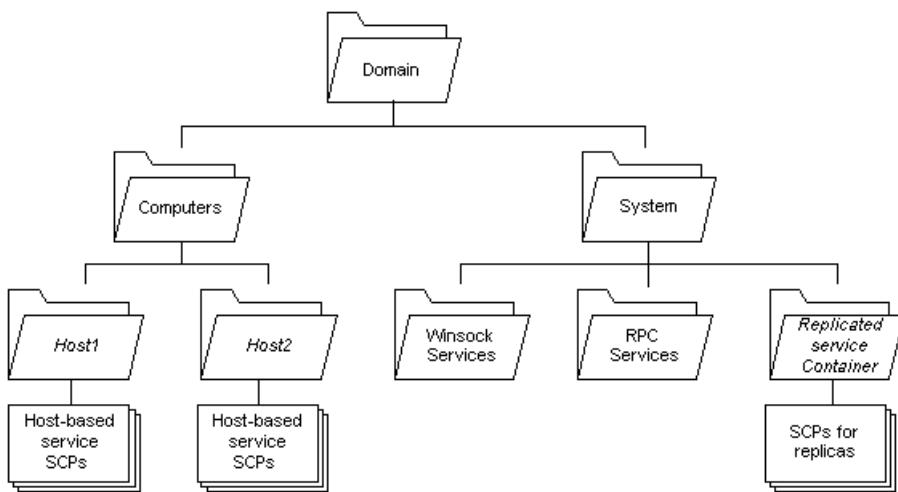
Consider the following guidelines for object administration:

- Place service-specific objects where administrators can control access to them using policy and inherited access permissions.
- Place the objects in where an administrator can easily find them.

A good default location that satisfies both goals is to create SCP and other service-specific objects under the computer object of the host computer of each service instance. For more information, see [Publishing Under a Computer Object](#).

A good alternative for services not tied to a single host is to create a container for the service objects under the System container in a domain partition. For more information, see [Publishing in a Domain System Container](#).

The following diagram shows part of the default container hierarchy for a domain partition.



The diagram shows the default domain hierarchy included with Active Directory Domain Services. However, many enterprises create a hierarchy of organizational unit (OU) containers to group object classes, such as users and computers, together for purposes of administration. Administrators can then apply policy and inheritable access-control entries (ACEs) to an OU to delegate administrative authority for objects in the OU. This enables administrators to efficiently manage an enterprise, but it has a few consequences for service programmers:

- The computer object for a service host might not be under the Computers container as shown in the diagram. For more information about how to find the computer object for the local computer, see [Publishing Under a Computer Object](#).
- Administrators may move objects as their organizational needs change. This means that you cannot depend on your objects remaining in a fixed location; that is, your service cannot depend on an object distinguished name remaining the same. Instead, use an object's **objectGUID** attribute, which does not change if the object is moved or renamed. For more information, and a code example that creates an SCP, stores its **objectGUID**, and later retrieves the **objectGUID** to bind to the SCP, see [Creating and Maintaining a Service Connection Point](#).
- All of the standard service-related object classes, as well as any subclasses of these classes, are valid children of the **computer** and **organizationalUnit** classes. If you extend the schema to define your own service-specific class, be sure that the **computer** and **organizationalUnit** classes are included in the possible superiors.
- The service installer determines the default location for creating SCPs. You may want to allow the administrator installing the service to specify an alternate install path.

Service-specific objects should not be created in the following areas:

- Services should not publish objects directly in the Users or Computers containers of a domain partition, nor should they create new containers in these containers. However, services can publish objects as child objects of a computer object, whether or not the computer object is stored in the Computers container.
- Services, that use Windows Sockets registration and resolution (RnR) or the RPC name service (RpcNs) APIs to advertise themselves, create the proper objects in the WinsockServices and RpcServices containers under a domain partition's System container. Do not explicitly create objects in these containers. Doing so does not cause direct harm, but may be confusing for administrators.

Publishing Under a Computer Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

Typically, host-based services create SCPs under the computer object for the host computer. Host-based services are services closely tied to a single host computer.

To create SCPs under a computer object

1. Call the [GetComputerObjectName](#) function to get the distinguished name (DN) in the directory of the computer object for the local computer.
2. Use that DN to bind to the computer object and create the SCP.

For more information and a code example, see [How Clients Find and Use a Service Connection Point](#).

Be aware that only domain-member computers have valid computer objects in the directory.

To get the DNS or NetBIOS name of the local computer, call the [GetComputerNameEx](#) function.

Publishing in a Domain System Container

6/3/2022 • 2 minutes to read • [Edit Online](#)

The System container of a domain partition holds per-domain operational information. This includes the default local security policy, file link tracking, network meetings, and containers for Windows Sockets registration and resolution (RnR) and RPC name service (RpcNs) connection points. The system container is hidden, by default, and provides a convenient place for storing objects that are of interest to administrators, but not to end users.

Services that are not tied to a single host may want to create their SCPs under the System container of a domain partition. This alternative can be useful for services with replicas installed on multiple hosts, each replica providing identical services to clients throughout the domain. It enables you to group all the objects for the replicated service under a single container.

Services that create service-specific objects in the System container must do the following:

1. Create a sub-container of object class **container** as an immediate child of the System container. Give this sub-container a name that clearly identifies it as pertaining to the service.
2. Create the service-related objects in this sub-container. For example, NetMeeting uses the Meetings container to publish network meeting objects.

A vendor with multiple products can use a similar strategy to group service-related objects for all of its products. In this case, you could create a **container** object with a name that clearly identifies the vendor; then create **container** objects for each service as children of the vendor container. Create the vendor-specific container as a child of the System container.

Service Connection Points for Replicated, Host-based, and Database Services

6/3/2022 • 3 minutes to read • [Edit Online](#)

When you publish a service using a service connection point (SCP), consider how clients will locate the SCP for your service. If multiple instances of the service exist, consider how clients will distinguish the service with the desired features from similar services with different features. If you publish a replicated service, consider how a client will choose a replica. This topic discusses these issues for various types of services.

Replicable services

For a replicable service there can be one or many instances, or replicas, of the service, and clients do not care which replica they connect to because each provides the same service. Active Directory Domain Services are examples of replicated services: all domain controllers for a given domain hold identical data, subject to replication latency, and provide identical services.

Replicable services can store the SCPs and other service-specific objects for multiple replicas in a single container. The setup application for the first replica can create the container as a child of the local domain's System container. For more information, see [Publishing in a Domain System Container](#). Ensure that the security descriptor for your container allows the setup programs for subsequent replicas to create their objects in the same container. Grant permissions for the installing administrator to specify the users or groups that can create or modify objects in the container.

One strategy for a replicable service is to create an SCP for each replica. When a client queries for the service's product GUID or other identifying keyword, it finds the SCP objects for all replicas and selects one at random or using some load-balancing algorithm. For example, an administrator could specify priority and load-balancing data for each replica, similar to the priority and weight fields of a DNS SRV record. The service's setup application can store this data in the **serviceBindingInformation** attribute of each replica's SCP. Clients retrieve the data from each SCP and use it to select a replica.

Another strategy is to create a single SCP for all replicas and set the SCP **serviceDNSName** attribute to the name of a DNS SRV record. Then the setup application for each replica registers a SRV record with that name. When a client identifies the service's lone SCP, the client retrieves the name of the SRV record and uses the **DnsQuery** function to retrieve the array of SRV records for the replicas. Each SRV record contains the name of a host computer and additional data that the client can use to select a replica.

Database services

Different instances of a database service may contain entirely different data, even though they are all the same kind of service, usually called service class. To publish this kind of service, the **keywords** attribute of the SCP can identify both the service class and the specific database. A general-purpose client that knows only the GUID of the service class can query for all databases published by that service class, and then present a user interface to allow the user to select one. For a client that is designed specifically for the target database, you can hard-code the database GUID into the client code.

Host-based services

Host-based services are services that are closely tied to a single host computer. You can install instances of the service class on many computers and each instance provides services that are identified with its host computer.

Each instance of a host-based service should create its own SCP under the computer object of its host. Clients who use a product GUID to search for the SCP of a host-based service typically find many instances of the service class throughout the enterprise forest. Clients can then use the **serviceDNSName** attribute of the SCPs to find the SCP for the service instance on the desired host computer.

Service Connection Point Properties

6/3/2022 • 3 minutes to read • [Edit Online](#)

The attributes of the **serviceConnectionPoint** class are sufficient for most services. Active Directory Domain Services do not define how the attributes are used, so the clients of your service must be able to interpret and use the data in your service SCPs. Services that must publish additional data about themselves can extend the Active Directory schema by creating a subclass of the **serviceConnectionPoint** class, giving the subclass a distinct name. For more information about schema extensions, see [Extending the Schema](#).

The most important attributes of an SCP are **keywords**, **serviceDNSName**, **serviceDNSNameType**, **serviceClassName**, and **serviceBindingInformation**. Client applications search the directory for **keywords** values to locate your SCP. When your SCP is found, clients can read other attributes to retrieve service data.

ATTRIBUTE	DESCRIPTION
keywords	<p>The keywords attribute can contain multiple string values that identify your service. This attribute is included in the Global Catalog, which means that clients in any domain of an enterprise forest can search the Global Catalog for keywords associated with your service. This attribute is also indexed, which improves query performance. The installer that creates the SCP sets the values of the keywords attribute. Typically, these values are not modified by the active service.</p> <p>The exact keywords you should include in your SCP depend on how clients search for your service. The best keywords to use are GUID strings because GUIDs are guaranteed to be unique in a forest. Use the GUID string format returned by the UuidToString function in the RPC library. You can also include human-readable names, if clients may use them to search for your service. The keywords in an SCP should include GUID strings and/or names that identify the following data about your service:</p> <ul style="list-style-type: none">• Your company or organization: for example, Fabrikam.• The product or service: for example, SQL Server. This enables client applications to find SCPs for services of that type.• The specific version of the product or service, such as 7.5.• For SCPs that publish a specific set of data or capabilities for a type of service, include a GUID string or name that identifies the specific instance. For example, a database service could publish an SCP for a specific database. In this case, the SCP would include a product GUID to identify the service and another GUID to identify the database.

ATTRIBUTE	DESCRIPTION
serviceDNSName and serviceDNSNameType	<p>Client applications use the serviceDNSName and serviceDNSNameType attributes to determine the service's host computer. The serviceDNSNameType value indicates the type of DNS name specified by serviceDNSName usually "A" if serviceDNSName contains a host name or "SRV" if serviceDNSName contains a SRV record name.</p> <p>The serviceDNSName value is typically the DNS name of the service's host computer. Your service installer can call the GetComputerNameEx function to get the DNS name of the local computer.</p> <p>For services that have DNS SRV records, serviceDNSName can be the name of the SRV record. A client application uses the DNS APIs to retrieve all the SRV records that match this name. The client then retrieves the DNS host name from one of the SRV records. This technique is useful for replicated services because SRV records also include data that enables the client to select the best replica.</p>
serviceBindingInformation	<p>A multi-value property that contains string values that store data required to bind to a service. This property is indexed and is replicated to the Global Catalog.</p> <p>The content of serviceBindingInformation is specific to the service that published the SCP; clients must interpret the binding data. In the most common case, the binding data consists of a port number on the service host computer.</p>
serviceClassName	<p>A single-value property that identifies the class of service represented by the SCP. This is a descriptive string specific to the service that published the SCP; for example, SqlServer.</p> <p>For services that support mutual authentication, clients can use this property, along with the DNS name of the service's host computer, to form a service principal name. For more information, see Mutual Authentication Using Kerberos.</p>

Creating and Maintaining a Service Connection Point

6/3/2022 • 2 minutes to read • [Edit Online](#)

When publishing with an SCP, remember that it must contain current data about the service instance. Otherwise, clients that bind to the SCP retrieve outdated data. Your service installer, that creates an SCP, specifies the initial values for the SCPs attributes. Then, when the service instance starts, it must locate the SCP and update the attribute values, if necessary. In this way, clients are assured the most current data.

After creating the SCP, your service installer performs two additional steps that enable your service to update the SCP:

- Set ACEs in the security descriptor of the SCP object to enable the service to modify SCP attributes at run time. For more information and a code example, see [Enabling Service Account to Access SCP Properties](#).
- Cache the **objectGUID** of the SCP in the registry on the service host computer. The service retrieves the cached GUID to bind to the SCP to verify and update its attributes.

The service installer caches the SCP's **objectGUID** rather than its DN. The **objectGUID** never changes, regardless of whether the SCP is moved or renamed. The DN can change if an administrator moves or renames the SCP. For example, if you create an SCP as a child of a computer object, the distinguished name of the SCP changes if the computer is renamed or moved to a different domain or organizational unit.

When a service installer creates an SCP, it must read the **objectGUID** of the newly created object and cache it in the registry of the service host computer. Use the [**IADs::get_GUID**](#) method to get the **objectGUID** value in string format suitable for binding. Cache the GUID string as a value under the following registry key.

```
HKEY_LOCAL_MACHINE  
  vendor name  
    product name
```

Where "vendor name" and "product name" identify the vendor and product.

When the service starts, it retrieves the cached GUID string from the registry and uses it to bind to the SCP. The service reads the important SCP attributes and compares them to current values. If the SCP values are outdated, the service updates them. Values that the service might require to update include **keywords**, **serviceBindingInformation**, **serviceDNSName**, and **serviceDNSNameType**.

Examples

- [Script samples](#)
- [Code \(C/C++\) samples](#)

Script Samples for Managing Service Connection Points

6/3/2022 • 2 minutes to read • [Edit Online](#)

Service connection points can be managed using Microsoft Windows scripting interfaces, as the sample code in this section demonstrates.

Examples:

- [Excel Macros to Create or Delete a Service Connection Point](#)
- [VB Script to display all Service Connection Points](#)

Excel Macros for Managing Service Connection Points

6/3/2022 • 2 minutes to read • [Edit Online](#)

Service Connection Points can be managed using simple Excel Macros.

The following Excel Macro shows the minimum requirements for creating a new Service Connection Point.

```
Option Explicit

Private Sub p_CreateExampleSCP()

    Dim oAdSysInfo As ADSysInfo
    Set oAdSysInfo = CreateObject("ADSystemInfo")

    Dim objComputer As ActiveDs.IADsContainer
    Set objComputer = GetObject("LDAP://" + oAdSysInfo.ComputerName)

    Dim IADsSCP As ActiveDs.IADs
    Set IADsSCP = objComputer.Create("ServiceConnectionPoint", _
        "CN=Example SCP")

    IADsSCP.PutEx 2, "serviceBindingInformation", _
        Array( _
            " - UNC connection to Service", _
            " - WS-Man connection to service" _
        )

    IADsSCP.PutEx 2, "Keywords", _
        Array( _
            "KW1=A keywrowd value" _
        )

    IADsSCP.SetInfo

End Sub
```

The following Excel Macro shows how to delete the sample Service Connection Point.

```
Option Explicit

Private Sub p_DeleteExampleScp()
    Dim oAdSysInfo As ADSysInfo
    Set oAdSysInfo = CreateObject("ADSystemInfo")

    Dim objComputer As ActiveDs.IADsContainer
    Set objComputer = GetObject("LDAP://" + oAdSysInfo.ComputerName)

    objComputer.Delete "ServiceConnectionPoint", _
        "CN=Example SCP"

End Sub
```

VB Script Listing all Service Connection Points

6/3/2022 • 2 minutes to read • [Edit Online](#)

Service Connection Points can be managed using VB Script. The following sample code shows how to list all Service Connection Points on the domain name provided on the command line:

```
Option Explicit

Dim scpDN
scpDN = WScript.Arguments.Named("SCP")

If ( scpDN = "" ) Then
    Usage
Else
    ScpProperties scpDN
End If

' End of Main

Sub Usage
    WScript.Echo "USAGE: ScpProperties /SCP:<SCP DN>"
    WScript.Echo ""
    Wscript.Echo "Remember to put the SCP DN between quotes"
End Sub

Sub ScpProperties( strDN )

    Dim oConn ' As ADODB.Connection
    Set oConn = CreateObject("ADODB.Connection")

    oConn.Open "Provider=ADsDSOOObject"

    Dim oCmd ' As ADODB.Command
    Set oCmd = CreateObject("ADODB.Command")
    oCmd.ActiveConnection = oConn

    ' In the next command, the real magic is in knowing LDAP queries
    ' GC means to ask the global catalog.
    oCmd.CommandText = "<GC://" & strDN & ">;" & _
                      "(objectClass=ServiceConnectionPoint);;" & _
                      "distinguishedname,name,serviceDNSName," & _
                      "serviceDNSNameType,serviceBindingInformation," & _
                      "serviceClassName,Keywords;" & _
                      "subtree"

    Dim oRs ' As ADODB.Recordset
    Set oRs = oCmd.Execute

    Dim lCount ' As Long
    lCount = 0
    Do While (Not oRs.EOF)

        Wscript.Echo "SCP Name: " & oRs("name")
        Wscript.Echo "DN: " & oRs("distinguishedname")
        Wscript.Echo "serviceDNSName: " & oRs("serviceDNSName")
        Wscript.Echo "serviceDNSNameType: " & oRs("serviceDNSNameType")
        Wscript.Echo "serviceClassName: " & oRs("serviceClassName")
        Dim a ' As Variant
        Dim l 'As Long

        a = oRs("serviceBindingInformation")
```

```
If (Not IsNull(a)) Then
    Wscript.Echo "serviceBindingInformation:"
    For l = LBound(a) To UBound(a)
        Wscript.Echo vbTab & a(l)
    Next
    Wscript.Echo
End If

a = oRs("Keywords")
If (Not IsNull(a)) Then
    Wscript.Echo "Keywords: "
    For l = LBound(a) To UBound(a)
        Wscript.Echo vbTab & a(l)
    Next
End If

oRs.MoveNext
lCount = lCount + 1
Wscript.Echo
Loop

Wscript.Echo "Total objects = " & lCount
oConn.Close

End Sub
```

Code Samples for Managing Service Connection Points

6/3/2022 • 2 minutes to read • [Edit Online](#)

Service Connection Points are managed using Active Directory Service Interfaces. The samples below demonstrate this interface.

Examples:

- [Create a Service Connection Point](#)
- [Update a Service Connection Point](#)
- [Find and use a Service Connection Point](#)

Creating a Service Connection Point

6/3/2022 • 4 minutes to read • [Edit Online](#)

The following code example shows how to create and initialize a service connection point. The code example performs additional steps that enable the service to update the SCP values at run time. Typically, a service installer performs these steps as part of installing a service instance on a host computer.

This code example creates the SCP object as a child object for the object of the local computer. It uses the [GetComputerObjectName](#) function to get the DN of the local computer object. It then uses the DN to bind to an [IDirectoryObject](#) pointer for the computer object. The [IDirectoryObject::CreateDSObject](#) method creates the SCP and specifies initial values for the important SCP attributes.

[CreateDSObject](#) returns a pointer to the new SCP, which the code example uses to retrieve an [IADs](#) pointer for the SCP. The code example uses [IADs](#) methods to retrieve the [objectGUID](#) and [distinguishedName](#) attributes of the SCP. The code example uses the [objectGUID](#) to compose a string used to bind to the SCP, and then caches the GUID binding string in the local registry where it can be retrieved by the service at run time. The [distinguishedName](#) is returned to the function caller for use in composing a service principal name (SPN) for the service instance.

The following code example calls this function as part of the basic steps of installing a directory-enabled service. For more information, see [Installing a Service on a Host Computer](#).

```
// ScpCreate
//
// Create a new service connection point as a child object of the
// local server computer object.
//
DWORD
ScpCreate(
    USHORT usPort, // Service's default port to store in SCP.
    LPTSTR szClass, // Service class string to store in SCP.
    LPTSTR szAccount, // Logon account that must access SCP.
    UINT ccDN, // Length of the pszDN buffer in characters
    TCHAR *pszDN) // Returns distinguished name of SCP.

{
    DWORD dwStat, dwAttr, dwLen;
    HRESULT hr;
    IDispatch *pDisp;           // Returned dispinterface of new object.
    IDirectoryObject *pComp;   // Computer object; parent of SCP.
    IADs *pIADsSCP;          // IADs interface on new object.

    if(!szClass || !szAccount || !pszDN || !(ccDN > 0))
    {
        hr = ERROR_INVALID_PARAMETER;
        ReportError(TEXT("Invalid parameter."), hr);
        return hr;
    }

    // Values for SCPs keywords attribute.
    TCHAR* KwVal[]={
        TEXT("83C29870-1DFC-11d3-A193-0000F87A9099"), // Vendor GUID.
        TEXT("A762885A-AA44-11d2-81F1-00C04FB9624E"), // Product GUID.
        TEXT("Microsoft"), // Vendor Name.
        TEXT("Windows 2000 Auth-O-Matic"), // Product Name.
    };

    TCHAR     szServer[MAX_PATH];
    TCHAR     szDn[MAX_PATH];
    TCHAR     szAdcDn[MAX_PATH];
```

```

TCHAR      szAttrs[ADS_MAX_ATTRS];
TCHAR      szPort[6];

HKEY      hReg;
DWORD     dwDisp;

ADSVVALUE cn,objclass,keywords[4],binding,classname,dnsname,nametype;

// SCP attributes to set during creation of SCP.
ADS_ATTR_INFO ScpAttrs[] =
{
{
    TEXT("cn"),
    ADS_ATTR_UPDATE,
    ADSTYPE_CASE_IGNORE_STRING,
    &cn,
    1
},
{
    TEXT("objectClass"),
    ADS_ATTR_UPDATE,
    ADSTYPE_CASE_IGNORE_STRING,
    &objclass,
    1
},
{
    TEXT("keywords"),
    ADS_ATTR_UPDATE,
    ADSTYPE_CASE_IGNORE_STRING,
    keywords,
    4
},
{
    TEXT("serviceDnsName"),
    ADS_ATTR_UPDATE,
    ADSTYPE_CASE_IGNORE_STRING,
    &dnsname,
    1
},
{
    TEXT("serviceDnsNameType"),
    ADS_ATTR_UPDATE,
    ADSTYPE_CASE_IGNORE_STRING,
    &nametype,
    1
},
{
    TEXT("serviceClassName"),
    ADS_ATTR_UPDATE,
    ADSTYPE_CASE_IGNORE_STRING,
    &classname,
    1
},
{
    TEXT("serviceBindingInformation"),
    ADS_ATTR_UPDATE,
    ADSTYPE_CASE_IGNORE_STRING,
    &binding,
    1
},
};

BSTR bstrGuid = NULL;
TCHAR pwszBindByGuidStr[1024];
VARIANT var;

// Get the DNS name of the local computer.
dwLen = sizeof(szServer);
if (!GetComputerNameEx(ComputerNameDnsFullyQualified,szServer,&dwLen))
    dwLen = dwLen - 1;

```

```

    return GetLastError();
    _tprintf(TEXT("GetComputerNameEx: %s\n"), szServer);

    // Enter the attribute values to be stored in the SCP.
    keywords[0].dwType = ADSTYPE_CASE_IGNORE_STRING;
    keywords[1].dwType = ADSTYPE_CASE_IGNORE_STRING;
    keywords[2].dwType = ADSTYPE_CASE_IGNORE_STRING;
    keywords[3].dwType = ADSTYPE_CASE_IGNORE_STRING;

    keywords[0].CaseIgnoreString=KwVal[0];
    keywords[1].CaseIgnoreString=KwVal[1];
    keywords[2].CaseIgnoreString=KwVal[2];
    keywords[3].CaseIgnoreString=KwVal[3];

    cn.dwType           = ADSTYPE_CASE_IGNORE_STRING;
    cn.CaseIgnoreString = TEXT("SockAuthAD");
    objclass.dwType     = ADSTYPE_CASE_IGNORE_STRING;
    objclass.CaseIgnoreString = TEXT("serviceConnectionPoint");

    dnsname.dwType      = ADSTYPE_CASE_IGNORE_STRING;
    dnsname.CaseIgnoreString = szServer;
    classname.dwType     = ADSTYPE_CASE_IGNORE_STRING;
    classname.CaseIgnoreString = szClass;

    _stprintf_s(szPort,TEXT("%d"),usPort);
    binding.dwType       = ADSTYPE_CASE_IGNORE_STRING;
    binding.CaseIgnoreString = szPort;
    nametype.dwType      = ADSTYPE_CASE_IGNORE_STRING;
    nametype.CaseIgnoreString = TEXT("A");

/*
Get the distinguished name of the computer object for the local
computer.
*/
dwLen = sizeof(szDn);
if (!GetComputerObjectName(NameFullyQualifiedDN,szDn,&dwLen))
    return GetLastError();
_tprintf(TEXT("GetComputerObjectName: %s\n"), szDn);

/*
Compose the ADSpath and bind to the computer object for the local
computer.
*/
_tcscopy_s(szAdsPath,TEXT("LDAP://"),MAX_PATH);
_tcscat_s(szAdsPath,szDn,MAX_PATH - _tcslen(szAdsPath));
hr = ADsGetObject(szAdsPath, IID_IDirectoryObject, (void **)&pComp);
if (FAILED(hr)) {
    ReportError(TEXT("Failed to bind Computer Object."),hr);
    return hr;
}

//*****
// Publish the SCP as a child of the computer object
//*****

// Calculate attribute count.
dwAttr = sizeof(ScpAttrs)/sizeof(ADS_ATTR_INFO);

// Complete the action.
hr = pComp->CreateDSObject(TEXT("cn=SockAuthAD"),
                             ScpAttrs, dwAttr, &pDisp);
if (FAILED(hr)) {
    ReportError(TEXT("Failed to create SCP:"), hr);
    pComp -> Release();
    return hr;
}

pComp -> Release();

```

```

// Query for an IADs pointer on the SCP object.
hr = pDisp->QueryInterface(IID_IADs,(void **)&pIADsSCP);
if (FAILED(hr)) {
    ReportError(TEXT("Failed to QueryInterface for IADs:"),hr);
    pDisp->Release();
    return hr;
}
pDisp->Release();

// Set ACEs on the SCP so a service can modify it.
hr = AllowAccessToScpProperties(
    szAccount,      // Service account to allow access.
    pIADsSCP);     // IADs pointer to the SCP object.
if (FAILED(hr)) {
    ReportError(TEXT("Failed to set ACEs on SCP DACL:"), hr);
    return hr;
}

// Get the distinguished name of the SCP.
VariantInit(&var);
hr = pIADsSCP->Get(CComBSTR("distinguishedName"), &var);
if (FAILED(hr)) {
    ReportError(TEXT("Failed to get distinguishedName:"), hr);
    pIADsSCP->Release();
    return hr;
}
_tprintf(TEXT("distinguishedName via IADs: %s\n"), var.bstrVal);

// Return the DN of the SCP, which is used to compose the SPN.
// The best practice is to either accept and return the buffer
// size or do this in a _try / _except block, both omitted here
// for clarity.
_tcscpy_s(pszDN, var.bstrVal, ccdN);

// Retrieve the SCP objectGUID in format suitable for binding.
hr = pIADsSCP->get_GUID(&bstrGuid);
if (FAILED(hr)) {
    ReportError(TEXT("Failed to get GUID:"), hr);
    pIADsSCP->Release();
    return hr;
}

// Build a string for binding to the object by GUID.
_tcscpy_s(pwszBindByGuidStr,
    TEXT("LDAP://<GUID="),
    1024);
_tcscat_s(pwszBindByGuidStr,
    bstrGuid,
    1024 - _tcslen(pwszBindByGuidStr));
_tcscat_s(pwszBindByGuidStr,
    TEXT(">"),
    1024 - _tcslen(pwszBindByGuidStr));
_tprintf(TEXT("GUID binding string: %s\n"),
    pwszBindByGuidStr);

pIADsSCP->Release();

// Create a registry key under
// HKEY_LOCAL_MACHINE\SOFTWARE\Vendor\Product.
dwStat = RegCreateKeyEx(HKEY_LOCAL_MACHINE,
    TEXT("Software\\Fabrikam\\Auth-O-Matic"),
    0,
    NULL,
    REG_OPTION_NON_VOLATILE,
    KEY_ALL_ACCESS,
    NULL,
    &hReg,
    &dwDisp);
if (dwStat != NO_ERROR) {

```

```
    ReportError(TEXT("RegCreateKeyEx failed:"), dwStat);
    return dwStat;
}

// Cache the GUID binding string under the registry key.
dwStat = RegSetValueEx(hReg, TEXT("GUIDBindingString"), 0, REG_SZ,
                      (const BYTE *)pwszBindByGuidStr,
                      2*(_tcslen(pwszBindByGuidStr)));
if (dwStat != NO_ERROR) {
    ReportError(TEXT("RegSetValueEx failed:"), dwStat);
    return dwStat;
}

RegCloseKey(hReg);

// Cleanup should delete the SCP and registry key if an error occurs.

return dwStat;
}
```

Updating a Service Connection Point

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example shows how to update a service connection point. This code is typically executed by a service when it starts.

This example retrieves the SCP's GUID binding string that the service installer cached in the registry. It uses this string to bind to an [IDirectoryObject](#) pointer on the SCP object, and then calls the [IDirectoryObject::GetObjectAttributes](#) method to get the **serviceDNSName** and **serviceBindingInformation** attributes of the SCP. Be aware that your service may require to verify and update additional attributes.

The code compares the **serviceDNSName** value to the DNS name returned by the [GetComputerNameEx](#) function. It also compares the service's current port number to the port number stored in the **serviceBindingInformation** attribute. If either of these values have changed, the code calls the [IDirectoryObject::SetObjectAttributes](#) method to update the SCP attributes.

```
DWORD
ScpUpdate(USHORT usPort)
{
    DWORD    dwStat, dwType, dwLen;
    BOOL    bUpdate=FALSE;

    HKEY    hReg;

    TCHAR    szAdsPath[MAX_PATH];
    TCHAR    szServer[MAX_PATH];
    TCHAR    szPort[8];
    TCHAR    *pszAttrs[]={
        {TEXT("serviceDNSName")},
        {TEXT("serviceBindingInformation")},
    };

    HRESULT        hr;
    IDirectoryObject    *pObj;
    DWORD            dwAttrs;
    int             i;

    PADS_ATTR_INFO  pAttrs;
    ADSVALUE        dnsname,binding;

    ADS_ATTR_INFO   Attrbs[]={
        {TEXT("serviceDnsName"),ADS_ATTR_UPDATE,ADSTYPE_CASE_IGNORE_STRING,&dnsname,1},
        {TEXT("serviceBindingInformation"),ADS_ATTR_UPDATE,ADSTYPE_CASE_IGNORE_STRING,&binding,1},
    };

    // Open the service registry key.
    dwStat = RegOpenKeyEx(
        HKEY_LOCAL_MACHINE,
        TEXT("Software\\Microsoft\\Windows 2000 Auth-O-Matic"),
        0,
        KEY_QUERY_VALUE,
        &hReg);
    if (dwStat != NO_ERROR)
    {
        ReportServiceError("RegOpenKeyEx failed", dwStat);
        return dwStat;
    }

    // Get the GUID binding string used to bind to the service SCP
```

```

// Get the GUID binding string used to bind to the service SCP.
dwLen = sizeof(szAdsPath);
dwStat = RegQueryValueEx(hReg, TEXT("GUIDBindingString"), 0, &dwType,
                        (LPBYTE)szAdsPath, &dwLen);
if (dwStat != NO_ERROR) {
    ReportServiceError("RegQueryValueEx failed", dwStat);
    return dwStat;
}

RegCloseKey(hReg);

// Bind to the SCP.
hr = ADsGetObject(szAdsPath, IID_IDirectoryObject, (void **)&pObj);
if (FAILED(hr))
{
    char szMsg1[1024];
    sprintf_s(szMsg1,
              "ADsGetObject failed to bind to GUID (bind string: %S): ",
              szAdsPath);
    ReportServiceError(szMsg1, hr);
    if(pObj)
    {
        pObj->Release();
    }
    return dwStat;
}

// Retrieve attributes from the SCP.
hr = pObj->GetObjectAttributes(pszAttrs, 2, &pAttrs, &dwAttrs);
if (FAILED(hr)) {
    ReportServiceError("GetObjectAttributes failed", hr);
    pObj->Release();
    return hr;
}

// Get the current port and DNS name of the host server.
_stprintf_s(szPort,TEXT("%d"),usPort);
dwLen = sizeof(szServer);
if (!GetComputerNameEx(ComputerNameDnsFullyQualified,szServer,&dwLen))
{
    pObj->Release();
    return GetLastError();
}

// Compare the current DNS name and port to the values retrieved from
// the SCP. Update the SCP only if nothing has changed.
for (i=0; i<(LONG)dwAttrs; i++)
{
    if ((_tcscmp(TEXT("serviceDNSName"),pAttrs[i].pszAttrName)==0) &&
        (pAttrs[i].dwADsType == ADSTYPE_CASE_IGNORE_STRING))
    {
        if (_tcscmp(szServer,pAttrs[i].pADsValues->CaseIgnoreString) != 0)
        {
            ReportServiceError("serviceDNSName being updated", 0);
            bUpdate = TRUE;
        }
        else
            ReportServiceError("serviceDNSName okay", 0);
    }

    if ((_tcscmp(TEXT("serviceBindingInformation"),pAttrs[i].pszAttrName)==0) &&
        (pAttrs[i].dwADsType == ADSTYPE_CASE_IGNORE_STRING))
    {
        if (_tcscmp(szPort,pAttrs[i].pADsValues->CaseIgnoreString) != 0)
        {
            ReportServiceError("serviceBindingInformation being updated", 0);
            bUpdate = TRUE;
        }
    }
}

```

```
        else
            ReportServiceError("serviceBindingInformation okay", 0);
    }

FreeADsMem(pAttribs);

// The binding data or server name have changed,
// so update the SCP values.
if (bUpdate)
{
    dnsname.dwType          = ADSTYPE_CASE_IGNORE_STRING;
    dnsname.CaseIgnoreString = szServer;
    binding.dwType          = ADSTYPE_CASE_IGNORE_STRING;
    binding.CaseIgnoreString = szPort;
    hr = pObj->SetObjectAttributes(Attribs, 2, &dwAttrs);
    if (FAILED(hr))
    {
        ReportServiceError("ScpUpdate: Failed to set SCP values.", hr);
        pObj->Release();
        return hr;
    }
}

pObj->Release();

return dwStat;
}
```

How Clients Find and Use a Service Connection Point

6/3/2022 • 4 minutes to read • [Edit Online](#)

The following code example shows how a client application searches the Global Catalog for a Service Connection Point (SCP). In this code example, the client application has a hard-coded GUID string that identifies the service. The service installer stored the same GUID string as one of the values of the SCPs multi-value **keywords** attribute.

This sample consists of two routines. The **GetGC** routine retrieves an **IDirectorySearch** pointer for a Global Catalog (GC). The **ScpLocate** routine uses the **IDirectorySearch** methods to search the GC.

The GC contains a partial replica of every object in the forest, but does not contain all of the SCP attributes that the client requires. First, the client must search the GC to find the SCP and retrieve its DN. Then the client uses the SCP's DN to bind to an **IDirectoryObject** pointer on the SCP. The client then calls the **IDirectoryObject::GetObjectAttributes** method to retrieve the rest of the attributes.

```
*****  
//  
// ScpLocate()  
//  
// All strings returned by ScpLocate must be freed by the caller using  
// FreeADsStr after it is finished using them.  
//  
*****  
  
DWORD ScpLocate (   
    LPWSTR *ppszDN,           // Returns distinguished name of SCP.  
    LPWSTR *ppszServiceDNSName, // Returns service DNS name.  
    LPWSTR *ppszServiceDNSNameType, // Returns type of DNS name.  
    LPWSTR *ppszClass,         // Returns name of service class.  
    USHORT *pusPort)          // Returns service port.  
{  
    HRESULT hr;  
    IDirectoryObject *pSCP = NULL;  
    ADS_ATTR_INFO *pPropEntries = NULL;  
    IDirectorySearch *pSearch = NULL;  
    ADS_SEARCH_HANDLE hSearch = NULL;  
  
    // Get an IDirectorySearch pointer for the Global Catalog.  
    hr = GetGCSearch(&pSearch);  
    if (FAILED(hr))  
    {  
        fprintf(stderr,"GetGC failed 0x%x",hr);  
        goto Cleanup;  
    }  
  
    // Set up a deep search.  
    // Thousands of objects are not expected in this example, therefore  
    // query for 1000 rows per page.  
    ADS_SEARCPREF_INFO SearchPref[2];  
    DWORD dwPref = sizeof(SearchPref)/sizeof(ADS_SEARCPREF_INFO);  
    SearchPref[0].dwSearchPref = ADS_SEARCPREF_SEARCH_SCOPE;  
    SearchPref[0].vValue.dwType = ADSTYPE_INTEGER;  
    SearchPref[0].vValue.Integer = ADS_SCOPE_SUBTREE;  
  
    SearchPref[1].dwSearchPref = ADS_SEARCPREF_PAGESIZE;  
    SearchPref[1].vValue.dwType = ADSTYPE_INTEGER;  
    SearchPref[1].vValue.Integer = 1000;
```

```

SearchPref[1].Value.Integer = 1000;

hr = pSearch->SetSearchPreference(SearchPref, dwPref);
fprintf (stderr, "SetSearchPreference: 0x%x\n", hr);
if (FAILED(hr))
{
    fprintf (stderr, "Failed to set search prefs: hr:0x%x\n", hr);
    goto Cleanup;
}

// Execute the search. From the GC get the distinguished name
// of the SCP. Use the DN to bind to the SCP and get the other
// properties.
LPWSTR rgszDN[] = {L"distinguishedName"};

// Search for a match of the product GUID.
hr = pSearch->ExecuteSearch(    L"keywords=A762885A-AA44-11d2-81F1-00C04FB9624E",
                                rgszDN,
                                1,
                                &hSearch);

fprintf (stderr, "ExecuteSearch: 0x%x\n", hr);

if (FAILED(hr))
{
    fprintf (stderr, "ExecuteSearch failed: hr:0x%x\n", hr);
    goto Cleanup;
}

// Loop through the results. Each row should be an instance of the
// service identified by the product GUID.
// Add logic to select from multiple service instances.
hr = pSearch->GetNextRow(hSearch);
if (SUCCEEDED(hr) && hr !=S_ADS_NOMORE_ROWS)
{
    ADS_SEARCH_COLUMN Col;

    hr = pSearch->GetColumn(hSearch, L"distinguishedName", &Col);
    *ppszDN = AllocADSStr(Col.pADsValues->CaseIgnoreString);
    pSearch->FreeColumn(&Col);
    hr = pSearch->GetNextRow(hSearch);
}

// Bind to the DN to get the other properties.
LPWSTR lpszLDAPPrefix = L"LDAP://";
DWORD dwSCPPathLength = wcslen(lpszLDAPPrefix) + wcslen(*ppszDN) + 1;
LPWSTR pwszSCPPath = new WCHAR[dwSCPPathLength];
if(pwszSCPPath)
{
    wcscpy_s(pwszSCPPath, lpszLDAPPrefix);
    wcscat_s(pwszSCPPath, *ppszDN);
}
else
{
    fprintf(stderr,"Failed to allocate a buffer");
    goto Cleanup;
}

hr = ADsGetObject( pwszSCPPath,
                   IID_IDirectoryObject,
                   (void**)&pSCP);

// Free the string buffer
delete pwszSCPPath;

if (SUCCEEDED(hr))
{
    // Properties to retrieve from the SCP object.
    LPWSTR rgszAttribs[]=

```

```

    {
        {L"serviceClassName"},
        {L"serviceDNSName"},
        {L"serviceDNSNameType"},
        {L"serviceBindingInformation"}
    };

    DWORD dwAttrs = sizeof(rgszAttrs)/sizeof(LPWSTR);
    DWORD dwNumAttrGot;
    hr = pSCP->GetObjectAttributes( rgszAttrs,
                                      dwAttrs,
                                      &pPropEntries,
                                      &dwNumAttrGot);

    if(FAILED(hr))
    {
        fprintf (stderr, "GetObjectAttributes Failed. hr:0x%x\n", hr);
        goto Cleanup;
    }

    // Loop through the entries returned by GetObjectAttributes
    // and save the values in the appropriate buffers.
    for (int i = 0; i < (LONG)dwAttrs; i++)
    {
        if ((wcscmp(L"serviceDNSName", pPropEntries[i].pszAttrName)==0) &&
            (pPropEntries[i].dwADsType == ADSTYPE_CASE_IGNORE_STRING))
        {
            *ppszServiceDNSName = AllocADsStr(pPropEntries[i].pADsValues->CaseIgnoreString);
        }

        if ((wcscmp(L"serviceDNSNameType", pPropEntries[i].pszAttrName)==0) &&
            (pPropEntries[i].dwADsType == ADSTYPE_CASE_IGNORE_STRING))
        {
            *ppszServiceDNSNameType = AllocADsStr(pPropEntries[i].pADsValues->CaseIgnoreString);
        }

        if ((wcscmp(L"serviceClassName", pPropEntries[i].pszAttrName)==0) &&
            (pPropEntries[i].dwADsType == ADSTYPE_CASE_IGNORE_STRING))
        {
            *ppszClass = AllocADsStr(pPropEntries[i].pADsValues->CaseIgnoreString);
        }

        if ((wcscmp(L"serviceBindingInformation", pPropEntries[i].pszAttrName)==0) &&
            (pPropEntries[i].dwADsType == ADSTYPE_CASE_IGNORE_STRING))
        {
            *pusPort=(USHORT)_wtoi(pPropEntries[i].pADsValues->CaseIgnoreString);
        }
    }
}

Cleanup:
    if (pSCP)
    {
        pSCP->Release();
        pSCP = NULL;
    }

    if (pPropEntries)
    {
        FreeADsMem(pPropEntries);
        pPropEntries = NULL;
    }

    if (pSearch)
    {
        if (hSearch)
        {
            pSearch->CloseSearchHandle(hSearch);
            hSearch = NULL;
        }
    }
}

```

```

        pSearch->Release();
        pSearch = NULL;
    }

    return hr;
}

//*****
// GetGCSearch()
//
// Retrieves an IDirectorySearch pointer for a Global Catalog (GC)
//
//*****

HRESULT GetGCSearch(IDirectorySearch **ppDS)
{
    HRESULT hr;
    IEnumVARIANT *pEnum = NULL;
    IADsContainer *pCont = NULL;
    IDispatch *pDisp = NULL;
    VARIANT var;
    ULONG lFetch;

    *ppDS = NULL;

    // Bind to the GC: namespace container object. The true GC DN
    // is a single immediate child of the GC: namespace, which must
    // be obtained using enumeration.
    hr = ADsOpenObject( L"GC:",
                        NULL,
                        NULL,
                        ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                        IID_IADsContainer,
                        (void**)&pCont);
    if (FAILED(hr))
    {
        _tprintf(TEXT("ADsOpenObject failed: 0x%x\n"), hr);
        goto cleanup;
    }

    // Get an enumeration interface for the GC container.
    hr = ADsBuildEnumerator(pCont, &pEnum);
    if (FAILED(hr))
    {
        _tprintf(TEXT("ADsBuildEnumerator failed: 0x%x\n"), hr);
        goto cleanup;
    }

    // Now enumerate. There is only one child of the GC: object.
    hr = ADsEnumerateNext(pEnum, 1, &var, &lFetch);
    if (FAILED(hr))
    {
        _tprintf(TEXT("ADsEnumerateNext failed: 0x%x\n"), hr);
        goto cleanup;
    }

    if ((hr == S_OK) && (lFetch == 1))
    {
        pDisp = V_DISPATCH(&var);
        hr = pDisp->QueryInterface( IID_IDirectorySearch, (void**)ppDS);
    }

cleanup:
    if (pEnum)
    {
        ADsFreeEnumerator(pEnum);
        pEnum = NULL;
    }
}

```

```
}

if (pCont)
{
    pCont->Release();
    pCont = NULL;
}

if (pDisp)
{
    pDisp->Release();
    pDisp = NULL;
}

return hr;
}
```

Publishing with the RPC Name Service

6/3/2022 • 2 minutes to read • [Edit Online](#)

RPC services can publish their identifiers in a namespace using the RPC name service (RpcNs) APIs. The RpcNs APIs in Windows 2000 publish the RPC entries in Active Directory Domain Services. Services create RPC bindings and publish them in the namespace as named RPC Server entries with attributes including the unique interface ID, a GUID that is recognized by clients. A client can then search for RPC Servers offering the desired interface, import the binding, and connect to the server.

For more information about publishing an RPC service, see [Server-side Binding](#).

Publishing with Windows Sockets Registration & Resolution

6/3/2022 • 2 minutes to read • [Edit Online](#)

Windows Sockets services can use the Registration and Resolution (RnR) APIs to publish services and look up services published with RnR. RnR publication occurs in two steps. The first step installs a service class that associates a GUID with a name for the service. The service class can hold service-specific configuration data. Services can then publish themselves as instances of the service class. When published, clients can query the directory service for instances of a given class using the RnR APIs and select an instance to bind to. When a class is no longer required, it can be removed.

Example Code for Installing an RnR Service Class

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following program installs an RnR Service class.

```
// Add activeds.dll to your project
// Add adtiveds.lib to your project
// Add adsiid.lib to your project

#include "stdafx.h"
#include "windows.h"
#include "stdio.h"
#include "activeds.h"

const IID IID_IAD = {0xFD8256D0, 0xFD15, 0x11CE, {0xAB,0xC4,0x02,0x60,0x8C,0x9E,0x75,0x53}};

// Entry point for the application
int wmain(int argc, WCHAR* argv[])
{
    HRESULT hr;
    IADsContainer *pAbsSchema = NULL; // For the abstract schema
    IADsClass *pClass = NULL; // For class objects
    IADsProperty *pProp = NULL; // For attribute objects
    IADsSyntax *pSyntax = NULL; // For syntax objects
    IEnumVARIANT *pEnum = NULL;
    ULONG lFetch;
    VARIANT var;
    VARIANT_BOOL bMulti, bAbstract, bAux;
    BSTR bstrPI, bstrClass, bstrName;
    LONG lCount = 0;
    LONG lVarType = 0;
    IADs *pChild = NULL;
    DWORD dwUnknownClass = 0;

    CoInitialize(NULL);

    // Bind to the abstract schema.
    hr = ADsGetObject(L"LDAP://schema",
                      IID_IADsContainer,
                      (void**)&pAbsSchema);
    if (FAILED(hr))
        goto cleanup;

    // Enumerate the attribute and class entries in the abstract schema.
    hr = ADsBuildEnumerator(pAbsSchema, &pEnum);
    if (FAILED(hr))
        goto cleanup;

    VariantInit(&var);
    hr = ADsEnumerateNext(pEnum, 1, &var, &lFetch);
    while(hr == S_OK && lFetch == 1)
    {
        // Identify whether this is a class, attribute, or syntax.
        hr = V_DISPATCH(&var)->QueryInterface(IID_IADs,
                                                (void**) &pChild);
        if (FAILED(hr))
            goto cleanuploop;
        hr = pChild->get_Class(&bstrClass);
        if ( FAILED(hr) )
            goto cleanuploop;
        wprintf(L"%s", bstrClass );
        hr = pChild->get_Name(&bstrName);
```

```

if (FAILED(hr))
    goto cleanuploop;
wprintf(L"%s", bstrName );

// Retrieve data, depending on the type of schema element.
if (_wcsicmp(L"Class", bstrClass)==0)
{
    hr = pChild->QueryInterface(IID_IADsClass,
                                (void**) &pClass);
    if (FAILED(hr))
        goto cleanuploop;
    pClass->get_Abstract(&bAbstract);
    pClass->get_Auxiliary(&bAux);
    if (bAbstract)
        wprintf(L",Abstract");
    else if (bAux)
        wprintf(L",Auxiliary");
    else
        wprintf(L",Structural");

    // Retrieve the primary ADSI
    // interface to use with this class.
    pClass->get_PrimaryInterface(&bstrPI);
    if (_wcsicmp(L"\{FD8256D0-FD15-11CE-ABC4-02608C9E7553\}",
                bstrPI)==0)
        wprintf(L",IID_IADS,%s", bstrPI);
    if (_wcsicmp(L"\{B15160D0-1226-11CF-A985-00AA006BC149\}",
                bstrPI)==0)
        wprintf(L",IID_IADsPrintQueue,%s", bstrPI);
    if (_wcsicmp(L"\{A2F733B8-EFFE-11CF-8ABC-00C04FD8D503\}",
                bstrPI)==0)
        wprintf(L",IID_IADsOU,%s", bstrPI);
    if (_wcsicmp(L"\{A05E03A2-EFFE-11CF-8ABC-00C04FD8D503\}",
                bstrPI)==0)
        wprintf(L",IID_IADsLocality,%s", bstrPI);
    if (_wcsicmp(L"\{3E37E320-17E2-11CF-ABC4-02608C9E7553\}",
                bstrPI)==0)
        wprintf(L",IID_IADsUser,%s", bstrPI);
    if (_wcsicmp(L"\{27636B00-410F-11CF-B1FF-02608C9E7553\}",
                bstrPI)==0)
        wprintf(L",IID_IADsGroup,%s", bstrPI);
    if (_wcsicmp(L"\{00E4C220-FD16-11CE-ABC4-02608C9E7553\}",
                bstrPI)==0)
        wprintf(L",IID_IADsDomain,%s", bstrPI);
    SysFreeString(bstrPI);
}

else if (_wcsicmp(L"Property", bstrClass)==0)
{
    hr = pChild->QueryInterface(IID_IADsProperty,
                                (void**) &pProp );
    if (FAILED(hr))
        goto cleanuploop;
    pProp->get_MultiValued(&bMulti);
    wprintf(L"%s", bMulti ? L"Multi-Valued" : L"Single-Valued");
}

else if (_wcsicmp(L"Syntax", bstrClass)==0)
{
    hr = pChild->QueryInterface(IID_IADsSyntax,
                                (void**) &pSyntax);
    if (FAILED(hr))
        goto cleanuploop;
    pSyntax->get_OleAutoDataType (&lVarType);
    wprintf(L"%u", lVarType);
}

else
    dwUnknownClass++;

cleanuploop:
    wprintf(L"\n");

```

```
SysFreeString(bstrClass);
SysFreeString(bstrName);
pChild->Release();
VariantClear(&var);
if (SUCCEEDED(hr))
    hr = ADsEnumerateNext(pEnum, 1, &var, &lFetch);
}

wprintf(L"dwUnknownClass: %u\n", dwUnknownClass);
cleanup:
if (pAbsSchema)
    pAbsSchema->Release();
if (pEnum)
    pEnum->Release();
VariantClear(&var);
CoUninitialize();

return hr;
}
```

Example Code for Implementing a Winsock Service with an RnR Publication

6/3/2022 • 3 minutes to read • [Edit Online](#)

The following code example implements the example Winsock service with RnR publication.

This sample uses the `serverRegister` function defined in the [Example Code for Publishing the RnR Connection Point](#) topic and the `serverUnregister` function defined in the [Example Code for Removing the RnR Connection Point](#) topic.

```
// Add Ws2_32.lib to the project.

#include <stdafx.h>
#include <winsock2.h>
#include <stdio.h>

// {A9033BC1-ECA4-11cf-A054-00AA006C33ED}
static GUID SVCID_EXAMPLE_SERVICE =
{ 0xa9033bc1, 0xec4, 0x11cf, { 0xa0, 0x54, 0x0, 0xaa, 0x0, 0x6c, 0x33, 0xed } };

INT serverRegister(SOCKADDR *sa,
                    GUID *pServiceID,
                    LPTSTR pszServiceInstanceName,
                    LPTSTR pszServiceInstanceComment);
INT serverUnregister(SOCKADDR *sa,
                     GUID *pServiceID,
                     LPTSTR pszServiceInstanceName,
                     LPTSTR pszServiceInstanceComment);

// Entry point for the application.
int main(void)
{
    LPTSTR pszServiceInstanceName = TEXT("Example Service Instance");
    LPTSTR pszServiceInstanceComment = TEXT("ExampleService instance registered in the directory service
through RnR");

    // Data structures used to initialize Winsock.
    WSADATA wsData;
    WORD wVer = MAKEWORD(2,2);

    // Data structures used to setup communications.
    SOCKET s, newsock;
    SOCKADDR sa;
    struct hostent *he;
    char szName[255];
    struct sockaddr_in sa_in;
    INT ilen;
    ULONG icmd;
    ULONG ulLen;

    // Miscellaneous variables
    INT status;
    WCHAR wszRecvBuf[100];

    memset(wszRecvBuf,0,sizeof(wszRecvBuf));

    // Begin: Init Winsock2
    status = WSAStartup(wVer,&wsData);
    if (status != NO_ERROR)
    {
```

```

        return -1;
    }

    // Create the socket.
    s = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    if (INVALID_SOCKET == s)
    {
        printf("Failed to create socket: %d\n",WSAGetLastError());
        WSACleanup();
        return -1;
    }

    // Disable non-blocking I/O for this example.
    icmd = 0;
    status = ioctlsocket(s,FIONBIO,&icmd);

    // Bind the socket to a dynamically assigned port.
    sa.sa_family=AF_INET;
    memset(sa.sa_data,0,sizeof(sa.sa_data));

    status = bind(s,&sa,sizeof(sa));

    // Convert the port to the local host byte order.
    ilen = sizeof(sa_in);
    status = getsockname(s,(struct sockaddr *)&sa_in,&ilen);
    if (status == NO_ERROR)
    {
        printf("Server: Bound to port %d\n",ntohs(sa_in.sin_port));
    }

    // Identify the net address to fill in to register ourselves
    // in the directory service. Explicitly call the ANSI text version
    // because gethostbyname does not recognize Unicode.

    ilen = sizeof(szName);
    GetComputerNameA(szName,&ulLen);
    he = gethostbyname(szName);

    // Put the address in the SOCKADDR structure.
    sa_in.sin_addr.S_un.S_addr = *((long *)(he->h_addr));

    // Listen for connections. SOMAXCONN tells the provider to queue
    // a "reasonable" number of connections.
    status = listen(s,SOMAXCONN);
    if (status != NO_ERROR)
    {
        printf("Failed to set socket listening: %d\n",
               WSAGetLastError());
        WSACleanup();
        return -1;
    }

    // Register this instance with RnR.
    status = serverRegister((SOCKADDR *)&sa_in,
                           &SVCID_EXAMPLE_SERVICE,
                           pszServiceInstanceName,
                           pszServiceInstanceComment);
    if (status != NO_ERROR)
    {
        printf("Failed to register instance: %d\n",WSAGetLastError());
        WSACleanup();
        return -1;
    }
    printf("Server: Registered instance in the directory service\n");

    // Block waiting for a connection. For simplicity, this example
    // is single-threaded. In a real service, spin off one or more threads
    // here to call AcceptEx here and process the connections through a
    // completion port.

```

```

ilen = sizeof(sa);
newsock = accept(s,&sa,&ilen);
if (newsock == INVALID_SOCKET)
{
    printf("Failed to create socket: %d\n",WSAGetLastError());
}
else
{
    printf("Server: There is a client connection\n");

    // Receive a message from the client and end.
    status = recv(newsock,(char *)wszRecvBuf,sizeof(wszRecvBuf),0);
    if (status > 0)
    {
        printf("Server: Received: %S\n",wszRecvBuf);
    }
}

// Unregister and end.
printf("Unregistering and shutting down.\n");
status = serverUnregister((SOCKADDR *)&sa_in,
    &SVCID_EXAMPLE_SERVICE,
    pszServiceInstanceName,
    pszServiceInstanceComment);

WSACleanup();

return 0;
}

// The server Unregister function removes the rnr connection point.
INT serverUnregister(SOCKADDR *sa,
    GUID *pServiceID,
    LPTSTR pszServiceInstanceName,
    LPTSTR pszServiceInstanceComment)
{
    DWORD ret;
    WSAVERSION Version;
    WSAQUERYSET QuerySet;
    CSADDR_INFO CSAddrInfo[1];
    SOCKADDR sa_local;

    memset(&QuerySet, 0, sizeof(QuerySet));
    memset(&CSAddrInfo, 0, sizeof(CSAddrInfo));
    memset(&sa_local, 0, sizeof(SOCKADDR));
    sa_local.sa_family = AF_INET;

    // Build the CSAddrInfo structure to contain address
    // data. This is what clients use to create a connection.
    //
    // Be aware that the LocalAddr is set to zero because
    // dynamically assigned port numbers are used.
    //
    CSAddrInfo[0].LocalAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].LocalAddr.lpSockaddr = &sa_local;
    CSAddrInfo[0].RemoteAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].RemoteAddr.lpSockaddr = sa;
    CSAddrInfo[0].iSocketType = SOCK_STREAM;
    CSAddrInfo[0].iProtocol = PF_INET;

    QuerySet.dwSize = sizeof(WSAQUERYSET);
    QuerySet.lpServiceClassId = pServiceID;
    QuerySet.lpszServiceInstanceName = pszServiceInstanceName;
    QuerySet.lpszComment = pszServiceInstanceComment;
    QuerySet.lpVersion = &Version;
    QuerySet.lpVersion->dwVersion = 2;
    QuerySet.lpVersion->ecHow = COMP_NOTLESS;
    QuerySet.dwNameSpace = NS_NTDS;
    QuerySet.dwNumberOfCsAddrs = 1;
}

```

```

    QuerySet.lpcsaBuffer = CSAddrInfo;

    ret = WSASetService( &QuerySet,
                        RNRSERVICE_DEREGISTER,
                        SERVICE_MULTIPLE);

    return(ret);
}

// The serverRegister function publishes the rnr connection point.
INT serverRegister(SOCKADDR * sa,
                    GUID *pServiceID,
                    LPTSTR pszServiceInstanceName,
                    LPTSTR pszServiceInstanceComment)
{
    DWORD ret;
    WSAVERSION Version;
    WSAQUERYSET QuerySet;
    CSADDR_INFO CSAddrInfo[1];
    SOCKADDR sa_local;

    memset(&QuerySet, 0, sizeof(QuerySet));
    memset(&CSAddrInfo, 0, sizeof(CSAddrInfo));
    memset(&sa_local, 0, sizeof(SOCKADDR));
    sa_local.sa_family = AF_INET;

    // Build the CSAddrInfo structure to contain address
    // data that clients use to establish a connection.
    //
    // Be aware that LocalAddr is set to zero because dynamically
    // assigned port numbers are used.
    //
    CSAddrInfo[0].LocalAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].LocalAddr.lpSockaddr = &sa_local;
    CSAddrInfo[0].RemoteAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].RemoteAddr.lpSockaddr = sa;
    CSAddrInfo[0].iSocketType = SOCK_STREAM;
    CSAddrInfo[0].iProtocol = PF_INET;

    QuerySet.dwSize = sizeof(WSAQUERYSET);
    QuerySet.lpServiceClassId = pServiceID;
    QuerySet.lpszServiceInstanceName = pszServiceInstanceName;
    QuerySet.lpszComment = pszServiceInstanceComment;
    QuerySet.lpVersion = &Version;
    QuerySet.lpVersion->dwVersion = 2;
    QuerySet.lpVersion->ecHow = COMP_NOTLESS;
    QuerySet.dwNameSpace = NS_NTDS;
    QuerySet.dwNumberOfCsAddrs = 1;
    QuerySet.lpcsaBuffer = CSAddrInfo;

    ret = WSASetService( &QuerySet,
                        RNRSERVICE_REGISTER,
                        SERVICE_MULTIPLE);

    return(ret);
}

```

Example Code for Publishing the RnR Connection Point

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example is used by the Winsock service to register the RnR connection point for the service.

```

#include <winsock2.h>
#include <stdio.h>

//*****************************************************************************
serverRegister()

Registers the RnR connection point for the specified service. WSAStartup
must be called before calling this function.

***** */

INT serverRegister(SOCKADDR * sa,
                    GUID *pServiceID,
                    LPTSTR pszServiceInstanceName,
                    LPTSTR pszServiceInstanceComment)
{
    DWORD ret;
    WSAVERSION Version;
    WSAQUERYSET QuerySet;
    CSADDR_INFO CSAddrInfo[1];
    SOCKADDR sa_local;

    memset(&QuerySet, 0, sizeof(QuerySet));
    memset(&CSAddrInfo, 0, sizeof(CSAddrInfo));
    memset(&sa_local, 0, sizeof(SOCKADDR));
    sa_local.sa_family = AF_INET;

    // Build the CSAddrInfo structure to contain address
    // data that clients use to establish a connection.
    //
    // Be aware that LocalAddr is set to zero because dynamically
    // assigned port numbers are used.
    //
    CSAddrInfo[0].LocalAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].LocalAddr.lpSockaddr = &sa_local;
    CSAddrInfo[0].RemoteAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].RemoteAddr.lpSockaddr = sa;
    CSAddrInfo[0].iSocketType = SOCK_STREAM;
    CSAddrInfo[0].iProtocol = PF_INET;

    QuerySet.dwSize = sizeof(WSAQUERYSET);
    QuerySet.lpServiceClassId = pServiceID;
    QuerySet.lpszServiceInstanceName = pszServiceInstanceName;
    QuerySet.lpszComment = pszServiceInstanceComment;
    QuerySet.lpVersion = &Version;
    QuerySet.lpVersion->dwVersion = 2;
    QuerySet.lpVersion->ecHow = COMP_NOTLESS;
    QuerySet.dwNameSpace = NS_NTDS;
    QuerySet.dwNumberOfCsAddrs = 1;
    QuerySet.lpcsaBuffer = CSAddrInfo;

    ret = WSASetService( &QuerySet,
                        RNRSERVICE_REGISTER,
                        SERVICE_MULTIPLE);

    return(ret);
}

```

Example Code for Removing the RnR Connection Point

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example is used by the Winsock service code example to unregister the RnR connection point for the service.

```

#include <winsock2.h>
#include <stdio.h>

//******************************************************************************

serverUnregister()

Unregisters the RnR connection point for the specified service.
WSAStartup must be called before calling this function.

*****/


INT serverUnregister(SOCKADDR *sa,
                      GUID *pServiceID,
                      LPTSTR pszServiceInstanceName,
                      LPTSTR pszServiceInstanceComment)
{
    DWORD ret;
    WSAVERSION Version;
    WSAQUERYSET QuerySet;
    CSADDR_INFO CSAddrInfo[1];
    SOCKADDR sa_local;

    memset(&QuerySet, 0, sizeof(QuerySet));
    memset(&CSAddrInfo, 0, sizeof(CSAddrInfo));
    memset(&sa_local, 0, sizeof(SOCKADDR));
    sa_local.sa_family = AF_INET;

    // Build the CSAddrInfo structure to contain address
    // data. This is what clients use to create a connection.
    //
    // Be aware that the LocalAddr is set to zero because
    // dynamically assigned port numbers are used.
    //
    CSAddrInfo[0].LocalAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].LocalAddr.lpSockaddr = &sa_local;
    CSAddrInfo[0].RemoteAddr.iSockaddrLength = sizeof(SOCKADDR);
    CSAddrInfo[0].RemoteAddr.lpSockaddr = sa;
    CSAddrInfo[0].iSocketType = SOCK_STREAM;
    CSAddrInfo[0].iProtocol = PF_INET;

    QuerySet.dwSize = sizeof(WSAQUERYSET);
    QuerySet.lpServiceClassId = pServiceID;
    QuerySet.lpszServiceInstanceName = pszServiceInstanceName;
    QuerySet.lpszComment = pszServiceInstanceComment;
    QuerySet.lpVersion = &Version;
    QuerySet.lpVersion->dwVersion = 2;
    QuerySet.lpVersion->ecHow = COMP_NOTLESS;
    QuerySet.dwNameSpace = NS_NTDS;
    QuerySet.dwNumberOfCsAddrs = 1;
    QuerySet.lpcsaBuffer = CSAddrInfo;

    ret = WSASetService( &QuerySet,
                        RNRSERVICE_DEREGISTER,
                        SERVICE_MULTIPLE);

    return(ret);
}

```

Example Code for Locating a Service Using an RnR Query

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example locates the example Winsock service and connects to it.

```
#include "stdafx.h"
#include "shlobj.h"
#include "dsclient.h"

// The PrintDomain() function displays the domain name.
void PrintDomain(DOMAINDESC *pDomainDesc, DWORD dwIndentLevel)
{
    DWORD i;

    // Display the domain name.
    for(i = 0; i < dwIndentLevel; i++)
    {
        wprintf(L"  ");
    }
    wprintf(pDomainDesc->pSzName);
    wprintf(L"\n");
}

// The WalkDomainTree() function prints the current domain name and walks the tree.
void WalkDomainTree(DOMAINDESC *pDomainDesc, DWORD dwIndentLevel = 0)
{
    DOMAINDESC *pCurrent;

    // Walk through the current item and any siblings it may have.
    for(pCurrent = pDomainDesc;
        NULL != pCurrent;
        pCurrent = pCurrent->pdNextSibling)
    {
        // Print the current domain name.
        PrintDomain(pCurrent, dwIndentLevel);

        // Walk the child tree, if one exists.
        if(NULL != pCurrent->pdChildList)
        {
            WalkDomainTree(pCurrent->pdChildList,
                           dwIndentLevel + 1);
        }
    }
}

// Entry point for the application.
int main(void)
{
    HRESULT hr;
    IDsBrowseDomainTree *pBrowseTree;
    CoInitialize(NULL);

    hr = CoCreateInstance(CLSID_DsDomainTreeBrowser,
                          NULL,
                          CLSCTX_INPROC_SERVER,
                          IID_IDsBrowseDomainTree,
                          (void**)&pBrowseTree);

    if(SUCCEEDED(hr))
```

```
{  
    DOMAINTREE *pDomTreeStruct;  
  
    hr = pBrowseTree->GetDomains(&pDomTreeStruct,  
                                DBDTF_RETURNFQDN);  
    if(SUCCEEDED(hr))  
    {  
        WalkDomainTree(&pDomTreeStruct->aDomains[0]);  
        hr = pBrowseTree->FreeDomains(&pDomTreeStruct);  
    }  
  
    pBrowseTree->Release();  
}  
  
CoUninitialize();  
}
```

Publishing COM+ Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

COM-based services provide an application-proxy in the form of a Windows installer (MSI) package. This .msi file contains the server name to be used and other required elements, such as proxy/stubs and type libraries required for marshalling. The Component Services snap-in auto-generate these application proxies for COM+ Server applications.

The application proxies are published into policy objects in Active Directory using the Group Policy Editor. No special intervention is required in the client application. The computer/user account on the client computer must be in an OU configured to use the policy object in which the application proxies are published. The COM binder automatically locates the server by means of the directory when the client establishes an instance of the objects concerned.

Service Logon Accounts

6/3/2022 • 2 minutes to read • [Edit Online](#)

A service, like any process, has a primary security identity that determines the granted access rights and privileges for local and network resources. This security identity, or security context, also determines the potential the service has for damaging local and network resources.

The security context for a Microsoft Win32 service is determined by the logon account that is used to start the service. This section discusses programming issues and best practices relating to the service logon account used by Win32 services, with a focus on directory-enabled services. This section includes the following topics:

- [About Service Logon Accounts](#)—An overview of service logon accounts and security context programming issues for a Win32 service.
- [Guidelines for Selecting a Service Logon Account](#) for a Win32 service.
- [Setting up a Service's User Account](#).
- [Installing a Service on a Host Computer](#) and specifying the service logon account.
- [Granting Logon as Service Right on the Host Computer](#)—Granting the service's user account the logon as a service right on the host computer.
- [Testing Whether Running on a Domain Controller](#)—Detecting at installation time whether the service instance is being installed on a domain controller.
- [Granting Access Rights to the Service Logon Account](#)—Setting and maintaining ACEs and group memberships to ensure that the system will grant the running service access to the necessary local and network resources.
- [Changing the Password on a Service's User Account](#)—Changing the password on a service's user account, and at the same time updating the password registered with the service control manager on each host server on which the service is installed.
- [Mutual Authentication Using Kerberos](#)—Maintaining service principal name (SPN) registration on the directory object associated with the logon account of each instance of your service. SPNs enable clients to authenticate a service using Kerberos mutual authentication.
- [Converting Domain Account Name Formats](#)—For example, converting a distinguished name to *Domain****UserName* format, and vice versa.

About Service Logon Accounts

6/3/2022 • 2 minutes to read • [Edit Online](#)

When a Win32-based service starts, it logs on to the local computer. It can log on as:

- A local or domain user account.
- The LocalSystem account.

The logon account determines the security identity of the service at run time, that is, the service's primary security context. The security context determines the service's ability to access local and network resources. For example, a service running in the security context of a local user account cannot access network resources. Conversely, a service running in the security context of the LocalSystem account on a Windows 2000 domain controller (DC), would have unrestricted access to the DC. For more information, and a discussion of the benefits and limitations between user accounts and LocalSystem, see [Security Contexts and Active Directory Domain Services](#).

Ultimately, administrators on the system where the service is installed have control over the service's logon account. For security reasons, some administrators may not allow you to install your service under the LocalSystem account. Your service must be able to run under a domain user account. As a programmer, you can exercise some control over your service's logon account. Your service installer specifies the service's logon account when it calls the [CreateService](#) function to install the service on a host computer. Your installer can suggest a default logon account, but it should allow an administrator to specify the actual account.

Your installer can also perform the following tasks relating to your service's logon account:

- Installation. If installing your service to run under a user account, the account must exist before you call [CreateService](#). You can use an existing account or create one as part of the host-computer installrt. For more information, see [Setting up a Service's User Account](#).
- Authentication. If you want clients to use Kerberos mutual authentication, register the SPNs on the service's logon account. If the service runs under the LocalSystem account, the service's logon account is the computer account of the host computer. For more information, see [Service Principal Names](#).
- Grant Access. Ensure that the service at run time has the access rights and privileges that it requires to perform its tasks. This can require setting access-control entries (ACEs) in the security descriptors of various resources, that is directory objects, file shares, and so on, to allow the necessary access rights to the user or computer account. For more information, see [Granting Access Rights to the Service Logon Account](#).
- Set Privileges. Assign privileges to the specified logon account, such as the right to logon as a service to the host computer. For more information, see [Granting Logon as Service Right on the Host Computer](#).

After a service is installed, there are maintenance tasks that relate to your service logon account. For more information, see [Logon Account Maintenance Tasks](#).

- Password maintenance. For a service that runs under a user account, you must periodically change the password and keep the password in sync with the password used by one or more local service control managers to start the service.
- SPN maintenance. If a service logon account changes, remove the SPNs registered on the old account and register them on the new account. Be aware that when a service is installed, a domain administrator can change the account under which your service runs; use Win32 functions or the user interface of the Computer Management administrative tool.
- ACE maintenance. If a service logon account changes, you need to update ACEs and group memberships to ensure that the service can still access the necessary resources.

Guidelines for Selecting a Service Logon Account

6/3/2022 • 2 minutes to read • [Edit Online](#)

A Win32-based service can run in the security context of a local user account, a domain user account, or the LocalSystem account. To decide which account to use, an administrator should install the service with the minimum set of permissions required to perform the service operations. In a typical directory-enabled service, this means the service installer should create a domain user account for the service and grant that account the specific access rights and privileges required by the service at run time. A service should only run under the LocalSystem account if the service requires administrative privileges or must act as part of the operating system on the local computer.

Be aware that the service installer should, by default, set up the service to run under a domain user account. To run the service under the LocalSystem account, the service installer should query the administrator for permission to do so.

For more information about descriptions, advantages, and disadvantages of each account type, see:

- [Local User Accounts](#).
- [Domain User Accounts](#).
- [The LocalSystem Account](#).

Using a Local User Account as a Service Logon Account

6/3/2022 • 2 minutes to read • [Edit Online](#)

A local user account (name format: "`\UserName`") exists only in the SAM database of the host computer; it does not have a user object in Active Directory Domain Services. This means that a local account cannot be authenticated by the domain. Consequently, a service that runs in the security context of a local user account does not have access to network resources (except as an anonymous user), and it cannot support Kerberos mutual authentication in which the service is authenticated by its clients. For these reasons, local user accounts are typically inappropriate for directory-enabled services. On the plus side, bugs in the service cannot damage the system. If your service can run under those limitations, it should.

Using a domain user account as a service logon account

6/3/2022 • 2 minutes to read • [Edit Online](#)

NOTE

The following documentation is for developers. If you're an end-user looking for information about an error message involving domain user accounts, then see the [Microsoft community forums](#). For information about managing domain user accounts, see [TechNet](#).

A domain user account enables the service to take full advantage of the service security features of Windows and Microsoft Active Directory Domain Services. The service has whatever local and network access is granted to the account, or to any groups of which the account is a member. The service can support Kerberos mutual authentication.

The advantage of using a domain user account is that the service's actions are limited by the access rights and privileges associated with the account. Unlike a `LocalSystem` service, bugs in a user-account service can't damage the system. If the service is compromised by a security attack, then the damage is isolated to the operations that the system allows the user account to perform. At the same time, clients running at varying privilege levels can connect to the service, which enables the service to impersonate a client to perform sensitive operations.

A service's user account should not be a member of any administrators groups that are local, domain, or enterprise. If your service needs local administrative privileges, then run it under the `LocalSystem` account. For operations that require domain administrative privileges, perform them by impersonating the security context of a client application.

A service instance that uses a domain user account requires periodic administrative action to maintain the account password. The service control manager (SCM) on the host computer of a service instance caches the account password for use in logging on the service. When you change the account password, you must also update the cached password on the host computer where the service is installed. For more information and a code example, see [Changing the password on a service's user account](#). You could avoid the regular maintenance by leaving the password unchanged, but that would increase the likelihood of a password attack on the service account. Be aware that even though the SCM stores the password in a secure portion of the registry, it is nevertheless subject to attack.

A domain user account has two name formats: the distinguished name of the user object in the directory and the "`<domain>\<username>`" format used by the local service control manager. For more information and a code example that converts from one format to the other, see [Converting domain account name formats](#).

Using the LocalSystem Account as a Service Logon Account

6/3/2022 • 2 minutes to read • [Edit Online](#)

One advantage of running under the LocalSystem account is that the service has complete unrestricted access to local resources. This is also the disadvantage of LocalSystem because a LocalSystem service can do things that would bring down the entire system. In particular, a service running as LocalSystem on a domain controller (DC) has unrestricted access to Active Directory Domain Services. This means that bugs in the service, or security attacks on the service, can damage the system or, if the service is on a DC, damage the entire enterprise network.

For these reasons, domain administrators at sensitive installations will be cautious about allowing services to run as LocalSystem. In fact, they may have policies against it, especially on DCs. If your service must run as LocalSystem, the documentation for your service should justify to domain administrators the reasons for granting the service the right to run at elevated privileges. Services should never run as LocalSystem on a domain controller. For more information and a code example that shows how a service or service installer can determine if it is running on a domain controller, see [Testing Whether Running on a Domain Controller](#).

When a service runs under the LocalSystem account on a computer that is a domain member, the service has whatever network access is granted to the computer account, or to any groups of which the computer account is a member. Be aware that in Windows 2000, a domain computer account is a service principal, similar to a user account. This means that a computer account can be in a security group, and an ACE in a security descriptor can grant access to a computer account. Be aware that adding computer accounts to groups is not recommended for two reasons:

- Computer accounts are subject to deletion and re-creation if the computer leaves and then rejoins the domain.
- If you add a computer account to a group, all services running as LocalSystem on that computer are permitted the access rights of the group. This is because all LocalSystem services share the computer account of their host server. For this reason, it is particularly important that computer accounts not be made members of any domain administrator groups.

Computer accounts typically have few privileges and do not belong to groups. The default ACL protection in Active Directory Domain Services permits minimal access for computer accounts. Consequently, services running as LocalSystem, on computers other than DCs, have only minimal access to Active Directory Domain Services.

If your service runs under LocalSystem, you must test your service on a member server to ensure that your service has sufficient rights to read/write to Active Directory Domain Controllers. A domain controller should not be the only Windows computer on which you test your service. Be aware that a service running under LocalSystem on a Windows domain controller has complete access to Active Directory Domain Services and that a member server runs in the context of the computer account which has substantially fewer rights.

Setting up a Service's User Account

6/3/2022 • 2 minutes to read • [Edit Online](#)

Your service installer can suggest a default logon account for a service instance and allow the administrator to select the default account or specify a different one. If the administrator selects a user account, rather than the LocalSystem account, the account must exist before you call the [CreateService](#) function to install an instance of the service on a host server. For more information and a code example that can be used to create a new domain user object in Active Directory Domain Services, see [Creating a User](#).

Ideally, each instance of a service, whether a host-based or replicable service, should have its own domain user account. Using separate accounts for each service instance is more secure than having multiple instances share the same account. Also, using separate accounts makes it possible to audit the activities of each service instance.

When your installer suggests a default logon account, it should specify the name of a new account to be created for the new service instance. The account name could be composed from the same elements used to compose a service principal name, such as the service class, host computer, and service name. For more information, see [Service Principal Names](#). Typically, you create the account in the Users container on the domain of the host computer.

You must generate a password for each account. For more information about how to write a tool that automates the task of updating service account passwords, see [Changing the Password on a Service's User Account](#).

Installing a Service on a Host Computer

6/3/2022 • 4 minutes to read • [Edit Online](#)

The following code example shows the basic steps of installing a directory-enabled service on a host computer. It performs the following operations:

1. Calls the [OpenSCManager](#) function to open a handle to the service control manager (SCM) on the local computer.
2. Calls the [CreateService](#) function to install the service in the SCM database. This call specifies the service's logon account and password, as well as the service's executable and other information about the service. [CreateService](#) fails if the specified logon account is invalid. However, [CreateService](#) does not check the validity of the password. It also does not verify that the account has the logon as a service right on the local computer. For more information, see [Granting Logon as Service Right on the Host Computer](#).
3. Calls the service's [ScpCreate](#) subroutine that creates a service connection point object (SCP) in the directory to publish the location of this instance of the service. For more information, see [How Clients Find and Use a Service Connection Point](#). This routine also stores the service's binding information in the SCP, sets an ACE on the SCP so the service can access it at run time, caches the distinguished name of the SCP in the local registry, and returns the distinguished name of the new SCP.
4. Calls the service's [SpnCompose](#) subroutine that uses the service's class string and the distinguished name of the SCP to compose a service principal name (SPN). For more information, see [Composing the SPNs for a Service with an SCP](#). The SPN uniquely identifies this instance of the service.
5. Calls the service's [SpnRegister](#) subroutine that registers the SPN on the account object associated with service's logon account. For more information, see [Registering the SPNs for a Service](#). Registration of the SPN enables client applications to authenticate the service.

This code example works correctly regardless of whether the logon account is a local or domain user account or the LocalSystem account. For a domain user account, the *szServiceAccountSAM* parameter contains the *Domain****UserName* name of the account, and the *szServiceAccountDN* parameter contains the distinguished name of the user account object in the directory. For the LocalSystem account, *szServiceAccountSAM* and *szPassword* are **NULL**, and *szServiceAccountSN* is the distinguished name of the local computer's account object in the directory. If *szServiceAccountSAM* specifies a local user account (name format is ".\UserName"), the code example skips the SPN registration because mutual authentication is not supported for local user accounts.

Be aware that the default security configuration allows only domain administrators to execute this code.

Also, be aware that this code example, as written, must be executed on the computer where the service is being installed. Consequently, it is typically in a separate installation executable from your service installation code, if any, that extends the schema, extends the UI, or sets up group policy. Those operations install service components for an entire a forest, whereas this code installs the service on a single computer.

```
void InstallServiceOnLocalComputer(
    LPTSTR szServiceAccountDN, // Distinguished name of logon account.
    LPTSTR szServiceAccountSAM, // SAM name of logon account.
    LPTSTR szPassword)        // Password of logon account.
{
    SC_HANDLE schService = NULL;
    SC_HANDLE schSCManager = NULL;
    TCHAR szPath[512];
    LPTSTR lpFilePart;
    TCHAR szDNofSCP[MAX_PATH];
    TCHAR szServiceClass[] = TEXT("ADSockAuth");
```

```

DWORD dwStatus;
TCHAR **pspn=NULL;
ULONG ulSpn=1;

// Get the full path of the service's executable.
// The code example assumes that the executable is in the current directory.
dwStatus = GetFullPathName(TEXT("service.exe"), 512, szPath, &lpFilePart);
if (dwStatus == 0) {
    _tprintf(TEXT("Unable to install %s - %s\n"),
            TEXT(SZSERVICEDISPLAYNAME), GetLastErrorText(szErr, 256));
    return;
}
_tprintf(TEXT("path of service.exe: %s\n"), szPath);

// Open the Service Control Manager on the local computer.
schSCManager = OpenSCManager(
    NULL,                      // Computer (NULL == local)
    NULL,                      // Database (NULL == default)
    SC_MANAGER_ALL_ACCESS     // Access required
);
if (! schSCManager) {
    _tprintf(TEXT("OpenSCManager failed - %s\n"),
            GetLastErrorText(szErr,256));
    goto cleanup;
}

// Install the service in the SCM database.
schService = CreateService(
    schSCManager,              // SCManager database
    TEXT(SZSERVICENAME),       // Name of service
    TEXT(SZSERVICEDISPLAYNAME), // Name to display
    SERVICE_ALL_ACCESS,        // Desired access
    SERVICE_WIN32_OWN_PROCESS, // Service type
    SERVICE_DEMAND_START,     // Start type
    SERVICE_ERROR_NORMAL,     // Error control type
    szPath,                   // Service binary
    NULL,                     // No load ordering group
    NULL,                     // No tag identifier
    TEXT(SZDEPENDENCIES),     // Dependencies
    szServiceAccountSAM,      // Service account
    szPassword);              // Account password
if (! schService) {
    _tprintf(TEXT("CreateService failed - %s\n"),
            GetLastErrorText(szErr,256));
    goto cleanup;
}

_tprintf(TEXT("%s installed.\n"), TEXT(SZSERVICEDISPLAYNAME) );

// Create the service's Service Connection Point (SCP).
dwStatus = ScpCreate(
    2000,                      // Service default port number
    szServiceClass,             // Specifies the service class string
    szServiceAccountSAM,        // SAM name of logon account for ACE
    szDNofSCP                  // Buffer returns the DN of the SCP
);
if (dwStatus != 0) {
    _tprintf(TEXT("ScpCreate failed: %d\n"), dwStatus );
    DeleteService(schService);
    goto cleanup;
}

// Compose and register a service principal name for this service.
// This is performed on the install path because this requires elevated
// privileges for updating the directory.
// If a local account of the format ".\user name", skip the SPN.
if ( szServiceAccountSAM[0] == '.' )
{

```

```

_tprintf(TEXT("Do not register SPN for a local account.\n"));
goto cleanup;
}

dwStatus = SpnCompose(
    &pspn,           // Receives pointer to the SPN array.
    &ulSpn,          // Receives number of SPNs returned.
    szDNofSCP,       // Input: DN of the SCP.
    szServiceClass); // Input: the service's class string.

if (dwStatus == NO_ERROR)
    dwStatus = SpnRegister(
        szServiceAccountDN, // Account on which SPNs are registered.
        pspn,               // Array of SPNs to register.
        ulSpn,              // Number of SPNs in array.
        DS_SPN_ADD_SPN_OP); // Operation code: Add SPNs.

if (dwStatus != NO_ERROR)
{
    _tprintf(TEXT("Failed to compose SPN: Error was %X\n"),
            dwStatus);
    DeleteService(schService);
    ScpDelete(szDNofSCP, szServiceClass, szServiceAccountDN);
    goto cleanup;
}

cleanup:
if (schSCManager)
    CloseServiceHandle(schSCManager);
if (schService)
    CloseServiceHandle(schService);
DsFreeSpnArray(ulSpn, pspn);
return;
}

```

For more information about the previous code example, see [Composing the SPNs for a Service with an SCP](#) and [Registering the SPNs for a Service](#).

Granting Logon as Service Right on the Host Computer

6/3/2022 • 2 minutes to read • [Edit Online](#)

When installing a service to run under a domain user account, the account must have the right to logon as a service on the local computer. Be aware that this logon right applies only to the local computer and must be granted in the local LSA policy of each host computer. This step is not required if your service runs as LocalSystem, which, by default, is granted this right.

For more information on how to programmatically grant logon as a service right, see the [LSAPrivs sample code](#).

Testing Whether Running on a Domain Controller

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code uses the [VerifyVersionInfo](#) function to determine whether the calling process is running on a Windows 2000 Server domain controller. Your service installation program could use this test before installing a service under the LocalSystem account. If the test indicates that you are running on a domain controller, you either install the service to run under a user account, or display a dialog box warning of the dangers in running as LocalSystem on a domain controller (which are that the service would then have unrestricted access to Active Directory Domain Services, a supremely powerful security context that has the potential to damage the entire network).

```
BOOL Is_Win2000_DomainController ()
{
    OSVERSIONINFOEX osvi;
    DWORDLONG dwlConditionMask = 0;

    // Initialize the OSVERSIONINFOEX structure.
    ZeroMemory(&osvi, sizeof(OSVERSIONINFOEX));
    osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);
    osvi.dwMajorVersion = 5;
    osvi.wProductType = VER_NT_DOMAIN_CONTROLLER;

    // Initialize the condition mask.
    VER_SET_CONDITION( dwlConditionMask, VER_MAJORVERSION,
        VER_GREATER_EQUAL );
    VER_SET_CONDITION( dwlConditionMask, VER_PRODUCT_TYPE,
        VER_EQUAL );

    // Perform the test.
    return VerifyVersionInfo(
        &osvi,
        VER_MAJORVERSION | VER_PRODUCT_TYPE,
        dwlConditionMask);
}
```

Granting Access Rights to the Service Logon Account

6/3/2022 • 2 minutes to read • [Edit Online](#)

A primary consideration of installing a service instance is to ensure that the installed service can access the necessary resources. To do this, set ACEs in the security descriptors of objects that the service must access. An ACE can grant or deny access rights to a specified security principal, such as the service user account, or the computer account for a LocalSystem service, or a group to which the service's account belongs. For more information about ACEs, security descriptors, and access control, see [Controlling Access to objects in Active Directory Domain Services](#) and [Access Control](#).

For more information and a code example that can be used to set an ACE that enables the service to modify its service connection point, see [Enabling Service Account to Access SCP Properties](#).

In some cases, you must add your service user account as a member of one or more security groups. For example, if you create an administrators group for your service, you might make the service a member of the group. You can then grant access rights to the group rather than granting them explicitly to the service account. For more information about security groups, see [Managing Groups](#).

Enabling Service Account to Access SCP Properties

6/3/2022 • 5 minutes to read • [Edit Online](#)

The following code example sets a pair of Access Control Entries (ACEs) on a service connection point (SCP) object. The ACEs grant read/write access to the user or computer account under which the service instance will be running. The service installer uses code similar to the following to ensure that the service can update its properties at run time. If ACEs similar to these are not set, the service will not have access to the properties of the SCP.

Typically, a service installer will set these ACEs after creating the SCP object. For more information, and a code example that creates an SCP and calls this function, see [How Clients Find and Use a Service Connection Point](#). If the service is reconfigured to run under a different account, the ACEs must be updated. To run successfully, this code example must be run in the security context of a domain administrator.

The first parameter of the sample function specifies the name of the user account to be granted access. The function assumes the name is in *Domain****UserName* format. If no account is specified, the function assumes the service uses the LocalSystem account. This means the function must grant access to the computer account of the host server on which the service is running. To do this, the code example calls the [GetComputerObjectName](#) function to obtain the domain and user name of the local computer.

The following code example can be modified to grant the service full access to the SCP object, but the best practice is to grant only the specific access rights that the service requires at run time. In this case, the function grants access to two properties.

PROPERTY	DESCRIPTION
serviceDNSName	The name of the host server on which the service is running.
serviceBindingInformation	Private binding information that the service updates when it starts.

Each property is identified by the **schemaIDGUID** of the property's **attributeSchema** class. Every property in the schema has its own unique **schemaIDGUID**. The following code example uses strings to specify the GUIDs. The GUID strings have the following format where each "X" is replaced by a hexadecimal digit: {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX}.

Refer to the Active Directory Schema reference pages for the **schemaIDGUID** values assigned to the properties to grant or deny access to.

The following code example uses the [IADsSecurityDescriptor](#), [IADsAccessControlList](#), and [IADsAccessControlEntry](#) interfaces to perform the following operations.

1. Obtain the security descriptor of the SCP object.
2. Set the appropriate ACEs in the discretionary access control list (DACL) of the security descriptor.
3. Modify the security descriptor of the SCP object.

```
#include <atldbase.h>

//*****
// AllowAccessToScpProperties()
```

```

// ****
//***** ****

HRESULT AllowAccessToScpProperties(
    LPWSTR wszAccountSAM,    // Service account to allow access.
    IADs *pSCPObject)        // IADs pointer to the SCP object.
{
    HRESULT hr = E_FAIL;
    IADsAccessControlList *pACL = NULL;
    IADsSecurityDescriptor *pSD = NULL;
    IDispatch *pDisp = NULL;
    IADsAccessControlEntry *pACE1 = NULL;
    IADsAccessControlEntry *pACE2 = NULL;
    IDispatch *pDispACE = NULL;
    long lFlags = 0L;
    CComBSTR sbstrTrustee;
    CComBSTR sbstrSecurityDescriptor = L"nTSecurityDescriptor";
    VARIANT varSD;

    if(NULL == pSCPObject)
    {
        return E_INVALIDARG;
    }

    VariantInit(&varSD);

    /*
    If no service account is specified, service runs under
    LocalSystem. Allow access to the computer account of the
    service's host.
    */
    if (wszAccountSAM)
    {
        sbstrTrustee = wszAccountSAM;
    }
    else
    {
        LPWSTR pwszComputerName;
        DWORD dwLen;

        // Get the size required for the SAM account name.
        dwLen = 0;
        GetComputerObjectNameW(NameSamCompatible,
                               NULL, &dwLen);

        pwszComputerName = new WCHAR[dwLen + 1];
        if(NULL == pwszComputerName)
        {
            hr = E_OUTOFMEMORY;
            goto cleanup;
        }

        /*
        Get the SAM account name of the computer object for
        the server.
        */
        if(!GetComputerObjectNameW(NameSamCompatible,
                                   pwszComputerName, &dwLen))
        {
            delete pwszComputerName;

            hr = HRESULT_FROM_WIN32(GetLastError());
            goto cleanup;
        }

        sbstrTrustee = pwszComputerName;
        wprintf(L"GetComputerObjectName: %s\n", pwszComputerName);
        delete pwszComputerName;
    }
}

```

```

        ,

// Get the nTSecurityDescriptor.
hr = pSCPObject->Get(sbstrSecurityDescriptor, &varSD);
if (FAILED(hr) || (varSD.vt != VT_DISPATCH))
{
    _tprintf(TEXT("Get nTSecurityDescriptor failed: 0x%x\n"), hr);
    goto cleanup;
}

/*
Use the V_DISPATCH macro to get the IDispatch pointer from
VARIANT structure and QueryInterface for an
IADsSecurityDescriptor pointer.
*/
hr = V_DISPATCH( &varSD )->QueryInterface(
    IID_IADsSecurityDescriptor,
    (void**)&pSD);
if (FAILED(hr))
{
    _tprintf(
        TEXT("Cannot get IADsSecurityDescriptor: 0x%x\n"),
        hr);
    goto cleanup;
}

/*
Get an IADsAccessControlList pointer to the security
descriptor's DACL.
*/
hr = pSD->get_DiscretionaryAcl(&pDisp);
if (SUCCEEDED(hr))
{
    hr = pDisp->QueryInterface(
        IID_IADsAccessControlList,
        (void**)&pACL);
}
if (FAILED(hr))
{
    _tprintf(TEXT("Cannot get DACL: 0x%x\n"), hr);
    goto cleanup;
}

// Create the COM object for the first ACE.
hr = CoCreateInstance(CLSID_AccessControlEntry,
                      NULL,
                      CLSCTX_INPROC_SERVER,
                      IID_IADsAccessControlEntry,
                      (void **)&pACE1);

// Create the COM object for the second ACE.
if (SUCCEEDED(hr))
{
    hr = CoCreateInstance(CLSID_AccessControlEntry,
                          NULL,
                          CLSCTX_INPROC_SERVER,
                          IID_IADsAccessControlEntry,
                          (void **)&pACE2);
}
if (FAILED(hr))
{
    _tprintf(TEXT("Cannot create ACEs: 0x%x\n"), hr);
    goto cleanup;
}

// Set the properties of the two ACEs.

// Allow read and write access to the property.
hr = pACE1->put_AccessMask(
    (ADS_RIGHT_DS_READ_PROP | ADS_RIGHT_DS_WRITE_PROP));

```

```

    ADS_RIGHT_DS_READ_PROP | ADS_RIGHT_DS_WRITE_PROP);

hr = pACE2->put_AccessMask(
    ADS_RIGHT_DS_READ_PROP | ADS_RIGHT_DS_WRITE_PROP);

// Set the trustee, which is either the service account or the
// host computer account.
hr = pACE1->put_Trustee( sbstrTrustee );
hr = pACE2->put_Trustee( sbstrTrustee );

// Set the ACE type.
hr = pACE1->put_AceType( ADS_ACETYPE_ACCESS_ALLOWED_OBJECT );
hr = pACE2->put_AceType( ADS_ACETYPE_ACCESS_ALLOWED_OBJECT );

// Set AceFlags to zero because ACE is not inheritable.
hr = pACE1->put_AceFlags( 0 );
hr = pACE2->put_AceFlags( 0 );

// Set Flags to indicate an ACE that protects a specified object.
hr = pACE1->put_Flags( ADS_FLAG_OBJECT_TYPE_PRESENT );
hr = pACE2->put_Flags( ADS_FLAG_OBJECT_TYPE_PRESENT );

// Set ObjectType to the schemaIDGUID of the attribute.
// serviceDNSName
hr = pACE1->put_ObjectType(
    L"\\{28630eb8-41d5-11d1-a9c1-0000f80367c1}");
// serviceBindingInformation
hr = pACE2->put_ObjectType(
    L"\\{b7b1311c-b82e-11d0-afee-0000f80367c1}");

/*
Add the ACEs to the DACL. Need an IDispatch pointer for
each ACE to pass to the AddAce method.
*/
hr = pACE1->QueryInterface(IID_IDispatch,(void**)&pDispACE);
if (SUCCEEDED(hr))
{
    hr = pACL->AddAce(pDispACE);
}
if (FAILED(hr))
{
    _tprintf(TEXT("Cannot add first ACE: 0x%x\n"), hr);
    goto cleanup;
}
else
{
    if (pDispACE)
        pDispACE->Release();

    pDispACE = NULL;
}

// Repeat for the second ACE.
hr = pACE2->QueryInterface(IID_IDispatch, (void**)&pDispACE);
if (SUCCEEDED(hr))
{
    hr = pACL->AddAce(pDispACE);
}
if (FAILED(hr))
{
    _tprintf(TEXT("Cannot add second ACE: 0x%x\n"), hr);
    goto cleanup;
}

// Write the modified DACL back to the security descriptor.
hr = pSD->put_DiscretionaryAcl(pDisp);
if (SUCCEEDED(hr))
{
    /*
    Write the ntSecurityDescriptor property to the

```

```
property cache.  
*/  
hr = pSCPObject->Put(sbstrSecurityDescriptor, varSD);  
if (SUCCEEDED(hr))  
{  
    // SetInfo updates the SCP object in the directory.  
    hr = pSCPObject->SetInfo();  
}  
}  
  
cleanup:  
if (pDispACE)  
    pDispACE->Release();  
  
if (pACE1)  
    pACE1->Release();  
  
if (pACE2)  
    pACE2->Release();  
  
if (pACL)  
    pACL->Release();  
  
if (pDisp)  
    pDisp->Release();  
  
if (pSD)  
    pSD->Release();  
  
VariantClear(&varSD);  
  
return hr;  
}
```

Logon Account Maintenance Tasks

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic describes issues that pertain to logon account maintenance tasks.

There are two primary issues that concern logon account maintenance tasks:

- Updating the account password for a service instance that runs under a user account. For more information, see [Changing the Password on a Service's User Account](#).
- Handling changes to the logon account of a service instance.

The latter is a rare case, but can happen. The system provides the Computer Management administrative tool that enables change to a service logon account. In addition, other applications can use the [ChangeServiceConfig](#) function to specify a new logon account for an installed service. By default, local administrator privileges are required to change a service account. If this did happen, it could affect your service in two ways:

- If you have registered service principal names (SPNs), they will be registered on the wrong account.
- If you set ACEs to grant access to the service, they would now grant access to the wrong account.

One approach is to have the service installer store the registered SPNs for each service instance in the registry on the host computer. You could use the same registry key under HKEY_LOCAL_MACHINE that you used to store the binding string for the service SCP. When the service starts, it calls the [QueryServiceConfig](#) function to determine its logon account and then queries the Active Directory server to determine whether the SPNs are registered on the directory object for that account. If the SPNs are not registered, or are registered on the wrong account, the service refuses to start and displays a message saying that a domain administrator must run the service's configuration program to update the logon account settings. Be aware that this reconfiguration must be completed by an administrator because the service account should not have access to update its own SPN. Also be aware that SPNs must be removed from the old account, otherwise the SPNs will be useless for authentication because they are not unique in the forest.

Changing the Password on a Service's User Account

6/3/2022 • 2 minutes to read • [Edit Online](#)

For a service instance that logs on with a user account, rather than the LocalSystem account, the Service Control Manager (SCM) on the host computer stores the account password, which it uses to log on the service when the service starts. As with any user account, you must change the password periodically to maintain security. When you change the password on a service account, update the password stored by the SCM. The following code example shows how to do both.

The code examples use [IADsUser.SetPassword](#) to set the account password. This method uses the distinguished name of the account. Then the sample opens a handle to the installed service on the specified host computer, and uses the [ChangeServiceConfig](#) function to update the password cached by the SCM. This function uses the SAM name ("<domain>\<username>") of the account.

NOTE

This code must be executed by a domain administrator.

For a replicable service in which each replica uses a different logon account, you could update the passwords for all of the replicas by enumerating the service instances. For more information and a code example, see [Enumerating the Replicas of a Service](#).

```
DWORD UpdateAccountPassword(
    LPTSTR szServerDNS,    // DNS name of host computer
    LPTSTR szAccountDN,   // Distinguished name of service
                           // logon account
    LPTSTR szServiceName, // Name of the service
    LPTSTR szOldPassword, // Old password
    LPTSTR szNewPassword // New password
)
{
    SC_HANDLE schService = NULL;
    SC_HANDLE schSCManager = NULL;

    DWORD dwLen = MAX_PATH;
    TCHAR szAccountPath[MAX_PATH];
    IADsUser *pUser = NULL;
    HRESULT hr;

    DWORD dwStatus = 0;
    SC_LOCK sclLock = NULL;

    if( !szServerDNS ||
        !szAccountDN ||
        !szServiceName ||
        !szOldPassword ||
        !szNewPassword)
    {
        _tprintf(TEXT("Invalid parameter"));
        goto cleanup;
    }

    // Set the password on the account.
    // Use the distinguished name to bind to the account object.
    _tcscpy_s(szAccountPath, TEXT("LDAP://"), MAX_PATH);
```

```

_tcscl_s(szAccountPath,
    szAccountDN,
    MAX_PATH - _tcslen(szAccountPath));
hr = ADsGetObject(szAccountPath, IID_IADsUser, (void**)&pUser);
if (FAILED(hr))
{
    _tprintf(TEXT("Get IADsUser failed - 0x%x\n"), dwStatus = hr);
    goto cleanup;
}

// Set the password on the account.
hr = pUser->ChangePassword(CComBSTR(szOldPassword),
    CComBSTR(szNewPassword));
if (FAILED(hr))
{
    _tprintf(TEXT("ChangePassword failed - 0x%x\n"),
        dwStatus = hr);
    goto cleanup;
}

// Update the account and password in the SCM database.
// Open the Service Control Manager on the specified computer.
schSCManager = OpenSCManager(
    szServerDNS,           // DNS name of host computer
    NULL,                 // database (NULL == default)
    SC_MANAGER_ALL_ACCESS // access required
);
if (! schSCManager)
{
    _tprintf(TEXT("OpenSCManager failed - %d\n"),
        dwStatus = GetLastError());
    goto cleanup;
}

// Open a handle to the service instance.
schService = OpenService(schSCManager,
    szServiceName,
    SERVICE_ALL_ACCESS);
if (! schService)
{
    _tprintf(TEXT("OpenService failed - %d\n"),
        dwStatus = GetLastError());
    goto cleanup;
}

// Get the SCM database lock before changing the password.
sclLock = LockServiceDatabase(schSCManager);
if (sclLock == NULL) {
    _tprintf(TEXT("LockServiceDatabase failed - %d\n"),
        dwStatus = GetLastError());
    goto cleanup;
}

// Set the account and password that the service uses at startup.
if (! ChangeServiceConfig(
    schService,           // Handle of service
    SERVICE_NO_CHANGE,   // Service type: no change
    SERVICE_NO_CHANGE,   // Change service start type
    SERVICE_NO_CHANGE,   // Error control: no change
    NULL,                // Binary path: no change
    NULL,                // Load order group: no change
    NULL,                // Tag ID: no change
    NULL,                // Dependencies: no change
    NULL,                // Account name: no change
    szNewPassword,       // New password
    NULL) )              // Display name: no change
{
    _tprintf(TEXT("ChangeServiceConfig failed - %d\n"),
        dwStatus = GetLastError());
}

```

```
    goto cleanup;
}

_tprintf(TEXT("Password changed for service instance on: %s\n"),
szServerDNS);

cleanup:

if (sclLock)
    UnlockServiceDatabase(sclLock);
if (schService)
    CloseServiceHandle(schService);
if (schSCManager)
    CloseServiceHandle(schSCManager);
if (pUser)
    pUser->Release();

return dwStatus;
}
```

Enumerating the Replicas of a Service

6/3/2022 • 3 minutes to read • [Edit Online](#)

This topic includes a code example that enumerates the installed instances of a replicated service on different host computers throughout an enterprise. To change the service account password for each instance of a replicated service, use this code example in conjunction with the code example in the [Changing the Password on a Service's User Account](#) topic.

The code example assumes that each service instance has its own service connection point (SCP) object in the directory. An SCP is an object of the [serviceConnectionPoint](#) class. This class has a **keywords** attribute, which is a multi-valued attribute replicated to each global catalog (GC) in the forest. The **keywords** attribute of each instance's SCP contains the service's product GUID. This enables finding all of the SCPs for the various service instances by searching a GC for objects with a **keywords** attribute that equals the product GUID.

The code example obtains an [IDirectorySearch](#) pointer to a GC, and uses the [IDirectorySearch::ExecuteSearch](#) method to search for the SCPs. Be aware that the GC contains a partial replica for each SCP. That is, it contains some of the SCP attributes, but not all. In this code example, focus on the **serviceDNSName** attribute, which contains the DNS name of the host server for that service instance. Because **serviceDNSName** is not one of the attributes replicated in a GC, the code example uses a two-step process to retrieve it. First, it uses the GC search to get the distinguished name (DN) of the SCP, then it uses that DN to bind directly to the SCP to retrieve the **serviceDNSName** property.

```
HRESULT EnumerateServiceInstances(
    LPWSTR szQuery,           // Search string filter.
    LPWSTR *pszAttribs,       // An array of attributes
                             // to retrieve.
    DWORD dwAttribs,          // # of attributes requested.
    DWORD *pdwAttribs,         // # of attributes retrieved.
    ADS_ATTR_INFO **ppPropEntries // Returns a pointer to the
                                 // retrieved attributes.
)
{
    HRESULT hr;
    IEnumVARIANT *pEnum = NULL;
    IADSContainer *pCont = NULL;
    VARIANT var;
    IDispatch *pDisp = NULL;
    BSTR bstrPath = NULL;
    ULONG lFetch;
    IADS *pADS = NULL;
    int iRows=0;
    static IDirectorySearch *pSearch = NULL;
    static ADS_SEARCH_HANDLE hSearch = NULL;

    // Parameters for IDirectoryObject.
    IDirectoryObject *pSCP = NULL;

    // Structures and parameters for IDirectorySearch.
    DWORD dwPref;
    ADS_SEARCH_COLUMN Col;
    ADS_SEARCHPREF_INFO SearchPref[2];

    // First time through; set up the search.
    if (pSearch == NULL)
    {
        // Bind to the GC: namespace container object. The GC DN
        // is a single immediate child of the GC: namespace, which must
        // be obtained through enumeration.
    }
}
```

```

    hr = ADsGetObject(L"GC:",
                      IID_IADsContainer,
                      (void**) &pCont );
    if (FAILED(hr)) {
        _tprintf(TEXT("ADsGetObject(GC) failed: 0x%x\n"), hr);
        goto Cleanup;
    }

    // Obtain an enumeration interface for the GC container.
    hr = ADsBuildEnumerator(pCont,&pEnum);
    if (FAILED(hr)) {
        _tprintf(TEXT("ADsBuildEnumerator failed: 0x%x\n"), hr);
        goto Cleanup;
    }

    // Enumerate. There is only one child of the GC: object.
    hr = ADsEnumerateNext(pEnum,1,&var,&lFetch);
    if (( hr == S_OK ) && ( lFetch == 1 ) )
    {
        pDisp = V_DISPATCH(&var);
        hr = pDisp->QueryInterface( IID_IADs, (void**)&pADs);
        if (hr == S_OK)
            hr = pADs->get_ADsPath(&bstrPath);
    }
    if (FAILED(hr)) {
        _tprintf(TEXT("Enumeration failed: 0x%x\n"), hr);
        goto Cleanup;
    }

    // Now bstrPath contains the ADsPath for the current GC.
    // Bind the GC to get the search interface.
    hr = ADsGetObject(bstrPath,
                      IID_IDirectorySearch,
                      (void**)&pSearch);
    if (FAILED(hr)) {
        _tprintf(TEXT("Failed to bind search root: 0x%x\n"), hr);
        goto Cleanup;
    }

    // Set up a deep search.
    // Thousands of objects are not expected
    // in this example; request 1000 rows per page.
    dwPref=sizeof(SearchPref)/sizeof(ADS_SEARCHPREF_INFO);
    SearchPref[0].dwSearchPref =     ADS_SEARCHPREF_SEARCH_SCOPE;
    SearchPref[0].vValue.dwType =     ADSTYPE_INTEGER;
    SearchPref[0].vValue.Integer =   ADS_SCOPE_SUBTREE;

    SearchPref[1].dwSearchPref =     ADS_SEARCHPREF_PAGESIZE;
    SearchPref[1].vValue.dwType =     ADSTYPE_INTEGER;
    SearchPref[1].vValue.Integer =   1000;

    hr = pSearch->SetSearchPreference(SearchPref, dwPref);
    if (FAILED(hr)) {
        _tprintf(TEXT("Failed to set search prefs: 0x%x\n"), hr);
        goto Cleanup;
    }

    // Execute the search. From the GC, get the distinguished name
    // of the SCP. Use the DN to bind to the SCP and get the other
    // properties.
    hr = pSearch->ExecuteSearch(szQuery, pszAttribs, 1, &hSearch);
    if (FAILED(hr)) {
        _tprintf(TEXT("ExecuteSearch failed: 0x%x\n"), hr);
        goto Cleanup;
    }

    // Get the next row.
    hr = pSearch->GetNextRow(hSearch);
}

```

```

// Process the row.
if (SUCCEEDED(hr) && hr !=S_ADS_NOMORE_ROWS)
{
    // Get the distinguished name of the object in this row.
    hr = pSearch->GetColumn(hSearch, L"distinguishedName", &Col);
    if FAILED(hr) {
        _tprintf(TEXT("GetColumn failed: 0x%x\n"), hr);
        goto Cleanup;
    }
    // Bind to the DN to get the properties.
    if (Col.dwADSType == ADSTYPE_CASE_IGNORE_STRING)
    {
        LPWSTR szSCPPPath =
            new WCHAR[7 + lstrlenW(Col.pADSValues->CaseIgnoreString) + 1];
        wcscpy_s(szSCPPPath, L"LDAP://");
        wcscat_s(szSCPPPath, Col.pADSValues->CaseIgnoreString);
        hr = ADsGetObject(szSCPPPath,
                           IID_IDirectoryObject,
                           (void**)&pSCP);
        delete szSCPPPath;
        if (SUCCEEDED(hr))
        {
            hr = pSCP->GetObjectAttributes(pszAttrs, dwAttrs,
                                              ppPropEntries, pdwAttrs);
            if(FAILED(hr)) {
                _tprintf(TEXT("GetObjectAttributes Failed."), hr);
                goto Cleanup;
            }
        }
    }
    pSearch->FreeColumn(&Col);}

Cleanup:
if (bstrPath)
    SysFreeString(bstrPath);
if (pSCP)
    pSCP->Release();
if (pCont)
    pCont->Release();
if (pEnum)
    ADsFreeEnumerator(pEnum);
if (pADS)
    pADS->Release();
if (pDisp)
    pDisp->Release();

return hr;
}

```

Converting Domain Account Name Formats

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Microsoft Win32 functions for working with the service control manager (SCM) on a host server use the "<domain>\<username>" format for service accounts. For example, if you call the [QueryServiceConfig](#) function to retrieve the user account under which your service runs, the function returns the name in "Fabrikam\jeffsmith" format. To bind to the user account in the directory, for example to change the account password, the distinguished name of the account is required, which has the format "CN=Jeff Smith,CN=Users,DC=Fabrikam,DC=COM".

To convert between the various name formats, use the [TranslateName](#) function, which can convert a name to other formats, such as a user principal name (UPN).

Mutual Authentication Using Kerberos

6/3/2022 • 2 minutes to read • [Edit Online](#)

Mutual authentication is a security feature in which a client process must prove its identity to a service, and the service must prove its identity to the client, before any application traffic is transmitted over the client/service connection.

Active Directory Domain Services and Windows provide support for service principal names (SPN), which are a key component in the Kerberos mechanism by which a client authenticates a service. An SPN is a unique name that identifies an instance of a service and is associated with the logon account under which the service instance runs. The components of an SPN are such that a client can compose an SPN for a service without the service logon account. This enables the client to request the service to authenticate its account even though the client does not have the account name.

This section includes an overview of:

- Mutual authentication using Kerberos.
- Composing a unique SPN.
- How a service installer registers SPNs on the account object associated with a service instance.
- How a client application uses a service instance's service connection point (SCP) object in Active Directory Domain Services to retrieve data from which to compose an SPN for the service.
- How a client application uses a service SPN in conjunction with the Security Support Provider Interface (SSPI) to authenticate the service.
- A code example for a Windows Sockets client/service application that uses an SCP and SSPI to perform mutual authentication.
- A code example for an RPC client/service that performs mutual authentication using the RPC name service and RPC authentication.
- How a Windows Sockets Registration and Resolution (RnR) service uses SPNs to perform mutual authentication.

This section discusses using Active Directory Domain Service for mutual authentication, in particular, the purpose of service connection points and service principal names in mutual authentication. It is not a complete discussion of how to use SSPI for mutual authentication or the authentication and security support available for RPC and Windows Sockets applications.

For more information, see:

- [Publishing with service connection points](#)
- [SSPI](#)
- [Windows Sockets](#)
- [RPC](#)

About Mutual Authentication Using Kerberos

6/3/2022 • 2 minutes to read • [Edit Online](#)

In mutual authentication, the client and service must verify their respective identities to each other before performing application functions. Neither party can perform operations with the other until identity has been verified; that is, the service must be able to verify the client without querying the client and the client must be able to verify the service without querying the service.

The value of a service that can authenticate a client is well-known. For example, a file service impersonates a client to determine which files the client can access.

The value of a client that can authenticate a service is less understood. Authenticating a service enables the client to trust the data that it gets from the service and to be secure in sending sensitive data to the service. The ability of a client to authenticate a service is particularly important in client/service applications that support delegation of the client's security context; that is, the client authorizes the service to act as its delegate in accessing additional services or network resources.

A service authenticates a client as follows: The client establishes a local security context either by executing in a previously established context, for example, in the session of a logged-in user, or by explicitly presenting credentials to the underlying security provider. The service cannot accept connections from any unauthenticated client.

The Kerberos mechanism by which a client authenticates a service works as follows: When a service is installed, a service installer, running with administrator privileges, registers one or more unique SPNs for each service instance. The names are registered in the Active Directory Domain Controller (DC) on the user or computer account object that the service instance will use to log on. When a client requests a connection to a service, it composes an SPN for a service instance, using known data or data provided by the user. The client then uses the SSPI negotiate package to present the SPN to the Key Distribution Center (KDC) for the client domain account. The KDC searches the forest for a user or computer account on which that SPN is registered. If the SPN is registered on more than one account, the authentication fails. Otherwise, the KDC encrypts a message using the password of the account on which the SPN was registered. The KDC passes this encrypted message to the client, which in turn passes it to the service instance. The service uses the SSPI negotiate package to decrypt the message, which it passes back to the client and on to the client's KDC. The KDC authenticates the service if the decrypted message matches its original message.

- For more information about composing and registering SPNs, see [Service Principal Names](#)
- For more information and a code example that composes an SPN for a Windows Sockets service that publishes itself with a service connection point, see [Composing the SPNs for a Service with an SCP](#).
- For more information and a code example that composes an SPN for an RPC service, see [Composing SPNs for an RpcNs Service](#).

Security Providers

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Security Support Provider Interface (SSPI) provides support for mutual authentication and is exposed directly through the SSPI APIs and services layered on SSPI, including RPC.

Not all security packages available to SSPI support mutual authentication. To obtain mutual authentication, the application must request mutual authentication and a security package that supports it. For example, the code example in [Mutual Authentication in a Windows Sockets Service with an SCP](#) uses the "negotiate" package in Secur32.dll, which is included with Windows 2000.

Integrity and Privacy

6/3/2022 • 2 minutes to read • [Edit Online](#)

Client/service communications that require mutual authentication must protect the traffic they exchange after successful authentication. It is unwise to mutually authenticate at the time of the initial connection to the service if the traffic is later subject to modification by an unauthorized user. SSPI, RPC, and COM all provide utilities for digitally signing and encrypting messages. Applying digital signatures prevents modified traffic from going undetected and discourages eavesdropping. Traffic can be intercepted of course, but decrypting the traffic is sufficiently difficult to deter most unauthorized users.

Unless performance requirements are severe, use of both signing and encryption is recommended, especially for administrative functions. Even when performance is an issue, some customers choose tighter security over better performance. In such cases, make integrity and privacy configurable options with higher security the default and higher performance the option.

RPC clients can specify the level of integrity and privacy when they call the [RpcBindingSetAuthInfoEx](#) function to set the authentication data for the RPC binding. Use [RPC_C_AUTHN_LEVEL_PKT_INTEGRITY](#) for signing and [RPC_C_AUTHN_LEVEL_PKT_PRIVACY](#) for encryption. For more information, see [Mutual Authentication in RPC Applications](#).

Services that use an SSPI package for mutual authentication can query the package to determine whether it supports the [MakeSignature](#), [VerifySignature](#), [EncryptMessage](#), and [DecryptMessage](#) functions for signing and encrypting messages. For more information and a code example, see [Ensuring Communication Integrity During Message Exchange](#).

Limitations of Mutual Authentication with Kerberos

6/3/2022 • 2 minutes to read • [Edit Online](#)

Both the client's account and the service's account must be in Windows 2000 native or mixed-mode domains because Kerberos services are not available in downlevel domains. In addition, both client and service accounts must be in the same forest because the client's KDC uses the global catalog to search for the service principal name.

Both service and client must be running on Windows 2000, otherwise mutual authentication with Kerberos will fail because earlier versions of Windows do not support Kerberos.

Service principal names must include the DNS name of the host server on which the service is running. You must use the DNS name.

Service Principal Names

6/3/2022 • 2 minutes to read • [Edit Online](#)

A service principal name (SPN) is a unique identifier of a service instance. SPNs are used by [Kerberos authentication](#) to associate a service instance with a service logon account. This allows a client application to request that the service authenticate an account even if the client does not have the account name.

If you install multiple instances of a service on computers throughout a forest, each instance must have its own SPN. A given service instance can have multiple SPNs if there are multiple names that clients might use for authentication. For example, an SPN always includes the name of the host computer on which the service instance is running, so a service instance might register an SPN for each name or alias of its host. For more information about SPN format and composing a unique SPN, see [Name Formats for Unique SPNs](#).

Before the Kerberos authentication service can use an SPN to authenticate a service, the SPN must be registered on the account object that the service instance uses to log on. A given SPN can be registered on only one account. For Win32 services, a service installer specifies the logon account when an instance of the service is installed. The installer then composes the SPNs and writes them as a property of the account object in Active Directory Domain Services. If the logon account of a service instance changes, the SPNs must be re-registered under the new account. For more information, see [How a Service Registers its SPNs](#).

When a client wants to connect to a service, it locates an instance of the service, composes an SPN for that instance, connects to the service, and presents the SPN for the service to authenticate. For more information, see [How Clients Compose a Service's SPN](#).

In this section

[Name Formats for Unique SPNs](#)

[How a Service Composes its SPNs](#)

[How a Service Registers its SPNs](#)

[How Clients Compose a Service's SPN](#)

Related topics

[What is an SPN and why should you care?](#)

[Mutual Authentication Using Kerberos](#)

Name Formats for Unique SPNs

6/3/2022 • 3 minutes to read • [Edit Online](#)

An SPN must be unique in the forest in which it is registered. If it is not unique, authentication will fail. The SPN syntax has four elements: two required elements and two additional elements that you can use, if necessary, to produce a unique name as listed in the following table.

<service class>/<host>:<port>/<service name>

ELEMENT	DESCRIPTION
"<service class>"	A string that identifies the general class of service; for example, "SqlServer". There are well-known service class names, such as "www" for a web service or "ldap" for a directory service. In general, this can be any string that is unique to the service class. Be aware that the SPN syntax uses a forward slash (/) to separate elements, so this character cannot appear in a service class name.
"<host>"	The name of the computer on which the service is running. This can be a fully qualified DNS name or a NetBIOS name. Be aware that NetBIOS names are not guaranteed to be unique in a forest, so an SPN that contains a NetBIOS name may not be unique.
"<port>"	An optional port number to differentiate between multiple instances of the same service class on a single host computer. Omit this component if the service uses the default port for its service class.
"<service name>"	An optional name used in the SPNs of a replicable service to identify the data or services provided by the service or the domain served by the service. This component can have one of the following formats: <ul style="list-style-type: none">• The distinguished name or objectGUID of an object in Active Directory Domain Services, such as a service connection point (SCP).• The DNS name of the domain for a service that provides a specified service for a domain as a whole.• The DNS name of an SRV or MX record.

The components present in a service's SPNs depend on how the service is identified and replicated. There are two basic scenarios: host-based services and replicable services.

Host-based services

For a host-based service, the "<service name>" component is omitted because the service is uniquely identified by the service class and the name of the host computer on which the service is installed.

```
<service class>/<host>
```

The service class alone is sufficient to identify for clients the features that the service provides. You can install instances of the service class on many computers and each instance provides services that are identified with its host computer. FTP and Telnet are examples of host-based services. The SPNs of a host-based service instance can include the port number if the service uses a non-default port or there are multiple instances of the service on the host.

```
<service class>/<host>:<port>
```

Replicable services

For a replicable service there can be one or many instances of the service (replicas), and clients do not differentiate which replica they connect to because each provides the same service. The SPNs for each replica have the same "<service class>" and "<service name>" components, where "<service name>" identifies more specifically the features provided by the service. Only the "<host>" and optional "<port>" components would vary from SPN to SPN.

```
<service class>/<host>:<port>/<service name>
```

An example of a replicable service would be an instance of a database service that provides access to a specified database. In this case, "<service class>" identifies the database application and "<service name>" identifies the specific database. "<service name>" could be the distinguished name of a service connection point (SCP) containing connection data for the database.

```
MyDBService/host1.example.com/CN=hrdb,OU=mktg,DC=example,DC=com  
MyDBService/host2.example.com/CN=hrdb,OU=mktg,DC=example,DC=com  
MyDBService/host3.example.com/CN=hrdb,OU=mktg,DC=example,DC=com
```

If clients will use the NetBIOS name to compose a service's SPN, each replica must also register an SPN containing the NetBIOS name.

```
MyDBService/host1/CN=hrdb,OU=mktg,DC=example,DC=com  
MyDBService/host2/CN=hrdb,OU=mktg,DC=example,DC=com  
MyDBService/host3/CN=hrdb,OU=mktg,DC=example,DC=com
```

Another example of a replicable service is one that provides services to an entire domain. In this case, the "<service name>" component is the DNS name of the domain being served. A Kerberos KDC is an example of this type of replicable service.

Be aware that if the DNS name of a computer changes, the system updates the "<host>" element for all registered SPNs for that host in the forest.

How a Service Composes its SPNs

6/3/2022 • 2 minutes to read • [Edit Online](#)

A service can use two functions to compose its SPNs: [DsGetSpn](#) is a general-purpose function for composing SPNs and [DsServerRegisterSpn](#) is a specialized function for composing and registering simple SPNs for a host-based service.

A service installer typically uses the [DsGetSpn](#) function to compose SPNs, which it then registers on the service's logon account using the [DsWriteAccountSpn](#) function. The [DsGetSpn](#) can perform the following functions.

- Create a simple SPN with the "<service class>/<host>" format for a host-based service.
- Create a complex SPN that includes the "<service name>" component used by replicable services or the "<port>" component that distinguishes multiple instances of a service on a single host.
- Create a single SPN with the "<host>" component set to either the name of a specified host or the name of the local computer by default.
- Create an array of SPNs for multiple service instances that will run on multiple hosts throughout the forest. Each SPN specifies the name of the host for a service instance.
- Create an array of SPNs for multiple service instances that will run on the same host. Each SPN specifies the name of the host and a port number for a service instance.

The array of names returned by [DsGetSpn](#) must be freed by calling the [DsFreeSpnArray](#) function.

Be aware that the [DsGetSpn](#), [DsWriteAccountSpn](#), and [DsServerRegisterSpn](#) functions do not verify that SPNs are unique. Because mutual authentication fails if a client presents an SPN that is not unique, verify uniqueness before registering an SPN. To do this, search the global catalog (GC) for **servicePrincipalName** attributes that match your SPN. For more information about searching the GC, see [Searching the Global Catalog](#).

How a Service Registers its SPNs

6/3/2022 • 2 minutes to read • [Edit Online](#)

Before a client can use an SPN to authenticate an instance of a service, the SPN must be registered on the user or computer account that the service instance will use to log on. Typically, SPN registration is done by a service installation program running with domain administrator privileges.

The service installer that installs a service instance on a host computer typically performs the following procedure.

To register SPNs for a service instance

1. Call the [DsGetSpn](#) function to create one or more unique SPNs for the service instance. For more information, see [Name Formats for Unique SPNs](#).
2. Call the [DsWriteAccountSpn](#) function to register the names on the service's logon account.

[DsWriteAccountSpn](#) registers SPNs as a property of a user or computer account object in the directory. **user** and **computer** objects have a **servicePrincipalName** attribute, which is a multi-valued attribute for storing all the SPNs associated with a user or computer account. If the service runs under a user account, the SPNs are stored in the **servicePrincipalName** attribute of that account. If the service runs in the LocalSystem account, the SPNs are stored in the **servicePrincipalName** attribute of the account of the service's host computer. The [DsWriteAccountSpn](#) caller must specify the distinguished name of the account object under which the SPNs are stored.

To ensure that registered SPNs are secure, the **servicePrincipalName** attribute cannot be written directly; it can only be written by calling [DsWriteAccountSpn](#). The caller must have write-access to the **servicePrincipalName** attribute of the target account. Typically, write access is granted by default only to domain administrators. However, there is a special case in which the system allows a service running under the LocalSystem account to register its own SPNs on the computer account of the service's host. In this case, the SPN being written must have the form "<service class>/<host>" and "<host>" must be the DNS name of the local computer.

[DsWriteAccountSpn](#) can also remove SPNs from an account. An operation parameter indicates whether the SPNs are to be added to the account, removed from the account, or used to completely replace all current SPNs for the account. When a service instance is uninstalled, remove any SPNs registered for that instance.

For more information and a code example that registers or unregisters a service's SPNs, see [Registering the SPNs for a Service](#).

Host-based services that use the simple SPN format "<service class>/<host>", have the option of using the [DsServerRegisterSpn](#) function, which both creates and registers SPNs for a service instance.

[DsServerRegisterSpn](#) is a helper function that calls [DsGetSpn](#) and [DsWriteAccountSpn](#).

For more information, see [Service Logon Accounts](#).

How Clients Compose a Service's SPN

6/3/2022 • 2 minutes to read • [Edit Online](#)

To authenticate a service, a client application composes an SPN for the service instance to which it must connect. The client application can use the [DsMakeSpn](#) function to compose an SPN. The client specifies the components of the SPN using known data or data retrieved from sources other than the service itself.

The form of an SPN is as shown in the following form:

```
<service class>/<host>:<port>/<service name>
```

In this form, "<service class>" and "<host>" are required. "<port>" and "<service name>" are optional.

Typically, the client recognizes the "<service class>" part of the name, and recognizes which of the optional components to include in the SPN. The client can retrieve components of the SPN from sources such as a service connection point (SCP) or user input. For example, the client can read the **serviceDNSName** attribute of a service's SCP to get the "<host>" component. The **serviceDNSName** attribute contains either the DNS name of the server on which the service instance is running or the DNS name of SRV records containing the host data for service replicas. The "<service name>" component, used only for replicable services, can be the distinguished name of the service's SCP, the DNS name of the domain served by the service, or the DNS name of SRV or MX records.

For more information and a code example used to compose an SPN for a service, see [How a Client Authenticates an SCP-based Windows Sockets Service](#).

For more information and a description of the SPN components, see [Name Formats for Unique SPNs](#).

Mutual Authentication in a Windows Sockets Service with SCP

6/3/2022 • 2 minutes to read • [Edit Online](#)

The topics in this section include code examples that show how to perform mutual authentication with a service that publishes itself using a service connection point (SCP). The examples are based on a Microsoft Windows Sockets service that uses an SSPI package to handle the mutual authentication negotiation between a client and the service. Use the following procedures to implement mutual authentication within this scenario.

To register SPNs in a directory when a service is installed

1. Call the [DsGetSpn](#) function to compose service principal names (SPNs) for the service.
2. Call the [DsWriteAccountSpn](#) function to register the SPNs on the service account or computer account in whose context the service will run. This step must be performed by a domain administrator; an exception is that a service running under the LocalSystem account can register its SPN in the form "<service class>/<host>" on the computer account of the service host.

To verify configuration at service startup

- Verify that the appropriate SPNs are registered on the account under which the service is running. For more information, see [Logon Account Maintenance Tasks](#).

To authenticate the service at client startup

1. Retrieve connection data from the service's service connection point.
2. Establish a connection to the service.
3. Call the [DsMakeSpn](#) function to compose an SPN for the service. Compose the SPN from the known service class string, and the data retrieved from the service connection point. This data includes the host name of the server on which the service is running. Be aware that the host name must be a DNS name.
4. Use an SSPI security package to perform the authentication:
 - a. Call the [AcquireCredentialsHandle](#) function to acquire the client's credentials.
 - b. Pass the client credentials and the SPN to the [InitializeSecurityContext](#) function to generate a security blob to send to the service for authentication. Set the **ISC_REQ_MUTUAL_AUTH** flag to request mutual authentication.
 - c. Exchange blobs with the service until the authentication is complete.
5. Verify the returned capabilities mask for the **ISC_REQ_MUTUAL_AUTH** flag to verify that mutual authentication was performed.
6. If the authentication was successful, exchange traffic with the authenticated service. Use digital signing to ensure that messages between client and service have not been tampered with. Unless performance requirements are severe, use encryption. For more information and a code example that shows how to use the [MakeSignature](#), [VerifySignature](#), [EncryptMessage](#), and [DecryptMessage](#) functions in an SSPI package, see [Ensuring Communication Integrity During Message Exchange](#).

To authenticate the client by the service when a client connects

1. Load an SSPI security package that supports mutual authentication.
2. When a client connects, use the security package to perform the authentication:
 - a. Call the [AcquireCredentialsHandle](#) function to acquire the service credentials.
 - b. Pass the service credentials and the security blob received from the client to the

[AcceptSecurityContext](#) function to generate a security blob to send back to the client.

- c. Exchange blobs with the client until the authentication is complete.
- 3. Check the returned capabilities mask for the **ASC_RET_MUTUAL_AUTH** flag to verify that mutual authentication was performed.
- 4. If the authentication was successful, exchange traffic with the authenticated client. Use digital signing and encryption unless performance is an issue.

For more information and a code example for this mutual authentication scenario, see:

- [How a Client Authenticates an SCP-based Windows Sockets Service](#)
- [Composing and Registering SPNs for an SCP-based Windows Sockets Service](#)
- [How a Windows Sockets Service Authenticates a Client](#)

For more information, see:

- [Publishing with service connection points](#)
- [SSPI documentation](#)
- [Windows Sockets documentation](#)

How a Client Authenticates an SCP-based Windows Sockets Service

6/3/2022 • 5 minutes to read • [Edit Online](#)

This topic shows the code that a client application uses to compose an SPN for a service. The client binds to the service's service connection point (SCP) to retrieve the data required to connect to the service. The SCP also contains data that the client can use to compose the service's SPN. For more information and a code example that binds to the SCP and retrieves the necessary properties, see [How Clients Find and Use a Service Connection Point](#).

This topic also shows how a client uses an SSPI security package and the service's SPN to establish a mutually authenticated connection to the Windows Sockets service. Be aware that this code is almost identical to the code required in Microsoft Windows NT 4.0 and earlier just to authenticate the client to the server. The only difference is that the client must supply the SPN and specify the ISC_REQ_MUTUAL_AUTH flag.

Client Code to Make an SPN for a Service

```
// Initialize these strings by querying the service's SCP.  
TCHAR szDn[MAX_PATH],      // DN of the service's SCP.  
       szServer[MAX_PATH], // DNS name of the service's server.  
       szClass[MAX_PATH]; // Service class.  
  
TCHAR szSpn[MAX_PATH];      // Buffer for SPN.  
SOCKET sockServer;          // Socket connected to service.  
DWORD dwRes, dwLen;  
.  
.  
.  
// Compose the SPN for the service using the DN, Class, and Server  
// returned by ScpLocate.  
dwLen = sizeof(szSpn);  
dwRes = DsMakeSpn(  
    szClass,      // Service class.  
    szDn,        // DN of the service's SCP.  
    szServer,    // DNS name of the server on which service is running.  
    0,           // No port component in SPN.  
    NULL,        // No referrer.  
    &dwLen,      // Size of szSpn buffer.  
    szSpn);     // Buffer to receive the SPN.  
  
if (!DoAuthentication (sockServer, szSpn)) {  
    closesocket (sockServer);  
    return(FALSE);  
}  
.  
.  
.
```

Client Code to Authenticate the Service

This code example consists of two routines: **DoAuthentication** and **GenClientContext**. After calling **DsMakeSpn** to compose an SPN for the service, the client passes the SPN to the **DoAuthentication** routine, which calls the **GenClientContext** to generate the initial buffer to send to the service. **DoAuthentication** uses the socket handle to send the buffer and receive the service's response passed to the SSPI package by another

call to **GenClientContext**. This loop is repeated until the authentication fails or **GenClientContext** sets a flag that indicates the authentication succeeded.

The **GenClientContext** routine interacts with the SSPI package to generate the authentication data to send to the service and process the data received from the service. The key components of the authentication data provided by the client include:

- The service principal name which identifies the credentials that the service must authenticate.
- The client's credentials. The **AcquireCredentialsHandle** function of the security package extracts these credentials from the client's security context established at logon.
- To request mutual authentication, the client must specify the ISC_REQ_MUTUAL_AUTH flag when it calls the **InitializeSecurityContext** function during the **GenClientContext** routine.

```
// Structure for storing the state of the authentication sequence.
typedef struct _AUTH_SEQ
{
    BOOL _fNewConversation;
    CredHandle _hcred;
    BOOL _fHaveCredHandle;
    BOOL _fHaveCtxHandle;
    struct _SecHandle _hctxt;
} AUTH_SEQ, *PAUTH_SEQ;

//********************************************************************/
// DoAuthentication routine for the client.
//
// Manages the client's authentication conversation with the service
// using the supplied socket handle.
//
// Returns TRUE if the mutual authentication is successful.
// Otherwise, it returns FALSE.
//
//********************************************************************/
BOOL DoAuthentication (
    SOCKET s,
    LPTSTR szSpn)
{
    BOOL done = FALSE;
    DWORD cbOut, cbIn;

    // Call the security package to generate the initial buffer of
    // authentication data to send to the service.
    cbOut = g_cbMaxMessage;
    if (!GenClientContext (s, NULL, 0, g_pOutBuf,
                          &cbOut, &done, szSpn))
        return(FALSE);

    if (!SendMsg (s, g_pOutBuf, cbOut))
        return(FALSE);

    // Pass the service's response back to the security package, and send
    // the package's output back to the service. Repeat until complete.
    while (!done)
    {
        if (!ReceiveMsg (s, g_pInBuf, g_cbMaxMessage, &cbIn))
            return(FALSE);

        cbOut = g_cbMaxMessage;
        if (!GenClientContext (s, g_pInBuf, cbIn, g_pOutBuf,
                              &cbOut, &done, szSpn))
            return(FALSE);

        if (!SendMsg (s, g_pOutBuf, cbOut))
            return(FALSE);
    }
}
```

```

    return(TRUE);
}

//*****************************************************************************
//  GenClientContext routine
//
// Handles the client's interactions with the security package.
// Optionally takes an input buffer coming from the service
// and generates a buffer of data to send back to the
// service. Also returns an indication when the authentication
// is complete.
//
// Returns TRUE if the mutual authentication is successful.
// Otherwise, it returns FALSE.
//
//*************************************************************************/
BOOL GenClientContext (
    DWORD dwKey,      // Socket handle used as key.
    BYTE *pIn,
    DWORD cbIn,
    BYTE *pOut,
    DWORD *pcbOut,
    BOOL *pfDone,
    LPTSTR szSpn)
{
SECURITY_STATUS ssStatus;
TimeStamp Lifetime;
SecBufferDesc OutBuffDesc;
SecBuffer OutSecBuff;
SecBufferDesc InBuffDesc;
SecBuffer InSecBuff;
ULONG ContextAttributes;
PAUTH_SEQ pAS; // Structure to store state of authentication.

// Get structure that contains the state of the authentication sequence.
if (!GetEntry (dwKey, (PVOID*) &pAS))
    return(FALSE);

if (pAS->_fNewConversation)
{
    ssStatus = g_pFuncs->AcquireCredentialsHandle (
        NULL,      // Principal
        PACKAGE_NAME,
        SECPKG_CRED_OUTBOUND,
        NULL,      // LOGON id
        NULL,      // Authentication data
        NULL,      // Get key function.
        NULL,      // Get key argument.
        &pAS->_hcred,
        &Lifetime
    );
    if (SEC_SUCCESS (ssStatus))
        pAS->_fHaveCredHandle = TRUE;
    else
    {
        fprintf (stderr,
            "AcquireCredentialsHandle failed: %u\n", ssStatus);
        return(FALSE);
    }
}

// Prepare output buffer.
OutBuffDesc.ulVersion = 0;
OutBuffDesc.cBuffers = 1;
OutBuffDesc.pBuffers = &OutSecBuff;

OutSecBuff.cbBuffer = *pcbOut;
OutSecBuff.BufferType = SECBUFFER_TOKEN;

```

```

OutSecBuff.pvBuffer = pOut;

// Prepare input buffer.
if (!pAS->_fNewConversation)
{
    InBuffDesc.ulVersion = 0;
    InBuffDesc.cBuffers = 1;
    InBuffDesc.pBuffers = &InSecBuff;

    InSecBuff.cbBuffer = cbIn;
    InSecBuff.BufferType = SECBUFFER_TOKEN;
    InSecBuff.pvBuffer = pIn;
}

_tprintf(TEXT("InitializeSecurityContext: pszTarget=%s\n"), szSpn);

ssStatus = g_pFuncs->InitializeSecurityContext (
    &pAS->_hcred,
    pAS->_fNewConversation ? NULL : &pAS->_hctxt,
    szSpn,
    ISC_REQ_MUTUAL_AUTH,           // Context requirements
    0,                            // Reserved1
    SECURITY_NATIVE_DREP,
    pAS->_fNewConversation ? NULL : &InBuffDesc,
    0,                            // Reserved2
    &pAS->_hctxt,
    &OutBuffDesc,
    &ContextAttributes,
    &Lifetime
);
if (!SEC_SUCCESS (ssStatus))
{
    fprintf (stderr, "init context failed: %X\n", ssStatus);
    return FALSE;
}

pAS->_fHaveCtxtHandle = TRUE;

// Complete token if applicable.
if ((SEC_I_COMPLETE_NEEDED == ssStatus) ||
    (SEC_I_COMPLETE_AND_CONTINUE == ssStatus))
{
    if (g_pFuncs->CompleteAuthToken)
    {
        ssStatus = g_pFuncs->CompleteAuthToken (&pAS->_hctxt,
                                                &OutBuffDesc);
        if (!SEC_SUCCESS(ssStatus))
        {
            fprintf (stderr, "complete failed: %u\n", ssStatus);
            return FALSE;
        }
    } else
    {
        fprintf (stderr, "Complete not supported.\n");
        return FALSE;
    }
}
*pcbOut = OutSecBuff.cbBuffer;

if (pAS->_fNewConversation)
    pAS->_fNewConversation = FALSE;

*pfDone = !((SEC_I_CONTINUE_NEEDED == ssStatus) ||
            (SEC_I_COMPLETE_AND_CONTINUE == ssStatus));

// Check the ISC_RET_MUTUAL_AUTH flag to verify that
// mutual authentication was performed.
if (*pfDone && !(ContextAttributes && ISC_RET_MUTUAL_AUTH) )

```

```
    _tprintf(TEXT("Mutual Auth not set in returned context.\n"));

return TRUE;
}
```

Composing and registering SPNs for SCP-based Windows Sockets Service

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example shows how to compose and register the SPNs for a service. Call this code from your service installer after calling [CreateService](#) and creating the service's service connection point (SCP).

The following code example calls the **SpnCompose** and **SpnRegister** example functions that compose and register the SPN. For more information and the **SpnCompose** source code, see [Composing the SPNs for a Service with an SCP](#). For more information and the **SpnRegister** source code, see [Registering the SPNs for a Service](#).

This example uses the service class name and the distinguished name of its SCP to create its service principal name. For more information and a code example that shows how the client binds to the service SCP to retrieve these name strings, see [How Clients Find and Use a Service Connection Point](#). Be aware that the code for composing an SPN varies depending on the type of service and the mechanisms used to publish the service.

The service registers its SPN by storing it in the **servicePrincipalName** attribute of the service's account object in the directory. If the service runs under the LocalSystem account instead of under a service account, it registers its SPN under the local computer account's object in the directory.

```
TCHAR szDNofSCP[MAX_PATH]; // DN of SCP. Initialize by querying SCP.  
TCHAR szServiceClass[] = TEXT("ADSockAuth");  
LPCTSTR szServiceAccountDN; // DN of a service logon account.  
  
DWORD dwStatus;  
TCHAR **pspn = NULL;  
ULONG ulSpn = 1;  
  
// Compose the SPNs.  
dwStatus = SpnCompose(  
    &pspn, // Receives pointer to the SPN array.  
    &ulSpn, // Receives number of SPNs returned.  
    szDNofSCP, // Input: DN of the SCP.  
    szServiceClass); // Input: the service class string.  
  
// Register the SPNs  
if (dwStatus == NO_ERROR)  
    dwStatus = SpnRegister(  
        szServiceAccountDN, // Logon account to register SPNs on  
        psxn, // Array of SPNs  
        ulSpn, // Number of SPNs in array  
        DS_SPN_ADD_SPN_OP); // Add SPNs to the account  
  
// Free the array of SPNs returned by SpnCompose.  
DsFreeSpnArray(ulSpn, psxn);
```

You can use similar code to unregister your SPNs when your service is uninstalled. Specify the **DS_SPN_DELETE_SPN_OP** operation instead of **DS_SPN_ADD_SPN_OP**.

Composing the SPNs for a Service with an SCP

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example composes an SPN for a service that uses a service connection point (SCP). The returned SPN has the following format.

```
<service class>/<host>/<service name>
```

"<service class>" and "<service name>" correspond to the *pszDNofSCP* and *pszServiceClass* parameters. "
<host>" defaults to the DNS name of the local computer.

```
DWORD
SpnCompose(
    TCHAR ***ppsn,           // Output: an array of SPNs
    unsigned long *pulSpn,   // Output: the number of SPNs returned
    TCHAR *pszDNofSCP,      // Input: DN of the service's SCP
    TCHAR* pszServiceClass) // Input: the name of the service class
{
    DWORD dwStatus;

    dwStatus = DsGetSpn(
        DS_SPN_SERVICE, // Type of SPN to create (enumerated type)
        pszServiceClass, // Service class - a name in this case
        pszDNofSCP, // Service name - DN of the service SCP
        0, // Default: omit port component of SPN
        0, // Number of entries in hostnames and ports arrays
        NULL, // Array of hostnames. Default is local computer
        NULL, // Array of ports. Default omits port component
        pulSpn, // Receives number of SPNs returned in array
        pspn // Receives array of SPN(s)
    );

    return dwStatus;
}
```

Registering the SPNs for a Service

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example registers or unregisters one or more service principal names (SPNs) for a service instance.

The example calls the **DsWriteAccountSpn** function, which stores the SPNs in Active Directory Domain Services under the **servicePrincipalName** attribute of the account object specified by the *pszServiceAcctDN* parameter. The account object corresponds to the logon account specified in the **CreateService** call for this service instance. If the logon account is a domain user account, *pszServiceAcctDN* must be the distinguished name of the account object in Active Directory Domain Servers for that user account. If the service's logon account is the LocalSystem account, *pszServiceAcctDN* must be the distinguished name of the computer account object for the host computer on which the service is installed.

```
*****
SpnRegister()

Register or unregister the SPNs under the service's account.

If the service runs in LocalSystem account, pszServiceAcctDN is the
distinguished name of the local computer account.

Parameters:

pszServiceAcctDN - Contains the distinguished name of the logon
account for this instance of the service.

pspn - Contains an array of SPNs to register.

ulSpn - Contains the number of SPNs in the array.

Operation - Contains one of the DS_SPN_WRITE_OP values that determines
the type of operation to perform on the SPNs.

*****
DWORD SpnRegister(TCHAR *pszServiceAcctDN,
                  TCHAR **pspn,
                  unsigned long ulSpn,
                  DS_SPN_WRITE_OP Operation)
{
    DWORD dwStatus;
    HANDLE hDs;
    TCHAR szSamName[512];
    DWORD dwSize = sizeof(szSamName) / sizeof(szSamName[0]);
    PDOMAIN_CONTROLLER_INFO pDcInfo;

    // Bind to a domain controller.
    // Get the domain for the current user.
    if(GetUserNameEx(NameSamCompatible, szSamName, &dwSize))
    {
        TCHAR *pWhack = _tcschr(szSamName, '\\');
        if(pWhack)
        {
            *pWhack = '\\0';
        }
    }
    else
    {
```

```
    return GetLastError();
}

// Get the name of a domain controller in that domain.
dwStatus = DsGetDcName(NULL,
    szSamName,
    NULL,
    NULL,
    DS_IS_FLAT_NAME |
        DS_RETURN_DNS_NAME |
        DS_DIRECTORY_SERVICE_REQUIRED,
    &pDcInfo);
if(dwStatus != 0)
{
    return dwStatus;
}

// Bind to the domain controller.
dwStatus = DsBind(pDcInfo->DomainControllerName, NULL, &hDs);

// Free the DOMAIN_CONTROLLER_INFO buffer.
NetApiBufferFree(pDcInfo);
if(dwStatus != 0)
{
    return dwStatus;
}

// Write the SPNs to the service account or computer account.
dwStatus = DsWriteAccountSpn(
    hDs,                      // Handle to the directory.
    Operation,                // Add or remove SPN from account's existing SPNs.
    pszServiceAcctDN,         // DN of service account or computer account.
    ulSpn,                    // Number of SPNs to add.
    (const TCHAR ** )pspn);   // Array of SPNs.

// Unbind the DS in any case.
DsUnBind(&hDs);

return dwStatus;
}
```

How a Windows Sockets Service Authenticates a Client

6/3/2022 • 3 minutes to read • [Edit Online](#)

When a client connects to the Windows Sockets service, the service begins its operations for the mutual authentication sequence, which is shown in the following code examples.

The **DoAuthentication** routine uses the socket handle to receive the first authentication packet from the client. The client buffer is passed to the **GenServerContext** function, which then passes the buffer to the SSPI security package for authentication. **DoAuthentication** then sends the security package output back to the client. This loop is repeated until the authentication fails or **GenServerContext** sets a flag indicating the authentication succeeded.

GenServerContext calls the following functions from an SSPI security package:

- **AcquireCredentialsHandle** extracts the service credentials from the service security context that was established when the service started.
- **AcceptSecurityContext** attempts to perform the mutual authentication using the service credentials and the authentication data received from the client. To request mutual authentication, the **AcceptSecurityContext** call must specify the ASC_REQ_MUTUAL_AUTH flag.
- **CompleteAuthToken** is called, if necessary, to complete the authentication operation.

The following code example uses the negotiate package from the Secur32.dll library of security packages.

```
*****  
//  DoAuthentication routine for the service  
//  
//  Manages the service authentication communication with the client  
//  using the supplied socket handle.  
//  
//  Returns TRUE if the mutual authentication succeeds.  
//  Otherwise, it returns FALSE.  
//  
*****  
  
BOOL DoAuthentication (SOCKET s)  
{  
    DWORD cbIn, cbOut;  
    BOOL done = FALSE;  
  
    if(s==INVALID_SOCKET)  
    {  
        return(FALSE);  
    }  
  
    // Receive authentication data from the client and pass  
    // it to the security package. Send the package output back  
    // to the client. Repeat until complete.  
    do  
    {  
        if (!ReceiveMsg (s, g_pInBuf, g_cbMaxMessage, &cbIn))  
            return(FALSE);  
  
        cbOut = g_cbMaxMessage;  
        if (!GenServerContext (s, g_pInBuf, cbIn, g_pOutBuf,  
                             &cbOut, &done))  
            return(FALSE);  
    } while (!done);  
}
```

```

        if (!SendMsg (s, g_pOutBuf, cbOut))
            return(FALSE);
    }
    while(!done);

    return(TRUE);
}

//*****************************************************************************
//  GenServerContext routine
//
// Handles the service interactions with the security package.
// Takes an input buffer coming from the client and generates a
// buffer of data to send back to the client. Also returns
// an indication when the authentication is complete.
//
// Returns TRUE if the mutual authentication is successful.
// Otherwise, it returns FALSE.
//
//*****************************************************************************

BOOL GenServerContext (
    DWORD dwKey,
    BYTE *pIn,
    DWORD cbIn,
    BYTE *pOut,
    DWORD *pcbOut,
    BOOL *pfDone)
{
    SECURITY_STATUS ssStatus;
    TimeStamp Lifetime;
    SecBufferDesc OutBuffDesc;
    SecBuffer OutSecBuff;
    SecBufferDesc InBuffDesc;
    SecBuffer InSecBuff;
    ULONG ContextAttributes = 0;
    PAUTH_SEQ pAS = null;

    if((pIn==NULL && cbIn>0) || (pOut==NULL) || (pcbOut==NULL) || (pfDone==NULL))
    {
        return(FALSE);
    }

    // Get a structure that contains the state of the authentication sequence.
    if (!GetEntry (dwKey, (PVOID*) &pAS))
        return(FALSE);

    if (pAS->_fNewConversation)
    {
        ssStatus = g_pFuncs->AcquireCredentialsHandle (
            NULL,      // principal
            PACKAGE_NAME,
            SECPKG_CRED_INBOUND,
            NULL,      // LOGON id
            NULL,      // authentication data
            NULL,      // get key function
            NULL,      // get key argument
            &pAS->_hcRed,
            &Lifetime
        );
        if (SEC_SUCCESS (ssStatus))
            pAS->_fHaveCredHandle = TRUE;
        else
        {
            fprintf (stderr, "AcquireCredentialsHandle failed: %u\n",
                    ssStatus);
            return(FALSE);
        }
    }
}

```

```

// Prepare the output buffer.
OutBuffDesc.ulVersion = 0;
OutBuffDesc.cBuffers = 1;
OutBuffDesc.pBuffers = &OutSecBuff;

OutSecBuff.cbBuffer = *pcbOut;
OutSecBuff.BufferType = SECBUFFER_TOKEN;
OutSecBuff.pvBuffer = pOut;

// Prepare the input buffer.
InBuffDesc.ulVersion = 0;
InBuffDesc.cBuffers = 1;
InBuffDesc.pBuffers = &InSecBuff;

InSecBuff.cbBuffer = cbIn;
InSecBuff.BufferType = SECBUFFER_TOKEN;
InSecBuff.pvBuffer = pIn;

ssStatus = g_pFuncs->AcceptSecurityContext (
    &pAS->_hcred,
    pAS->_fNewConversation ? NULL : &pAS->_hctxt,
    &InBuffDesc,
    ASC_REQ_MUTUAL_AUTH, // Context requirements.
    SECURITY_NATIVE_DREP,
    &pAS->_hctxt,
    &OutBuffDesc,
    &ContextAttributes,
    &Lifetime
);
if (!SEC_SUCCESS (ssStatus))
{
    fprintf (stderr, "AcceptSecurityContext failed: %u\n", ssStatus);
    return FALSE;
}
if (!(ContextAttributes && ASC_RET_MUTUAL_AUTH))
    _tprintf(TEXT("Mutual Auth not set in returned context.\n"));

pAS->_fHaveCtxtHandle = TRUE;

// Complete the authentication token, if necessary.
if ((SEC_I_COMPLETE_NEEDED == ssStatus) ||
    (SEC_I_COMPLETE_AND_CONTINUE == ssStatus))
{
    if (g_pFuncs->CompleteAuthToken)
    {
        ssStatus = g_pFuncs->CompleteAuthToken (&pAS->_hctxt,
                                                &OutBuffDesc);
        if (!SEC_SUCCESS(ssStatus))
        {
            fprintf (stderr, "complete failed: %u\n", ssStatus);
            return FALSE;
        }
    } else
    {
        fprintf (stderr, "Complete not supported.\n");
        return FALSE;
    }
}

*pcbOut = OutSecBuff.cbBuffer;

if (pAS->_fNewConversation)
    pAS->_fNewConversation = FALSE;

*pfDone = !((SEC_I_CONTINUE_NEEDED == ssStatus) ||
            (SEC_I_COMPLETE_AND_CONTINUE == ssStatus));

return TRUE;

```

```
    }  
}
```

Mutual Authentication in RPC Applications

6/3/2022 • 2 minutes to read • [Edit Online](#)

RPC services can use service connection points to publish themselves, or they can use the RPC name service (RpcNs) APIs. This topic discusses how to perform mutual authentication with an RPC service that publishes itself using the RPC name service (RpcNs) APIs.

To register an SPN in the directory

1. Call the [DsGetSpn](#) function to compose a service principal name (SPN) for the service.
2. Call the [DsWriteAccountSpn](#) function to register the SPN on the service account or computer account in whose context the service will run.

To register a service with the RPC naming service

1. Verify that the appropriate SPNs are registered on the account under which the service is running. For more information, see [Logon Account Maintenance Tasks](#).
2. Call the [RpcServerRegisterAuthInfo](#) function to register the service SPN with the RPC authentication service, and specify **RPC_C_AUTHN_GSS_NEGOTIATE** as the authentication service to use.

For more information about performing mutual authentication in an RPC service, see [Writing an Authenticated SSPI Server](#).

To authenticate the service from the client

1. Extract the host name from the RPC Binding.
2. Compose the SPN for the service by calling the [DsMakeSpn](#) function with the service class, the DNS host name, and the service name; that is the distinguished name of the connection point in the case of RpcNs.
3. Set up an [RPC_SECURITY_QOS](#) structure to request mutual authentication.
4. Call the [RpcBindingSetAuthInfoEx](#) function to set the authentication data for the RPC binding. The client must request at least **RPC_C_AUTHN_LEVEL_PKT_INTEGRITY** to ensure that communications have not been tampered. For increased security, the client should specify **RPC_C_AUTHN_LEVEL_PKT_PRIVACY** to request encryption.
5. Perform the RPC call.

For more information about performing mutual authentication in an RPC client, see [Writing an Authenticated SSPI Client](#).

To authenticate the client from the service

1. Call the [RpcBindingInqAuthClient](#) function to verify the authentication parameters specified by the client. If the client has not requested the desired level of authentication, reject the call. Be aware that an RPC service must verify the authentication level, authentication service, and client identity on every call to ensure that the client has been properly authenticated.
2. Call the [RpclImpersonateClient](#) function to impersonate the client.
3. Perform the requested operation.

4. Call the [RpcRevertToSelf](#) function to revert to the service security context.

For more information about RPC client impersonation, see [Client Impersonation](#).

For more information, see:

- [Publishing with the RPC Name Service \(RpcNs\)](#)
- [RPC Security Essentials](#)

Composing SPNs for an RpcNs Service

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code example composes the service principal names (SPNs) for an RPC service that has an entry in the RpcServices container in the directory. An RPC service uses the [RpcNsBindingExport](#) function to create its RpcServices entry.

An RPC service uses this code example to build the SPN or SPNs that identify an instance of the service. The service uses this routine to perform the following tasks:

- To register or unregister the SPNs in the directory, when the service is installed or removed. For more information and a code example, see [Registering the SPNs for a Service](#).
 - Register itself with the RPC authentication service when the service starts. For more information, see [Mutual Authentication in RPC Applications](#).

This code example uses the distinguished name of the service's RpcServices entry to compose the SPN. Before calling this code, call the [RpcNsBindingExport](#) function to create the service's RpcServices entry.

This code example calls the [DsGetSpn](#) function to build an SPN. The SPN is composed from service class name and the distinguished name of the service RpcServices entry.

```
*****
SpnCompose()

Composes a service principal name from a service name and class.

Parameters:

pszServiceName - Contains a null-terminated string that contains
the name of the service.

pszServiceClass - Contains a null-terminated string that contains
the name of the service class.

pbyn - Pointer to an LPTSTR array that receives the array of
SPNs. This array must freed with the DsFreeSpnArray function when
it is no longer required.

pulSpn - Pointer to a unsigned long that receives the number of
elements in the pbyn array.

*****/
HRESULT SpnCompose(LPTSTR pszServiceName,
                    LPTSTR pszServiceClass,
                    LPTSTR **pbyn,
                    unsigned long *pulSpn)
{
    HRESULT hr;
    CComPtr<IADS> spRoot;

    // Get the defaultNamingContext for the local domain.
    hr = ADsGetObject(L"LDAP://RootDSE",
                      IID_IADS,
                      (void**)&spRoot);

    if(FAILED(hr))
    {
        return hr;
    }

    if(pulSpn)
    {
        *pulSpn = 0;
    }
}
```

```
}

// Get the distinguished name of the current domain.
CComVariant svarDSRoot;
hr = spRoot->Get(CComBSTR("defaultNamingContext"), &svarDSRoot);

/*
Compose the DN of the service's entry in the RpcServices
container, created by a call to RpcNsBindingExport. The entry
for an RPC service is in the System/RpcServices container in the
defaultNamingContext of the local domain.
*/
CComBSTR sbstrDsEntryName = "CN=";
sbstrDsEntryName += pszServiceName;
sbstrDsEntryName += ",CN=RpcServices,CN=System,";
sbstrDsEntryName += svarDSRoot.bstrVal;

USES_CONVERSION; // Required for the W2T() macro.
DWORD status;

/*
Build the SPN for this service using the DN and the service
class.
*/
status = DsGetSpn(
    DS_SPN_SERVICE, // Type of SPN to create.
    pszServiceClass, // Service class - a name in this case.
    W2T(sbstrDsEntryName), // DN of the RpcServices for
                           // this RPC service.
    0, // Use the default instance port.
    0, // Number of additional instance names.
    NULL, // No additional instance names.
    NULL, // No additional instance ports.
    pulSpn, // Size of SPN array.
    pspn // Returned SPN(s).
);

return HRESULT_FROM_WIN32(status);
}
```

Mutual Authentication in Windows Sockets Applications

6/3/2022 • 2 minutes to read • [Edit Online](#)

Microsoft Windows Sockets services can use the Registration and Resolution (RnR) APIs to publish services, or they can use service connection points.

For more information and a code example that shows how to perform mutual authentication for a Windows Sockets service that publishes using a service connection point, see [Mutual Authentication in a Windows Sockets Service with an SCP](#). This code example uses an SSPI security package to manage the authentication negotiations between a client and the WinSock service.

A WinSock RnR service can use similar code to perform mutual authentication using an SSPI package. In this case, the service would compose its SPNs using the distinguished name of the service's entry in the WinsockServices container in the directory.

For example, if the service registers itself with the name "WinSockRnRSampleService", you would compose the service's SPN from the following distinguished name:

```
cn=WinSockRnRSampleService,cn=WinsockServices,cn=System,<domain DN>
```

Backing Up and Restoring an Active Directory Server

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services provide functions for backing up and restoring data in the directory database. This section describes how to [back up](#) and [restore](#) an Active Directory server. For more information about backing up an Active Directory server using the utilities provided in Windows 2000 and Windows Server 2003 operating systems, see the applicable Resource Kit, available on the Microsoft TechNet website.

Backup of an Active Directory server must be performed online and must be performed when the Active Directory Domain Services are installed. Active Directory Domain Services are built on a special database and export a set of backup functions that provide the programmatic backup interface. The backup does not support incremental backups. A backup application binds to a local client-side DLL with entry points defined in Ntdsbcli.h.

Restoration of an Active Directory server is always performed offline.

Although the topics in this section describe only how to back up and restore an Active Directory server, be aware that Windows 2000 and the Windows Server 2003 operating systems have several "system state" components that must be backed up and restored together. These system state components consist of:

- Boot files such as ntldr, ntdetect, all files protected by SFP, and performance counter configuration
- The Active Directory Domain Controller
- SysVol (domain controller only)
- Certificate Server (CA only)
- Cluster database (cluster node only)
- Registry
- COM+ class registration database

The system state can be backed up in any order, but restoration of the system state must occur in the following order:

1. Restore the boot files.
2. Restore SysVol, Certificate Server, Cluster database and COM+ class registration database, as applicable.
3. Restore the Active Directory server.
4. Restore the registry.

For more information about backing up and restoring Certificate Services, see [Using the Certificate Services Backup and Restore Functions](#).

For more information about backing up and restoring Active Directory Domain Services, see:

- [Considerations for Active Directory Domain Services Backup](#)
- [Backing Up an Active Directory Server](#)
- [Restoring an Active Directory Server](#)
- [Directory Backup Functions](#)

Considerations for Active Directory Domain Services Backup

6/3/2022 • 2 minutes to read • [Edit Online](#)

Directory service data can be replicated. A recovery plan must be created prior to restoration. One option is to restore a replica of a directory and then propagate changes that occurred since the backup from other replicas in the domain.

In some cases you may want the restored replica to take precedence over the other replicas in the domain. For example, if an object is accidentally deleted and the deletion is replicated to all domain controllers, you could restore the object by restoring one replica from a backup that was made before the object was deleted. Then you would use the NTDSUtil utility to mark the restored object as authoritatively restored. The restored object will then be replicated to the other DCs, and the replica that was restored will receive the updates for all other objects that occurred since the time the backup was made. The end result for all the replicas is the same as that prior to the restore, except that the authoritatively restored object is no longer deleted.

All changes occurring during backup are stored in a temporary log and added to the end of the backup set when the backup is complete.

Any recovery plan should ensure that the age of the backup should not exceed the Active Directory Tombstone Lifetime. For more information on Tombstone Lifetime, see the TechNet topic - [Introduction to Administering Active Directory Backup and Recovery](#). Restoration of a backup older than the tombstone lifetime may cause the restored domain controller to have objects that will not be replicated on other DCs. This occurs if an object is deleted after the backup is made and the restore occurs after the tombstone for the deleted object has been permanently removed. The restored DC would have the object as it existed before the deletion, and the other DCs would have no record that the object ever existed. In this case, an administrator will have to manually delete each non-replicated object on the restored domain controller.

Incremental backups of Active Directory Domain Services are not supported; full backups are required.

Backing Up an Active Directory Server

6/3/2022 • 2 minutes to read • [Edit Online](#)

An Active Directory server backup requires you to back up the database and the transaction logs. This topic provides a walkthrough of how a backup application backs up the Active Directory directory service.

The caller of these backup functions must have the SE_BACKUP_NAME privilege. You can use the [DsSetAuthIdentity](#) function to set the security context under which the directory backup/restore functions are called.

To backup an Active Directory server, perform the following steps

1. Call the [DsIsNTDSOnline](#) function to determine if Active Directory Domain Services are running.
2. If Active Directory Domain Services are running, call the [DsBackupPrepare](#) function to initialize a backup context handle. If Active Directory Domain Services are not running, it cannot be backed up and the backup application must fail the backup operation.
3. Call the [DsBackupGetDatabaseNames](#) function to get a list of files to back up. To release the memory returned by this function, call the [DsBackupFree](#) function.
4. For each name in the returned list of files, call the [DsBackupOpenFile](#) function followed by repeated calls to the [DsBackupRead](#) function until the entire file has been read. When you have finished reading the file, call the [DsBackupClose](#) function to close it.
5. After all database files are backed up, call the [DsBackupGetBackupLogs](#) function to get a list of transaction logs. This list is handled just like the list of database files.
6. When you have finished backing up the transaction log, call the [DsBackupTruncateLogs](#) function to delete all committed transaction logs that were backed up.
7. Save the contents of the expiry token provided by the [DsBackupPrepare](#) function. This can be saved in a file or some other persistent memory. This token must be passed to the [DsRestorePrepare](#) function to initiate a restore operation.
8. Free the memory for the expiry token by passing the token pointer to the [DsBackupFree](#) function.
9. Finally, call the [DsBackupEnd](#) function to release all resources associated with the backup context handle.

Restoring an Active Directory Server

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory servers must be restored offline. The system must be restarted in Directory Services Restore mode. In this mode, the operating system is running without Active Directory Domain Services and all user validation occurs through the Security Accounts Manager (SAM) in the registry. To restore Active Directory Domain Services, use the credentials of a local administrator on the domain controller that is restored.

The caller of the restore functions must have the **SE_RESTORE_NAME** access privilege. Use the **DsSetAuthIdentity** function to set the security context under which the directory backup and restore functions are called.

Be aware that when you restore Active Directory Domain Services, you must also restore the other system state components.

To restore Active Directory Domain Services

1. Call the **DsIsNTDSOnline** function to determine if Active Directory Domain Services are running.
2. If Active Directory Domain Services are not running, the **DsRestorePrepare** function is called to initialize the restore operation and obtain a backup context handle. If Active Directory Domain Services are running, it cannot be restored and the restore application must fail the restore operation. The **DsRestorePrepare** function requires that the expiry token be obtained from the **DsBackupPrepare** function during the backup operation.
3. Call the **DsRestoreGetDatabaseLocations** function to determine the directories where the files are to be restored. If this function fails, restore the data back to the original backup source directory; that is the directory from which the data was backed up.
4. Call the **DsRestoreRegister** function to prepare the Active Directory server for the restore operation and lock the restore directories.
5. Use standard Win32 functions to restore the files. First, delete all files in the destination directory; then copy the backup files to the destination directory.
6. Call the **DsRestoreRegisterComplete** function to indicate that the restore has completed.
7. Call the **DsRestoreEnd** function to release any resources associated with the context.

After a restore in Directory Services Restore mode, the domain controller should be restarted in normal mode. When the directory service starts, the domain controller will perform the normal consistency check and the restored directory will then be online.

Be aware that restoring an Active Directory server is always a two-part operation. First, restore the database to a time when the backup was taken. Second, replicate the directory, where the newly restored DSA replicates post-backup updates from other DSAs in the domain and enterprise forest.

A computer running on Windows 2000 or Windows Server 2003, that contains a replica of the directory service, is a domain controller.

The **DsRestoreRegister** function adds data to the registry that must survive the registry restoration process for the restoration to work correctly. To ensure this registry data is preserved, restore Active Directory Domain Services with the **DsRestore*** functions prior to restarting the computer after the **RegReplaceKey** function is called. This process works because **RegReplaceKey** does not actually replace the registry hive until the computer is restarted and the registry data added by the **DsRestoreRegister** function is specifically excluded from being replaced during a registry restore operation.

Storing Dynamic Data

6/3/2022 • 2 minutes to read • [Edit Online](#)

Most data stored using Active Directory Domain Services is static. For example, directory objects that represent users, groups, printers, network services, and so on, are typically long-lived and infrequently changed.

In Windows Server 2003 and later versions of Windows, Active Directory Domain Services provide support for the following features that enable applications to use the directory for storing dynamic data:

- [Dynamic Objects](#)
- [Dynamic Auxiliary Classes](#)

Dynamic Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Windows Server 2003 and later versions of Windows, Active Directory Domain Services provide support for storing dynamic entries in the directory. A dynamic entry is an object in the directory which has an associated time-to-live (TTL) value. The TTL for an entry is set when the entry is created. The time-to-live is automatically decremented, and when it expires the dynamic entry disappears. The client can cause a dynamic entry to remain longer than its current remaining life by refreshing (modifying) its TTL value. Clients that store dynamic data in the directory must periodically refresh that data to prevent it from disappearing.

Many applications and services that use LDAP to store and access relatively static and globally interesting data from an Active Directory server would also prefer to continue using LDAP access for their dynamic data needs. Moreover, for some applications, it may be desirable to off-load the task of garbage-collecting objects in the DS that have a limited useful life to the directory service. Application directory partitions (with the ability for controlled placement of replicas) and per-object TTLs together will provide the capability of hosting dynamic data in Active Directory Domain Services, thus allowing LDAP access to it.

A new auxiliary object class called **dynamicObject** with OID = 1.3.6.1.4.1.1466.101.119.2 will be defined in the base AD schema to be used by dynamic entries. This auxiliary class contains the attribute called **entryTTL** with OID = 1.3.6.1.4.1.1466.101.119.3 as a system-may-contain attribute. The value of this attribute is the "time in seconds" that the corresponding directory entry will continue to exist before disappearing from the directory. If the client does not supply a value for this attribute explicitly at object creation, then the DS provides a default value as specified later.

A new extended LDAP operation with OID = 1.3.6.1.4.1.1466.101.119.1 for client refresh of a dynamic entry in the directory will be defined and published in the **supportedExtension** attribute of the root DSE object.

In the actual implementation, **entryTTL** is a constructed attribute. The real object expiration time is stored as an absolute time when the object can be destroyed in a system-only attribute called **ms-DS-Entry-Time-To-Live**.

All dynamic objects have the following limitations:

- A deleted dynamic object due to its TTL expiring does not leave a tombstone behind.
- All DCs holding replicas of dynamic objects must run on Windows Server 2003.
- Dynamic entries with TTL values are supported in all partitions except the Configuration partition and Schema partition.
- Active Directory Domain Services do not publish the optional **dynamicSubtrees** attribute, as described in the RFC 2589, in the root DSE object.
- Dynamic entries are handled similar to non-dynamic entries when processing search, compare, add, delete, modify, and modifyDN operations.
- There is no way to change a static entry into a dynamic entry and vice-versa.
- A non-dynamic entry cannot be added subordinate to a dynamic entry.

Creating Dynamic Objects

6/3/2022 • 2 minutes to read • [Edit Online](#)

Specify the auxiliary class **dynamicObject** as a value for the **objectClass** attribute at the time of instantiating an object. In this case, the special auxiliary class will show up in the list of classes that is the value of the **objectClass** attribute for the object. If a value for the **entryTTL** attribute is not supplied at the time of adding the special auxiliary class, a default value, as specified later, will be provided by Active Directory Domain Services.

For more information about adding an auxiliary class to an object, see [Dynamic Auxiliary Classes](#).

Refreshing a Dynamic Object

6/3/2022 • 2 minutes to read • [Edit Online](#)

A client can refresh the Time To Live (TTL) of a given directory entry to keep it alive in one of two ways:

- Performing an LDAP update to the value of its **entryTTL** attribute before the entry is garbage collected. This method of refreshing a dynamic entry in the directory is an additional (optional) feature of Active Directory Domain Services that is not specified by RFC 2589.
- Performing an LDAP extended operation with an OID of 1.3.6.1.4.1.1466.101.119.1 for TTL refresh, as stipulated in the RFC 2589. This OID is defined as **LDAP_TTL_EXTENDED_OP_OID** in WINLDAP.H.

Remark*: Dynamic objects are removed by Garbage Collector when they have expired, there is no phase of the object being a tombstone when it has expired. You need to be careful with the AD replication latency when you refresh objects. You have to ensure that the update to refresh the TTL arrives early enough so all replicas of the naming context (full and global catalog) see the refresh transaction before the object expires locally.

Configuration of TTL Limits

6/3/2022 • 2 minutes to read • [Edit Online](#)

A client can request a TTL value for an entry ranging between 1 and 31557600 (that is, from 1 second to 1 year). This attribute value range will be enforced by the **rangeLower** and **rangeUpper** attribute values in the **attributeSchema** definition for the **entryTTL** attribute. As per RFC 2589, directory servers are not required to accept this value and might return a different TTL value to the client. Clients must be able to use this server-dictated value as their client refresh period (CRP) which will govern how often the client will need to perform the refresh operation on the dynamic entry.

Active Directory Domain Services provide administrators with the ability to configure the default and minimum TTL values for a forest. The default TTL value is what will be assigned to a newly created dynamic entry if a TTL is not explicitly specified. The minimum TTL will override any client specified TTL value that is lower than the configured minimum. No configurable maximum TTL value needs to be provided because a maximum will be imposed by the **rangeUpper** attribute value in the **attributeSchema** object. Allowing administrators to configure these values will enable them to set TTL values which will enforce a low refresh traffic or, at the other extreme, provide a highly up-to-date directory.

The default values for the two configurable TTL parameters will be as follows:

- Default TTL value = 86400 seconds (1 day)
- Minimum TTL value = 900 seconds (15 minutes)

The configurable TTL parameters will be stored as AVA (attribute value assertion) entries of the form "<value-name>=<value>" in the attribute **ms-DS-Other-Settings** of the NTDS-Service object given by the following DN in the Configuration partition:

```
CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration,DC=...
```

The exact AVA syntax, for the two configurable TTL parameters, is as follows where NNNN is expressed in seconds:

```
DynamicObjectDefaultTTLSeconds=NNNN
```

```
DynamicObjectMinTTLSeconds=NNNN
```

These values can be set by an administrator through the command-line utility ntdsutil.

Dynamic Auxiliary Classes

6/3/2022 • 2 minutes to read • [Edit Online](#)

Similar to structural and abstract object classes, auxiliary classes are defined by a [classSchema](#) object in the Active Directory schema. For more information, see [Structural, Abstract, and Auxiliary Classes](#). This schema definition specifies various characteristics of the class, including the attributes associated with the class.

Unlike with structural classes, You cannot create an instance of an auxiliary class as you can with a structural class. Instead, you use an auxiliary class to extend the list of attributes that is associated with another structural, abstract, or auxiliary object class.

In the initial release of Windows 2000, Active Directory Domain Services provided support for statically linking auxiliary classes to the [classSchema](#) definition of another object class. When an auxiliary class is used in this way, every instance of the object class supports the attributes of the auxiliary class.

Windows 2000 Server and earlier: Active Directory Domain Services does not provide support for dynamically linking auxiliary classes to individual objects, and not just to entire classes of objects. In addition, auxiliary classes that have been attached to an object instance cannot subsequently be removed from the instance.

Windows Server 2003: Dynamic Auxiliary Classes are supported when all domain controllers in the forest are running Windows Server 2003 and the forest functional mode is Windows Server 2003.

For more information about auxiliary classes, see:

- [Dynamically Linked Auxiliary Classes](#)
- [Statically Linked Auxiliary Classes](#)
- [Determining the Classes Associated With an Object Instance](#)
- [Adding an Auxiliary Class to an Object Instance](#)

For more information about forest functional levels, see "How to raise Active Directory domain and forest functional levels" in the Help and Support Knowledge Base at <https://support.microsoft.com/kb/322692#4>.

Dynamically Linked Auxiliary Classes

6/3/2022 • 2 minutes to read • [Edit Online](#)

A dynamically linked auxiliary class is a class that is attached to an individual object, rather than to an object class. Dynamic linking enables you to store additional attributes with an individual object without the forest-wide impact of extending the schema definition for an entire class. For example, an enterprise could use dynamic linking to attach a sales-specific auxiliary class to the user objects of its sales people, and other department-specific auxiliary classes to the user objects of employees in other departments.

Dynamic linking is not complex: add the name of the auxiliary class to the values of an object's **objectClass** attribute. If the auxiliary class has any mandatory attributes (**mustHave** or **systemMustHave**), you must set them at the same time. For more information and a code example, see [Adding an Auxiliary Class to an Object Instance](#).

To remove a dynamically linked auxiliary class, clear the values of all attributes from the auxiliary class, and then remove the name of the auxiliary class from the **objectClass** attribute of the object.

If you dynamically add an auxiliary class that is a subclass of another auxiliary class, both auxiliary classes are added to the target object. However, removing the child auxiliary class does not remove its parent; each class must be explicitly removed.

Statically Linked Auxiliary Classes

6/3/2022 • 2 minutes to read • [Edit Online](#)

A statically linked auxiliary class is one that is included in the **auxiliaryClass** or **systemAuxiliaryClass** attribute of an object class's **classSchema** definition in the schema. This means that the auxiliary class is part of every instance of the class with which it is associated.

An auxiliary class can be statically linked to an object class when the class is defined, that is, when its **classSchema** object is added to the schema container. This is the only time that the **systemAuxiliaryClass** can be used; after a **classSchema** object is created, its **systemAuxiliaryClass** attribute cannot be modified. An auxiliary class that is statically linked at this time can have mandatory (**mustHave**) and/or optional (**mayHave**) attributes.

A privileged user with the permissions required to extend the schema can add or remove auxiliary classes from the **systemAuxiliaryClass** attribute of an existing **classSchema** object. Doing so adds or removes the auxiliary class from every existing instance of the object class. An auxiliary class that is statically linked at this time can have optional attributes, but cannot have mandatory attributes. This is because there may be existing instances of the object class, in which case, adding a new mandatory attribute would create problems. A privileged user can subsequently remove an auxiliary class from the **auxiliaryClass** attribute of a **classSchema** object.

Determining the Classes Associated With an Object Instance

6/3/2022 • 2 minutes to read • [Edit Online](#)

Every object in Active Directory Domain Services has two attributes whose values identify the hierarchy of object classes of which the object is an instance.

ATTRIBUTE	DESCRIPTION
structuralObjectClass	Identifies the structural and abstract classes of which the object is an instance. For example, the values for a user object can be: <ul style="list-style-type: none">• top• person• organizationalPerson• user
objectClass	Identifies the classes included in structuralObjectClass , plus any auxiliary classes that are dynamically attached. For example, if a "vehicle" auxiliary class is attached to a user object, the values can be: <ul style="list-style-type: none">• top• vehicle• person• organizationalPerson• user

Be aware that neither of these attributes include auxiliary classes that are statically linked with the object classes of which the object is an instance. For example, the **securityPrincipal** auxiliary class is statically linked with the **user** class because it is included in the **systemAuxiliaryClass** values of the user **classSchema** object; it is not included in either the **objectClass** or **structuralObjectClass** attributes of instances of the user class.

Adding an Auxiliary Class to an Object Instance

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following code examples show how to use ADSI and LDAP to dynamically add an auxiliary class to an existing object instance. The examples assume that an auxiliary class named **vehicle** is defined in the Active Directory schema, and that the **vehicle** class has a **vin** attribute.

When you dynamically add an auxiliary class to an object instance, you must simultaneously specify values for any mandatory **mustHave** attributes in the class. The following examples show how to do this with the "vin" attribute, which is assumed to be mandatory.

The following C++ example binds to an object, and uses **IADs.PutEx** to append the auxiliary class to the list of classes in the object's **objectClass** property. Then the example uses **IADs.Put** to set the value of the **vin** attribute. Finally, it calls **IADs.SetInfo** to commit the changes to the directory.

```
LPWSTR pszAuxClass[]={L"vehicle"};
LPWSTR pszVIN[]={L"df897dsfsa-0"};
VARIANT var;

VariantInit(&var);

ADsOpenObject(L"cn=johnd,cn=users,dc=fabrikam,dc=com",
    NULL,
    NULL,
    ADS_SECURE_AUTHENTICATION,
    IID_IADs,
    (VOID**)&pIADs);

ADsBuildVarArrayStr(pszAuxClass, 1, &var);
pIADs->PutEx(ADS_PROPERTY_APPEND, CComBSTR("objectClass"), var);
ADsBuildVarArrayStr( pszVIN, 1, &var);
pIADs->Put(CComBSTR("vin"), var);
pIADs->SetInfo();

if(pIADs)
    pIADs->Release();

VariantClear(&var);
```

Application Directory Partitions

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Windows Server 2003, Active Directory Domain Services support application directory partitions. An application directory partition can contain a hierarchy of any type of objects, except security principals, and can be configured to replicate to any set of domain controllers in the forest. Unlike a domain partition, an application directory partition is not required to replicate to all domain controllers in a domain and the partition can replicate to domain controllers in different domains of the forest. For more information about application directory partitions, see [About Application Directory Partitions](#).

An application directory partition can be created, modified and deleted using standard ADSI, LDAP and [System.DirectoryServices](#) operations. The security context required to create and modify an application directory partition is the same as that for creating a domain partition.

The following topics describe how to work with application directory partitions:

- [Application Directory Partition Replication](#)
- [Application Directory Partition Security](#)
- [Creating an Application Directory Partition](#)
- [Deleting an Application Directory Partition](#)
- [Enumerating Application Directory Partitions in a Forest](#)
- [Locating an Application Directory Partition Host Server](#)
- [Adding or Deleting an Application Directory Partition Replica](#)
- [Enumerating Replicas of an Application Directory Partition](#)
- [Modifying Application Directory Partition Configuration](#)

Application Directory Partition Replication

6/3/2022 • 3 minutes to read • [Edit Online](#)

Application directory partitions are most often used to store dynamic data. Because data changes more often than the configuration data for a forest, the replication scope and frequency of an application directory partition can be set for each partition. The replication features of Active Directory Domain Services can be utilized, but the replication data can be fine-tuned to suit the type of data stored on the partition.

The operating system does not enforce a maximum number of replicas, but the number of replicas should be kept to a minimum to reduce the performance impact of replicating the dynamic application directory partition data.

The KCC generates and maintains the replication topology for an application directory partition.

Application Directory Partition Replication Within a Site

The replication intervals that control intra-site replication of an application directory partition can be configured. This enables the dynamic data in an application directory partition to be synchronized more promptly than the more static data in a domain partition. For more information about programmatically configuring an application directory partition, see [Modifying Application Directory Partition Configuration](#).

Two attributes on the application directory partition's **crossRef** object and two registry values on each Windows 2000 and later domain controller control the latency of initiating an originating change notification to replication partners within a site.

- The **msDS-Replication-Notify-First-DSA-Delay** attribute of a **crossRef** object specifies the delay, in seconds, after an originating object change before the first replication partner is notified. A registry value on each domain controller can specify a similar value. In a Windows Server 2003 forest, the default first delay is 15 seconds. In a mixed-mode forest, the default first delay is five minutes.
- The **msDS-Replication-Notify-Subsequent-DSA-Delay** attribute of a **crossRef** object specifies the delay, in seconds, between subsequent notifications to the second, third, and so on replication partners. A registry value on each domain controller can specify a similar value. In a Windows Server 2003 forest, the default subsequent delay is 3 seconds. In a mixed-mode forest, the default subsequent delay is 30 seconds.

The **crossRef** attributes apply to all domain controllers hosting a replica of the application directory partition and affect only replication of the application directory partition identified by the **crossRef** object. The registry values apply only to the domain controller on which they are set, and affect intra-site replication for all partitions that the domain controller is hosting. If neither the **crossRef** attributes nor its registry values are set, a domain controller uses the default values. If the registry values are set, they override any values set in the **crossRef** object. By default, the registry and **crossRef** values are not set, so the default values are used. This enables an administrator to speed up replication for all replicas of an application directory partition by setting the **crossRef** values, while enabling finer tuning with the registry settings on each domain controller.

Beginning with Windows Server 2003, domain partitions also use these attributes of the **crossRef** object to control intra-site replication latency. This is a change from previous server versions in which the delay intervals were controlled by registry values on each domain controller. When the forest is upgraded to Windows Server 2003, the existing registry values are preserved only if they have been modified from the default values. The domain controllers notification intervals in the registry override the notification intervals stored on the partition **crossRef** object.

Application Directory Partition Replication Across Sites

The replicas of an application directory partition that are located across sites observe the inter-site replication schedule as done for domain partition and global catalog replication. However, replicas of an application directory partition are more often located within a site when hosting truly volatile data because inter-site replication latency may not be acceptable to get the replicas to be consistent with each other.

Application Directory Partitions Are Not Replicated To Global Catalogs

Objects in an application directory partition are not replicated to the global catalog. The application directory partition is envisioned as hosting dynamic data and objects for which it may neither be sensible, nor feasible to replicate the objects widely. For this reason, the application directory partition offers controlled scope and frequency of replication. Because of this, there is no reason to allow these objects to replicate to the global catalog and hence be distributed across the entire forest where a global catalog server exists. This does not restrict objects in an application directory partition from using attributes in the schema marked as [isMemberOfPartialAttributeSet](#). As with any domain controller, a global catalog server can still be configured to be a full replica of an application directory partition.

Application Directory Partition Security

6/3/2022 • 2 minutes to read • [Edit Online](#)

The security and access control model for application directory partitions is the same as that for other partitions in Active Directory Domain Services. Normal users can access objects in an application directory partition subject to the ACLs placed on those objects. For more information, see [Controlling Access to Objects in Active Directory Domain Services](#).

However, because application directory partitions can span multiple security domains in a directory service, there arises the question of how to interpret domain-relative well-known SID string constants in the **defaultSecurityDescriptor** of an object's schema class at the time of object creation in an application directory partition. For example, if "DA" refers to the domain administrators group, but in an application directory partition, it is not known which domain the "DA" group refers to.

To solve this problem, the **crossRef** object of an application directory partition has an **msDS-SDReferenceDomain** attribute that contains the distinguished name of the reference domain for that application directory partition. The security system uses the specified domain to interpret local domain references for default security descriptors attached to objects created in that application directory partition. The reference domain can be specified when the **crossRef** object for an application directory partition is created. This requires, however, that a **crossRef** object be pre-created for the application directory partition. If no reference domain is specified, the system automatically sets the reference domain based on one of the following conditions:

- If the parent of the application directory partition object is another application directory partition, then the **msDS-SDReferenceDomain** attribute of the parent application directory partition is used for the reference domain.
- If the parent object is a domain, then that domain is used for the reference domain.
- If there is no parent object, then the forest root domain is used for the reference domain.

The **crossRef** attribute can also be changed to the reference domain after the partition is created, but this is not recommended.

A similar issue arises if a local group is specified in an ACL for an object in an application directory partition. In this case, the **msDS-SDReferenceDomain** attribute cannot be used to interpret the reference domain for a local group. To avoid this problem, local groups should not be used in the ACLs of application directory partition objects.

Creating an Application Directory Partition

6/3/2022 • 5 minutes to read • [Edit Online](#)

An application directory partition is represented by a **domainDNS** object with an **instanceType** attribute value of **DS_INSTANCETYPE_IS_NC_HEAD** combined with **DS_INSTANCETYPE_NC_IS_WRITEABLE**. This **domainDNS** object represents the application directory partition root (NC head), and is named similar to a regular domain partition, for example, "DC=dynamicdata,DC=fabrikam,DC=com", which corresponds to a DNS name of "dynamicdata.fabrikam.com". An application directory partition can, therefore, be instantiated anywhere a domain partition can be instantiated. There is no NetBIOS name associated with an application directory partition.

It is possible to nest application directory partitions, that is, an application directory partition can have child application directory partitions. Searches with subtree scope rooted at an application directory partition head will generate continuation references to the child application directory partitions.

An application directory partition replica can only be created on a domain controller that is running on Windows Server 2003 and later and only while the Domain-Naming FSMO role is held by a Windows Server 2003 and later domain controller. In a mixed forest that has both Windows Server 2003 domain controllers and down-level domain controllers (Windows 2000 domain controllers or Windows NT 4.0 primary domain controllers), an attempt to create an application directory partition replica on a down-level domain controller will fail.

An application directory partition also has a corresponding **crossRef** object in the Partitions container of the configuration partition. The **crossRef** can be pre-created manually before creating the **domainDNS** object. The pre-created **crossRef** object must have the attribute values shown in the following table or the partition creation will fail. If the **crossRef** object does not exist, the Active Directory server will create one when the application directory partition is created.

ATTRIBUTE	DESCRIPTION
dnsRoot	Contains the DNS path of the domain controller that the application directory partition will be created on.
Enabled	Contains FALSE.
nCName	Contains the distinguished name of the partition. In the example above, this attribute would contain "DC=dynamicdata,DC=mydomain,DC=com".

To create a new application directory partition with its first replica, perform the following steps

- Bind to the namespace for the new partition, specifying the domain controller that will host the application directory partition in the ADsPath. For example, to create a partition with an ADsPath of "DC=dynamicdata,DC=mydomain,DC=com", the binding ADsPath would be "LDAP://<domain controller>/DC=mydomain,DC=com", where "<domain controller>" is the DNS name of the domain controller that will host the partition.

The bind operation must specify the fast and delegation options. The fast option allows the bind to succeed even if the namespace does not exist. The delegation option is required to allow the domain controller to contact the Domain-Naming FSMO role holder using the same credentials.

The system version of the domain controller must be Windows Server 2003 operating system and later.

2. Create a **domainDNS** object with an appropriate name for the partition, for example, "DC=dynamicdata", to represent the naming context head for the new partition. The **domainDNS** object must have an **instanceType** attribute with a value of 5 (DS_INSTANCETYPE_IS_NC_HEAD | DS_INSTANCETYPE_NC_IS_WRITEABLE). The **instanceType** attribute can only be set at creation time because it is a system-only attribute.

When the **domainDNS** object is created, the Active Directory server will perform the following steps:

1. Searches for a **crossRef** object in the Partitions container that has an **nCName** attribute value that matches the distinguished name of the partition and, if a match is found, modifies the **crossRef** object to represent the application directory partition. If a matching **crossRef** object is not found, the Active Directory server creates a new **crossRef** object to represent the application directory partition.

The following table lists important attributes of the **crossRef** object.

ATTRIBUTE	DESCRIPTION
nCName	Contains the distinguished name of the partition.
dnsRoot	Contains the DNS name of the partition.
msDS-NC-Replica-Locations	The distinguished name of the nTDSSDA object of the domain controller for the first replica is added to this attribute.

2. Initiate a synchronization of the configuration partition and wait for completion. This enables the client application to modify configuration parameters for the newly created application directory partition while bound to the same domain controller used for creating the application directory partition.
3. Create the **domainDNS** object with the DS_INSTANCETYPE_IS_NC_HEAD and DS_INSTANCETYPE_NC_IS_WRITEABLE flags set on the **instanceType** property. The **instanceType** property may contain other, private flags as well.
4. Populate the **ms-DS-Has-Master-NCs** attribute of the **nTDSSDA** object for the target domain controller with the distinguished name of the newly created application directory partition.

When the application directory partition is created, or when a new replica of the application directory partition is added and fully synchronized, the Active Directory server correctly registers the replica with NetLogon and DNS. For more information, and a list of the registered SRV records, see [Locating an Application Directory Partition Host Server](#).

For more information about creating an application directory partition, see [Example Code for Creating an Application Directory Partition](#).

Locating the Partitions Container

The distinguished name of the Partitions container can be found in one of two ways. The first is more complicated to perform, but will always provide an accurate result:

1. Bind to RootDSE and obtain the **configurationNamingContext** attribute.
2. Use the **configurationNamingContext** attribute to bind to the configuration container.
3. Search in the configuration container for an object that is of the **crossRefContainer** type.
4. Obtain the **distinguishedName** attribute value of the **crossRefContainer** object. This will be the

distinguished name of the Partitions container.

For more information and a code example that shows this method for locating the Partitions container, see the **GetPartitionsDNSearch** function in [Example Code for Locating the Partitions Container](#).

The second method is easier to implement, but relies on the Partitions container having a particular relative distinguished name. It is not currently possible to change the name of the Partitions container, but if this capability is added in the future, the procedure below will not work correctly if the Partitions container has been renamed.

1. Bind to RootDSE and obtain the **configurationNamingContext** attribute.
2. Combine the string "CN=Partitions," followed by the **configurationNamingContext** attribute to form the distinguished name of the Partitions container. The distinguished name will be in the form "CN=Partitions, <configuration DN>".

For more information and a code example that shows this method for locating the Partitions container, see the **GetPartitionsDNManual** function in [Example Code for Locating the Partitions Container](#).

Example Code for Creating an Application Directory Partition

6/3/2022 • 6 minutes to read • [Edit Online](#)

The following C++ code example creates a new application directory partition using ADSI.

```
*****  
CreateApplicationPartitionIADs()  
  
Description: Creates an application directory partition.  
  
Parameters:  
  
pwszDCADsPath - Contains the ADsPath of the partition. This must  
also contain the DNS name of the domain controller on which the  
partition will be created. For example, the ADsPath  
"LDAP://DC01.fabrikam.com/DC=test,DC=com" would cause the  
partition to be created on DC01.fabrikam.com. The distinguished  
name of the partition will be "<pwszPartitionPath>,DC=test,DC=com".  
  
pwszUsername - Contains the user name to be used for  
authentication.  
  
pwszPassword - Contains the password to be used for  
authentication.  
  
pwszPartitionPath - Contains the relative distinguished name of  
the partition. This must be in the form of "DC=dynamicdata".  
  
pwszDescription - Contains a string that will be used for the  
description property for the domainDNS object.  
*****/  
  
HRESULT CreateApplicationPartitionIADs(LPCWSTR pwszDCADsPath,  
                                      LPCWSTR pwszUsername,  
                                      LPCWSTR pwszPassword,  
                                      LPCWSTR pwszPartitionPath,  
                                      LPCWSTR pwszDescription)  
{  
    HRESULT hr = E_FAIL;  
  
    IADsContainer *padsDC;  
  
    /*  
    Bind to the specified domain controller. The path must be in the  
    form "LDAP://<server DNS name>/<partition path>", in most cases,  
    the <partition path> will not be a valid path, so ADS_FAST_BIND  
    is used to allow the bind to succeed even if the path is invalid.  
    ADS_USE_DELEGATION is used to enable the LDAP provider to use the  
    credentials to contact the Domain Naming FSMO role holder to  
    create or modify the crossRef object.  
    */  
    hr = ADsOpenObject( pwszDCADsPath,  
                      pwszUsername,  
                      pwszPassword,  
                      ADS_SECURE_AUTHENTICATION |  
                      ADS_FAST_BIND |  
                      ADS_USE_DELEGATION,  
                      IID_IADsContainer,
```

```

        (LPVOID*)&padsDC);

    if(SUCCEEDED(hr))
    {
        CComBSTR sbstrPath = pwszPartitionPath;
        IDispatch *pDisp;

        // Create the domainDNS object.
        hr = padsDC->Create(CComBSTR("domainDNS"),
                              sbstrPath,
                              &pDisp);

        if(SUCCEEDED(hr))
        {
            IADs *padsPartition;

            // Get the IADs interface.
            hr = pDisp->QueryInterface(IID_IADs,
                                         (LPVOID*)&padsPartition);
            if(SUCCEEDED(hr))
            {
                CComVariant svar;

                // Set the instanceType property.
                svar = DS_INSTANCETYPE_IS_NC_HEAD |
                       DS_INSTANCETYPE_NC_IS_WRITEABLE;
                hr = padsPartition->Put(CComBSTR("instanceType"),
                                         svar);

                // Set the description property.
                svar = pwszDescription;
                hr = padsPartition->Put(CComBSTR("description"),
                                         svar);

                // Commit the new object to the server.
                hr = padsPartition->SetInfo();

                padsPartition->Release();
            }

            pDisp->Release();
        }

        padsDC->Release();
    }

    return hr;
}

```

The following Visual Basic Scripting Edition code example shows how to create a new application directory partition using ADSI.

```

' CreateApplicationPartitionVBS()
'
' Description: Creates an application directory partition.
'
' Parameters:
'
' DCADsPath - Contains the ADsPath of the partition. This must also
' contain the DNS name of the domain controller on which the
' partition will be created. For example, the ADsPath
' "LDAP://DC01.fabrikam.com/DC=test,DC=com" would cause the partition
' to be created on DC01.fabrikam.com. The distinguished name of the
' partition will be "<pwszPartitionPath>,DC=test,DC=com".
'
' Username - Contains the user name to be used for authentication.
'
```

```

' Password - Contains the password to be used for authentication.
'

' PartitionPath - Contains the relative distinguished name of the
' partition. This must be in the form of "DC=dynamicdata".
'

' Description - Contains a string that will be used for the
' description property for the domainDNS object.

Const ADS_SECURE_AUTHENTICATION = 1
Const ADS_FAST_BIND = 32
Const ADS_USE_DELEGATION = 256

Const DS_INSTANCETYPE_IS_NC_HEAD = 1
Const DS_INSTANCETYPE_NC_IS_WRITEABLE = 4

Sub CreateApplicationPartitionVBS( DCADsPath, _
                                 Username, _
                                 Password, _
                                 PartitionPath, _
                                 Description)
    set oNSP = GetObject("LDAP:")

    ' Bind to the specified domain controller. The path must be in the
    ' form "LDAP://<server DNS name>/<partition path>", in most cases,
    ' the <partition path> will be an invalid path, so ADS_FAST_BIND
    ' is used to allow the bind to succeed even if the path is
    ' invalid. ADS_USE_DELEGATION is used to enable the LDAP provider
    ' to use the credentials to contact the Domain Naming FSMO role
    ' holder to create or modify the crossRef object.
    If Username = "" or Username = vbNullString Then
        set oParent = oNSP.OpenDSObject(DCADsPath, _
                                         vbNullString, _
                                         vbNullString, _
                                         ADS_SECURE_AUTHENTICATION Or _
                                         ADS_FAST_BIND Or _
                                         ADS_USE_DELEGATION)
    Else
        set oParent = oNSP.OpenDSObject(DCADsPath, _
                                         Username, _
                                         Password, _
                                         ADS_SECURE_AUTHENTICATION Or _
                                         ADS_FAST_BIND Or _
                                         ADS_USE_DELEGATION)
    End If

    ' Create the domainDNS object.
    set oNewPartition = oParent.Create("domainDNS", PartitionPath)

    ' Set the instanceType property.
    oNewPartition.Put "instanceType", DS_INSTANCETYPE_IS_NC_HEAD Or _
                     DS_INSTANCETYPE_NC_IS_WRITEABLE

    ' Set the description property.
    oNewPartition.Put "description", Description

    ' Commit the new object to the server.
    oNewPartition.SetInfo

    set oNewPartition = Nothing
    set oFalseParent = Nothing
    set oNSP = Nothing
    set oPathName = Nothing
End Sub

```

The following Visual Basic .NET code example shows how to create a new application directory partition using [System.DirectoryServices](#).

```

Imports System.DirectoryServices

' CreateApplicationPartitionVBNet()
'
' Description: Creates an application directory partition.
'
' Parameters:
'
' DCADsPath - Contains the ADsPath of the partition. This must also
' contain the DNS name of the domain controller that the partition
' will be created on. For example, the ADsPath
' "LDAP://DC01.fabrikam.com/DC=test,DC=com" would cause the partition
' to be created on DC01.fabrikam.com. The distinguished name of the
' partition will be "<pwszPartitionPath>,DC=test,DC=com".
'
' Username - Contains the user name to be used for authentication.
'
' Password - Contains the password to be used for authentication.
'
' PartitionPath - Contains the relative distinguished name of the
' partition. This must be in the form of "DC=dynamicdata".
'
' Description - Contains a string that will be used for the
' description property for the domainDNS object.

Sub CreateApplicationPartitionVBNet(ByVal DCADsPath As String,
                                    ByVal Username As String,
                                    ByVal Password As String,
                                    ByVal PartitionPath As String,
                                    ByVal Description As String)
    Dim parent As DirectoryEntry
    Dim domainDNS As DirectoryEntry

    ' Bind to the specified domain controller. The path must be in the
    ' form "LDAP://<server DNS name>/<partition path>", in most cases,
    ' the <partition path> will be an invalid path, so
    ' AuthenticationTypes.FastBind is used to enable the bind to
    ' succeed even if the path is invalid.
    ' AuthenticationTypes.Delegation is used to enable the LDAP
    ' provider to use the credentials to contact the Domain Naming
    ' FSMO role holder to create or modify the crossRef object.
    parent = New DirectoryEntry(DCADsPath, _
                                Username, _
                                Password, _
                                AuthenticationTypes.Secure Or _
                                AuthenticationTypes.FastBind Or _
                                AuthenticationTypes.Delegation)

    ' Create the domainDNS object.
    domainDNS = parent.Children.Add(PartitionPath, "domainDNS")

    ' Set the instanceType property.
    domainDNS.Properties("instanceType").Value = 5

    ' Set the description property.
    domainDNS.Properties("description").Value = Description

    ' Commit the new object to the server.
    domainDNS.CommitChanges()
End Sub

```

The following C# code example shows how to create a new application directory partition using [System.DirectoryServices](#).

```

using System;
using System.DirectoryServices;

```

```

***** CreateApplicationPartitionCS()

Description: Creates an application directory partition.

Parameters:

DCADsPath - Contains the ADsPath of the partition. This must also
contain the DNS name of the domain controller that the partition
will be created on. For example, the ADsPath
"LDAP://DC01.fabrikam.com/DC=test,DC=com" would cause the
partition to be created on DC01.fabrikam.com. The distinguished
name of the partition will be
"<pwszPartitionPath>,DC=test,DC=com".

Username - Contains the user name to be used for authentication.

Password - Contains the password to be used for authentication.

PartitionPath - Contains the relative distinguished name of the
partition. This must be in the form of "DC=dynamicdata".

Description - Contains a string that will be used for the
description property for the domainDNS object.

*****

```

```

static void CreateApplicationPartitionCS(string DCADsPath,
    string Username,
    string Password,
    string PartitionPath,
    string Description)
{
    /*
    Bind to the specified domain controller. The path must be in the
    form "LDAP://<server DNS name>/<partition path>", in most cases,
    the <partition path> will be an invalid path, so
    AuthenticationTypes.FastBind is used to enable the bind to
    succeed even if the path is invalid.
    AuthenticationTypes.Delegation is used to allow the LDAP
    provider to use the credentials to contact the Domain Naming
    FSMO role holder to create or modify the crossRef object.
    */
    DirectoryEntry parent = new DirectoryEntry(DCADsPath,
        Username,
        Password,
        AuthenticationTypes.Secure |
        AuthenticationTypes.FastBind |
        AuthenticationTypes.Delegation);

    // Create the domainDNS object.
    DirectoryEntry domainDNS = parent.Children.Add(PartitionPath,
        "domainDNS");

    // Set the instanceType property.
    domainDNS.Properties["instanceType"].Value = 5;

    // Set the description property.
    domainDNS.Properties["description"].Value = Description;

    // Commit the new object to the server.
    domainDNS.CommitChanges();
}

```


Example Code for Locating the Partitions Container

6/3/2022 • 4 minutes to read • [Edit Online](#)

This topic includes code examples that locate the partition container.

The following C/C++ code example gets the distinguished name of the Partitions container by searching the configuration container for the `crossRefContainer` object.

```
/*****************************************************************************  
GetPartitionsDNSearch()  
  
Description: Gets the distinguished name of the Partitions container in  
the current enterprise forest.  
  
Parameters:  
  
ppwszPartitionsDN - Pointer to an LPWSTR that receives the distinguished  
name string. The caller must free this memory with  
FreeADsMem when it is no longer required.  
  
*****  
HRESULT GetPartitionsDNSearch(LPWSTR *ppwszPartitionsDN)  
{  
  
    *ppwszPartitionsDN = NULL;  
  
    HRESULT hr;  
  
    IADs *padsRootDSE;  
  
    // Bind to the RootDSE to get the configurationNamingContext property.  
    hr = ADsOpenObject( L"LDAP://RootDSE",  
                        NULL,  
                        NULL,  
                        ADS_SECURE_AUTHENTICATION,  
                        IID_IADs,  
                        (LPVOID*)&padsRootDSE);  
  
    if(SUCCEEDED(hr))  
    {  
        CComVariant svar;  
  
        // Get the configurationNamingContext property.  
        hr = padsRootDSE->Get(CComBSTR("configurationNamingContext"), &svar);  
        if(SUCCEEDED(hr))  
        {  
            IDirectorySearch *pConfigSearch;  
  
            // Bind to the Configuration container.  
            CComBSTR sbstrPath = "LDAP://";  
            sbstrPath += svar.bstrVal;  
  
            hr = ADsOpenObject( sbstrPath,  
                                NULL,  
                                NULL,  
                                ADS_SECURE_AUTHENTICATION,  
                                IID_IDirectorySearch,  
                                (LPVOID*)&pConfigSearch);  
  
            if(SUCCEEDED(hr))  
            {  
                pConfigSearch->Close();  
                pConfigSearch->Release();  
            }  
        }  
    }  
}
```

```

    {
        // Search for an object that is of type crossRefContainer.
        ADS_SEARCHPREF_INFO SearchPref[1];

        // Set the search scope.
        SearchPref[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
        SearchPref[0].vValue.dwType = ADSTYPE_INTEGER;
        SearchPref[0].vValue.Integer = ADS_SCOPE_ONELEVEL;

        hr = pConfigSearch->SetSearchPreference(SearchPref,
sizeof(SearchPref)/sizeof(ADS_SEARCHPREF_INFO));
        if(SUCCEEDED(hr))
        {
            ADS_SEARCH_HANDLE hSearch = NULL;
            LPWSTR pwszAttributes[1] = {L"distinguishedName"};
            LPWSTR pwszSearchFilter = L"(&(objectClass=crossRefContainer))";

            // Execute the search.
            hr = pConfigSearch->ExecuteSearch(pwszSearchFilter,
                pwszAttributes,
                sizeof(pwszAttributes)/sizeof(LPWSTR),
                &hSearch);
            if(SUCCEEDED(hr))
            {
                // Get the first result row. There should never be more than one match.
                hr = pConfigSearch->GetFirstRow(hSearch);
                if(S_OK == hr)
                {
                    ADS_SEARCH_COLUMN col;

                    // Get the search result. The distinguishedName attribute will be a string.
                    hr = pConfigSearch->GetColumn(hSearch, pwszAttributes[0], &col);
                    if(SUCCEEDED(hr))
                    {
                        // col.pADsValues[0].DNString;
                        DWORD dwChars = lstrlenW(col.pADsValues[0].DNString) + 1;

                        *ppwszPartitionsDN = (LPWSTR)AllocADSMem(dwChars * sizeof(WCHAR));

                        if(*ppwszPartitionsDN)
                        {
                            wcsncpy_s(*ppwszPartitionsDN, col.pADsValues[0].DNString, dwChars);
                        }
                        else
                        {
                            hr = E_OUTOFMEMORY;
                        }

                        pConfigSearch->FreeColumn(&col);
                    }
                }
                else
                {
                    hr = HRESULT_FROM_WIN32(ERROR_NOT_FOUND);
                }

                // Close the search handle to cleanup.
                pConfigSearch->CloseSearchHandle(hSearch);
            }
        }

        pConfigSearch->Release();
    }

    padsRootDSE->Release();
}

return hr;

```

```
    ' Search ... ,
```

The following Visual Basic Scripting Edition code example gets the distinguished name of the Partitions container by searching the configuration container for the [crossRefContainer](#) object.

```
Const ADS_SECURE_AUTHENTICATION = 1

' GetPartitionsDNSearch

' Description: Gets the distinguished name of the Partitions container in
' the current enterprise forest.

Function GetPartitionsDNSearch()
    ' Bind to RootDSE.
    Set oRootDSE = GetObject("LDAP://RootDSE")

    Set oConn = CreateObject("ADODB.Connection")
    Set oComm = CreateObject("ADODB.Command")

    oConn.Provider = "ADsDSOObject"

    ' Set the binding options for the search.
    oConn.Properties("ADSI Flag") = ADS_SECURE_AUTHENTICATION

    oConn.Open
    oComm.ActiveConnection = oConn

    oComm.CommandText = "<LDAP://" + oRootDSE.Get("configurationNamingContext") + ">;(&
(objectClass=crossRefContainer));distinguishedName;onelevel"

    ' WScript.Echo oComm.CommandText

    ' Execute the query.
    Set oResults = oComm.Execute

    ' Get the first result. This should be the only result.
    Set oField = oResults(0)

    GetPartitionsDNSearch = oField.Value
End Function
```

The following C/C++ code example gets the distinguished name of the Partitions container by manually building the distinguished name.

```

*****
GetPartitionsDNManual()

Description: Gets the distinguished name of the Partitions container in
the current enterprise forest.

Parameters:

ppwszPartitionsDN - Pointer to an LPWSTR that receives the distinguished
name string. The caller must free this memory with
FreeADsMem when it is no longer required.

*****

```

HRESULT GetPartitionsDNManual(LPWSTR *ppwszPartitionsDN)

{

```

*ppwszPartitionsDN = NULL;

HRESULT hr;

IADs *padsRootDSE;

// Bind to the RootDSE to get the configurationNamingContext property.
hr = ADsOpenObject( L"LDAP://RootDSE",
                    NULL,
                    NULL,
                    ADS_SECURE_AUTHENTICATION,
                    IID_IADs,
                    (LPVOID*)&padsRootDSE);

if(SUCCEEDED(hr))
{
    CComVariant svar;

    // Get the configurationNamingContext property.
    hr = padsRootDSE->Get(CComBSTR("configurationNamingContext"), &svar);
    if(SUCCEEDED(hr))
    {
        CComBSTR sbstrDN = "CN=Partitions,";
        sbstrDN += svar.bstrVal;
        DWORD dwChars = sbstrDN.Length() + 1;

        *ppwszPartitionsDN = (LPWSTR)AllocADsMem(dwChars * sizeof(WCHAR));

        if(*ppwszPartitionsDN)
        {
            wcsncpy_s(*ppwszPartitionsDN, sbstrDN.m_str, dwChars);
        }
        else
        {
            hr = E_OUTOFMEMORY;
        }
    }

    padsRootDSE->Release();
}

return hr;
}

```

The following Visual Basic code example gets the distinguished name of the Partitions container by manually building the distinguished name.

```

' GetPartitionsDNManual
'
' Description: Gets the distinguished name of the Partitions container in
' the current enterprise forest.
'

Function GetPartitionsDNManual()
    ' Bind to RootDSE.
    Set oRootDSE = GetObject("LDAP://RootDSE")

    ' Get the configurationNamingContext property.
    GetPartitionsDNManual = "CN=Partitions," + oRootDSE.Get("configurationNamingContext")
End Function

```

The following C# code example gets the distinguished name of the Partitions container by searching the configuration container for the [crossRefContainer](#) object. This example uses C# with System.DirectoryServices.

```

//****************************************************************************
GetPartitionsDN()

Description: Gets the distinguished name of the Partition container in
the current enterprise forest.

***** */

static string GetPartitionsDN()
{
    // Bind to the RootDSE to get the configurationNamingContext property.
    DirectoryEntry RootDSE = new DirectoryEntry("LDAP://RootDSE");
    DirectoryEntry ConfigContainer = new DirectoryEntry("LDAP://" +
        RootDSE.Properties["configurationNamingContext"].Value);

    // Search for an object that is of type crossRefContainer.
    DirectorySearcher ConfigSearcher = new DirectorySearcher(ConfigContainer);
    ConfigSearcher.Filter = "(&(objectClass=crossRefContainer))";
    ConfigSearcher.PropertiesToLoad.Add("distinguishedName");
    ConfigSearcher.SearchScope = SearchScope.OneLevel;

    SearchResult result = ConfigSearcher.FindOne();

    return result.Properties["distinguishedName"][0].ToString();
}

```

The following Visual Basic .NET code example gets the distinguished name of the Partition container by searching the configuration container for the [crossRefContainer](#) object. This example uses Visual Basic .NET with System.DirectoryServices.

```
'*****  
' GetPartitionsDN()  
'  
' Description: Gets the distinguished name of the Partitions container in  
' the current enterprise forest.  
'  
'*****  
  
Function GetPartitionsDN() As String  
    ' Bind to the RootDSE to get the configurationNamingContext property.  
    Dim RootDSE As New DirectoryEntry("LDAP://RootDSE")  
  
    ' Bind to the Configuration container.  
    Dim Path As String = "LDAP://" + RootDSE.Properties("configurationNamingContext").Value  
    Dim ConfigContainer As New DirectoryEntry(Path)  
  
    ' Search for an object that is of type crossRefContainer.  
    Dim ConfigSearcher As New DirectorySearcher(ConfigContainer)  
    ConfigSearcher.Filter = "(&(objectClass=crossRefContainer))"  
    ConfigSearcher.SearchScope = SearchScope.OneLevel  
  
    Dim result As SearchResult = ConfigSearcher.FindOne()  
  
    Return result.Properties("distinguishedName")(0).ToString()  
End Function
```

Deleting an Application Directory Partition

6/3/2022 • 2 minutes to read • [Edit Online](#)

An application directory partition is deleted by deleting the **crossRef** object that represents the application directory partition. When the deletion of the **crossRef** object replicates to a domain controller that has a replica of the application directory partition, the KCC will delete the local replica of the application directory partition. This eventually causes all replicas of the application directory partition to be deleted.

When the **crossRef** object is deleted, the Active Directory server will delete the **domainDNS** object that was created when the application directory partition was created, as well as deleting all objects in the application directory partition. None of the objects in the application directory partition are tombstoned when they deleted.

When an application directory partition is deleted, it is very difficult to restore it. To restore an application directory partition, it is necessary to restore a backup that has a replica of the application directory partition, find the **crossRef** object that represents the application directory partition and authoritatively restore the **crossRef** object.

To delete an application directory partition and its replicas, perform the following steps

1. Search the Partitions container for a **crossRef** object that has an **nCName** attribute value that is equal to the distinguished name of the application directory partition.
2. Delete the **crossRef** object.

For more information, see [Example Code for Deleting an Application Directory Partition](#).

Example Code for Deleting an Application Directory Partition

6/3/2022 • 7 minutes to read • [Edit Online](#)

The following C++ code example can be used to delete an application directory partition using ADSI. This example uses one of the `GetPartitionsDN*` example functions described in [Example Code for Locating the Partitions Container](#).

```
*****  
GetCrossRefDNFromPartitionDN()  
  
Description: Gets the distinguished name of the crossRef object  
that represents the specified application directory partition.  
  
Parameters:  
  
pwszPartitionDN - Contains the distinguished name of the  
application directory partition for which to find the crossRef  
object.  
  
pwszUsername - Contains the user name to be used for  
authentication.  
  
pwszPassword - Contains the password to be used for  
authentication.  
  
ppwszCrossRefDN - Pointer to an LPWSTR that receives the  
distinguished name string. The caller must free this memory  
with FreeADsMem when it is no longer required.  
*****/  
  
HRESULT GetCrossRefDNFromPartitionDN(LPCWSTR pwszPartitionDN,  
                                     LPCWSTR pwszUsername,  
                                     LPCWSTR pwszPassword,  
                                     LPWSTR *ppwszCrossRefDN)  
{  
    *ppwszCrossRefDN = NULL;  
  
    HRESULT hr;  
    LPWSTR pwszPartitionsDN;  
  
    // Get the distinguished name of the Partitions container.  
    hr = GetPartitionsDN(&pwszPartitionsDN);  
    if(SUCCEEDED(hr))  
    {  
        /*  
        Search the Partitions container for an object that is of  
        type crossRef and has an nCName that matches the distinguished  
        name of the partition. This will be the crossRef object that  
        represents the partition and the one to delete.  
        */  
        IDirectorySearch *pdsPartitions;  
  
        CComBSTR sbstrADsPath = "LDAP://";  
        sbstrADsPath += pwszPartitionsDN;  
  
        // Bind to the Partitions container.  
        hr = ADsOpenObject( sbstrADsPath,
```

```

        pwszUsername,
        pwszPassword,
        ADS_SECURE_AUTHENTICATION,
        IID_IDirectorySearch,
        (LPVOID*)&pdsPartitions);

if(SUCCEEDED(hr))
{
    ADS_SEARCHPREF_INFO SearchPref[1];

    // Set the search scope.
    SearchPref[0].dwSearchPref = ADS_SEARCHPREF_SEARCH_SCOPE;
    SearchPref[0].vValue.dwType = ADSTYPE_INTEGER;
    SearchPref[0].vValue.Integer = ADS_SCOPE_ONELEVEL;

    hr = pdsPartitions->SetSearchPreference(
        SearchPref,
        sizeof(SearchPref)/sizeof(ADS_SEARCHPREF_INFO)
    );
    if(SUCCEEDED(hr))
    {
        ADS_SEARCH_HANDLE hSearch = NULL;
        LPWSTR pwszAttributes[1] = {L"distinguishedName"};

        CComBSTR sbstrSearchFilter;
        sbstrSearchFilter = "(&(objectClass=crossRef)(nCName=";
        sbstrSearchFilter += pwszPartitionDN;
        sbstrSearchFilter += ")");

        // Execute the search.
        hr = pdsPartitions->ExecuteSearch(
            sbstrSearchFilter,
            pwszAttributes,
            sizeof(pwszAttributes)/sizeof(LPWSTR),
            &hSearch
        );
        if(SUCCEEDED(hr))
        {
            // Get the first result row. There should never be more
            // than one result.
            hr = pdsPartitions->GetFirstRow(hSearch);
            if(S_OK == hr)
            {
                ADS_SEARCH_COLUMN col;

                // Get the search result. The distinguishedName
                // attribute will be a string.
                hr = pdsPartitions->GetColumn(hSearch,
                    pwszAttributes[0],
                    &col);
                if(SUCCEEDED(hr))
                {
                    // Allocate and copy the returned
                    // distinguished name buffer.
                    DWORD dwChars =
                        lstrlenW(col.pADsValues[0].DNString) + 1;

                    *ppwszCrossRefDN =
                        (LPWSTR)AllocADSMem(dwChars *
                            sizeof(WCHAR));

                    if(*ppwszCrossRefDN)
                    {
                        wcsncpy_s(*ppwszCrossRefDN,
                            col.pADsValues[0].DNString,
                            dwChars);
                    }
                    else
                    {
                        hr = E_OUTOFMEMORY;
                    }
                }
            }
        }
    }
}

```

```

        }

        // Free the column.
        pdsPartitions->FreeColumn(&col);
    }
}

else
{
    hr = HRESULT_FROM_WIN32(ERROR_NOT_FOUND);
}

// Close the search handle to cleanup.
pdsPartitions->CloseSearchHandle(hSearch);
}

}

FreeADsMem(pwszPartitionsDN);
}

return hr;
}

//******************************************************************************

DeleteAppPartition()

Description - Deletes an application directory partition by
finding the crossRef object for the partition and then deleting
the crossRef object.

***** */

HRESULT DeleteAppPartition( LPCWSTR pwszPartitionDN,
                            LPCWSTR pwszUsername,
                            LPCWSTR pwszPassword)
{
    LPWSTR pwszCrossRefDN;
    HRESULT hr;

    // Get the distinguished name of the crossRef object.
    hr = GetCrossRefDNFromPartitionDN(pwszPartitionDN,
                                       pwszUsername,
                                       pwszPassword,
                                       &pwszCrossRefDN);

    if(SUCCEEDED(hr))
    {
        CComBSTR sbstrADsPath;
        IADsDeleteOps *pDelete;

        // Bind to the crossRef object and delete it.
        sbstrADsPath = "LDAP://";
        sbstrADsPath += pwszCrossRefDN;
        hr = ADsOpenObject( sbstrADsPath,
                           NULL,
                           NULL,
                           ADS_SECURE_AUTHENTICATION,
                           IID_IADsDeleteOps,
                           (LPVOID*)&pDelete);

        if(SUCCEEDED(hr))
        {
            hr = pDelete->DeleteObject(0);

            pDelete->Release();
        }

        FreeADsMem(pwszCrossRefDN);
    }
}

```

```
        return hr;  
    }
```

The following Visual Basic Scripting Edition code example can be used to delete an application directory partition using ADSI. This example uses one of the **GetPartitionsDN*** example functions described in [Example Code for Locating the Partitions Container](#).

```
Const ADS_SECURE_AUTHENTICATION = 1  
  
' GetCrossRefDNFromPartitionDN()  
'  
' Description: Gets the distinguished name of the crossRef object  
' that represents the specified application directory partition.  
'  
'Parameters:  
'  
' PartitionDN - Contains the distinguished name of the application  
' directory partition for which to find the crossRef object.  
'  
' Username - Contains the user name to be used for authentication.  
'  
' Password - Contains the password to be used for authentication.  
'  
Function GetCrossRefDNFromPartitionDN(PartitionDN, Username, Password)  
    ' Search the Partitions container for an object that is of type  
    ' crossRef and has an nCName that matches the distinguished name  
    ' of the partition. This will be the crossRef object that represents  
    ' the partition.  
  
    partitionsDN = GetPartitionsDN()  
  
    commandString = "<LDAP://" + partitionsDN + ">(&(objectClass=crossRef)(nCName=" + PartitionDN +  
    "));distinguishedName;onelevel"  
  
    Set oConn = CreateObject("ADODB.Connection")  
    Set oComm = CreateObject("ADODB.Command")  
  
    oConn.Provider = "ADsDSOObject"  
  
    ' Set the binding options for the search.  
    oConn.Properties("ADSI Flag") = ADS_SECURE_AUTHENTICATION  
  
    If Username <> vbNullString And Username <> "" Then  
        oConn.Properties("User ID") = strUsername  
        oConn.Properties("Password") = strPassword  
    End If  
  
    oConn.Open  
    oComm.ActiveConnection = oConn  
  
    oComm.CommandText = commandString  
  
    ' Execute the query.  
    Set oResults = oComm.Execute  
  
    ' Get the first result. This should be the only result.  
    Set oField = oResults(0)  
  
    GetCrossRefDNFromPartitionDN = oField.Value  
End Function  
  
' DeleteAppPartition()  
'
```

```

' Description: Deletes the specified application directory
' partition.
'

'Parameters:
'

' PartitionDN - Contains the distinguished name of the
' application directory partition to delete.
'

' Username - Contains the user name to be used for authentication.
'

' Password - Contains the password to be used for authentication.
'

Sub DeleteAppPartition(PartitionDN, Username, Password)
    ' Get the distinguished name of the crossRef object that
    ' represents the application directory partition.
    CrossRefADsPath = "LDAP://" + GetCrossRefDNFromPartitionDN(PartitionDN, Username, Password)

    Set oNSP = GetObject("LDAP:")

    ' Bind to the crossRef object.
    If Username = "" Then
        Set oCrossRef = GetObject(CrossRefADsPath)
    Else
        Set oCrossRef = oNSP.OpenDSObject(CrossRefADsPath, _
            Username, _
            Password, _
            ADS_SECURE_AUTHENTICATION)
    End If

    ' Delete the crossRef object using IADsDeleteOps.DeleteObject().
    oCrossRef.DeleteObject(0)
End Sub

```

The following C# code example can be used to delete an application directory partition using `System.DirectoryServices`. This example uses one of the `GetPartitionsDN*` example functions described in [Example Code for Locating the Partitions Container](#).

```

 ****
GetCrossRefDNFromPartitionDN()

Description: Gets the distinguished name of the crossRef object
that represents the specified application directory partition.

Parameters:

PartitionDN - Contains the distinguished name of the application
directory partition for which to find the crossRef object.

Username - Contains the user name to be used for authentication.

Password - Contains the password to be used for authentication.

****

static string GetCrossRefDNFromPartitionDN(
    string PartitionPath,
    string Username,
    string Password)
{
    string PartitionsDN = GetPartitionsDN();

    /*
    Search the Partitions container for an object that is of type crossRef
    application and has an nCName that matches the distinguished name
    of the partition. This will be the crossRef object that represents
    the partition and the one to delete.

```

```


/*
DirectoryEntry PartitionsContainer =
    new DirectoryEntry("LDAP://" + PartitionsDN,
                      Username,
                      Password,
                      AuthenticationTypes.Secure);
DirectorySearcher PartitionsSearcher =
    new DirectorySearcher(PartitionsContainer);

// Build the search filter.
PartitionsSearcher.Filter = "(&(objectClass=crossRef)(nCName=";
PartitionsSearcher.Filter += PartitionPath;
PartitionsSearcher.Filter += "))";

// Request the distinguishedName.
PartitionsSearcher.PropertiesToLoad.Add("distinguishedName");

// Search only one level.
PartitionsSearcher.SearchScope = SearchScope.OneLevel;

 SearchResult result = PartitionsSearcher.FindOne();
return result.Properties["distinguishedName"][0].ToString();
}

*****DeleteAppPartition()

Description: Deletes an application directory partition.

*****

```

```


static void DeleteAppPartition(
    string PartitionPath,
    string Username,
    string Password)
{
    /*
    Get the distinguished name of the crossRef object that represents
    the specified application directory partition.
    */
    string CrossRefDN = GetCrossRefDNFromPartitionDN(PartitionPath,
                                                    Username,
                                                    Password);

    // Bind to the crossRef object.
    DirectoryEntry Partition =
        new DirectoryEntry("LDAP://" + CrossRefDN,
                          Username,
                          Password,
                          AuthenticationTypes.Secure);

    // Delete the crossRef object.
    /*
    To avoid a problem in DirectoryEntry.DeleteTree(), it is necessary
    to access a property value before calling DeleteTree(). If this is
    not done, DeleteTree() will fail.
    */
    object CommonName = Partition.Properties["cn"].Value;
    Partition.DeleteTree();
}

```

The following Visual Basic .NET code example can be used to delete an application directory partition using `System.DirectoryServices`. This example uses one of the `GetPartitionsDN*` example functions described in [Example Code for Locating the Partitions Container](#).

```

'*****
'
' GetCrossRefDNFromPartitionDN()
'
' Description: Gets the distinguished name of the crossRef object
' that represents the specified application directory partition.
'
' Parameters:
'
' PartitionDN - Contains the distinguished name of the application
' directory partition for which to find the crossRef object.
'
' Username - Contains the user name to be used for authentication.
'
' Password - Contains the password to be used for authentication.
'
'*****

Function GetCrossRefDNFromPartitionDN(ByVal PartitionPath As String, _
    ByVal Username As String, _
    ByVal Password As String) _
    As String
Dim PartitionsDN As String
PartitionsDN = GetPartitionsDN()

    ' Search the Partitions container for an object that is of type
    ' crossRef and has an nCName that matches the distinguished name of
    ' the partition. This will be the crossRef object that represents
    ' the partition and the one to delete.

Dim Path As String = "LDAP://" + PartitionsDN
Dim PartitionsContainer _
    As New DirectoryEntry(Path, _
        Username, _
        Password, _
        AuthenticationTypes.Secure)
Dim PartitionsSearcher _
    As New DirectorySearcher(PartitionsContainer)

    ' Set the search filter.
PartitionsSearcher.Filter = "(&(objectClass=crossRef)(nCName="
PartitionsSearcher.Filter += PartitionPath
PartitionsSearcher.Filter += "))"

    ' Retrieve the distinguishedName property.
PartitionsSearcher.PropertiesToLoad.Add("distinguishedName")

    ' Search only one level.
PartitionsSearcher.SearchScope = SearchScope.OneLevel

    ' Perform the search.
Dim result As SearchResult = PartitionsSearcher.FindOne()

    ' Form and return the distinguished name of the crossRef object.
Return result.Properties("distinguishedName")(0).ToString()
End Function

'*****
'
' DeleteAppPartition()
'
' Description: Deletes an application directory partition.
'
'*****


Sub DeleteAppPartition(ByVal PartitionPath As String, _
    ByVal Username As String, _
    ByVal Password As String)
    ' Get the distinguished name of the crossRef object that

```

```
' represents the specified application directory partition.  
Dim CrossRefDN _  
    As String = GetCrossRefDNFromPartitionDN(PartitionPath, _  
        Username, _  
        Password)  
  
' Bind to the crossRef object.  
Dim Path As String = "LDAP://" + CrossRefDN  
Dim Partition As New DirectoryEntry(Path, _  
    Username, _  
    Password, _  
    AuthenticationTypes.Secure)  
  
' Delete the crossRef object.  
' To avoid a problem in DirectoryEntry.DeleteTree(), it is necessary to  
' access a property value before calling DeleteTree(). If this is not  
' done, DeleteTree() will fail.  
Dim CommonName As Object = Partition.Properties("cn").Value  
Partition.DeleteTree()  
End Sub
```

Enumerating Application Directory Partitions in a Forest

6/3/2022 • 2 minutes to read • [Edit Online](#)

Like domain partitions, every application directory partition is represented by a **crossRef** object in the Partitions container of the configuration partition. Each **crossRef** object stores data about its corresponding partition.

A **crossRef** object that represents a domain partition is distinguished from a **crossRef** object that represents an application directory partition by the contents of the **systemFlags** attribute. The **crossRef** object that represents a domain partition will have both the **ADS_SYSTEMFLAG_CR_NTDS_NC** and **ADS_SYSTEMFLAG_CR_NTDS_DOMAIN** flags set in the **systemFlags** attribute. The **crossRef** object that represents an application directory partition will have the **ADS_SYSTEMFLAG_CR_NTDS_NC** flag set and the **ADS_SYSTEMFLAG_CR_NTDS_DOMAIN** flag will not be set in the **systemFlags** attribute.

The **crossRef** objects that represent the Schema and Configuration partitions will also have the **ADS_SYSTEMFLAG_CR_NTDS_NC** flag set and the **ADS_SYSTEMFLAG_CR_NTDS_DOMAIN** flag will not be set in the **systemFlags** attribute. This requires that these two **crossRef** objects be distinguished by the contents of the **nName** attribute. The **nName** attribute for the **crossRef** object that represents the Schema container will be identical to the **schemaNamingContext** attribute of the RootDSE object. Similarly, the **nName** attribute for the **crossRef** object that represents the Configuration container will be identical to the **configurationNamingContext** attribute of the RootDSE object.

To identify all application directory partitions in a forest, perform the following steps

1. In the Partitions container of the configuration partition, search for or enumerate all **crossRef** objects.
2. If a **crossRef** object does not have the **ADS_SYSTEMFLAG_CR_NTDS_NC** flag set or has the **ADS_SYSTEMFLAG_CR_NTDS_DOMAIN** flag set in the **systemFlags** attribute value, exclude the object from the result set.
3. Exclude the Schema partition from the result set by comparing the **nName** attribute of the **crossRef** object to the **schemaNamingContext** attribute of the RootDSE object.
4. Exclude the Configuration partition from the result set by comparing the **nName** attribute of the **crossRef** object to the **configurationNamingContext** attribute of the RootDSE object.
5. The remaining **crossRef** objects in the result set all represent application directory partitions.

Locating an Application Directory Partition Host Server

6/3/2022 • 2 minutes to read • [Edit Online](#)

The NetLogon service registers the following list of SRV records for an application directory partition:

- _ldap_tcp.<partition name>
- _ldap_tcp.<site name>._sites.<partition name>

To locate a domain controller that hosts a replica of a specified application directory partition, call the **DsGetDcName** function with the *DomainName* parameter set to the DNS name of the application directory partition and the DS_ONLY_LDAP_NEEDED flag set in the *Flags* parameter. For more information and a code example that shows, using the **DsGetDcName** function, how to locate a domain controller that hosts a replica of an application directory partition, see [Example Code for Locating an Application Directory Partition Host Server](#).

Example Code for Locating an Application Directory Partition Host Server

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic includes a code example that locates an application directory partition host server.

The following C/C++ code example shows how to use the [DsGetDcName](#) function to locate a domain controller that hosts a replica of an application directory partition. This code example also shows how to use the [DsCrackNames](#) function to convert the distinguished name of an application directory partition to a DNS name.

```
*****  
AppPartitionDNToDNS()  
  
Converts the distinguished name of an application directory partition to  
a DNS name for the partition.  
  
The caller must free the memory allocated to ppszDNS by calling  
FreeADsMem when it is no longer required.  
*****  
  
DWORD AppPartitionDNToDNS(LPCTSTR pszDN, LPTSTR *ppszDNS)  
{  
    DWORD dwRet;  
    LPCTSTR rgpszNames[] = {pszDN};  
    PDS_NAME_RESULT pResults;  
  
    /*  
    Convert the distinguished name to a DNS name using only syntactic  
    mapping. The DS_NAME_FLAG_SYNTACTICAL_ONLY flag allows DsCrackNames to  
    be called without binding.  
    */  
    dwRet = DsCrackNames(NULL,  
        DS_NAME_FLAG_SYNTACTICAL_ONLY,  
        DS_FQDN_1779_NAME,  
        DS_CANONICAL_NAME,  
        1,  
        rgpszNames,  
        &pResults);  
  
    if(NO_ERROR == dwRet)  
    {  
        /*  
        Allocate the memory and copy the DNS name of the partition.  
        */  
        DWORD dwBytes = (lstrlen(pResults->rItems[0].pDomain) + 1) * sizeof(TCHAR);  
        *ppszDNS = (LPTSTR)AllocADsMem(dwBytes);  
        if(*ppszDNS)  
        {  
            wcsncpy_s(*ppszDNS, pResults->rItems[0].pDomain, dwBytes);  
  
            dwRet = NO_ERROR;  
        }  
        else  
        {  
            dwRet = ERROR_NOT_ENOUGH_MEMORY;  
        }  
    }  
}
```

```

    // Free the result set.
    DsFreeNameResult(pResults);
}

return dwRet;
}

//****************************************************************************

PrintDCFromAppPartition()

Given the distinguished name of an application directory partition,
prints to the console the DNS name of a domain controller that hosts a
replica of the application directory partition.

***** */

DWORD PrintDCFromAppPartition(LPCTSTR pszAppPartitionDN)
{
    DWORD dwRet;

    /*
    Convert the distinguished name of the partition to a DNS name so that
    the DNS name can be passed to DsGetDcName.
    */
    LPTSTR pszDNS;
    dwRet = AppPartitionDNToDNS(pszAppPartitionDN, &pszDNS);
    if(NO_ERROR == dwRet)
    {
        PDOMAIN_CONTROLLER_INFO pdci;

        /*
        Get the name of a domain controller that hosts a replica of the
        application directory partition.
        */
        dwRet = DsGetDcName(NULL,
                           pszDNS,
                           NULL,
                           NULL,
                           DS_ONLY_LDAP_NEEDED,
                           &pdci);

        if(NO_ERROR == dwRet)
        {
            // Print the DNS name of the domain controller.
            _tprintf(pdci->DomainControllerName);

            NetApiBufferFree(pdci);
        }

        FreeADsMem(pszDNS);
    }

    return dwRet;
}

```

Adding or Deleting an Application Directory Partition Replica

6/3/2022 • 2 minutes to read • [Edit Online](#)

The first replica of an application directory partition is created on the domain controller that was bound to it at creation time. Additional replicas can be created on any domain controller in the forest, not necessarily in the same domain as the initial domain controller. An application directory partition replica can only exist on a domain controller that is running Windows Server 2003 or later. For more information, see this TechNet article on [Partition Management](#).

To add a replica for an application directory partition, perform the following steps

1. Search the Partitions container for a **crossRef** object that has an **nCName** attribute value that is equal to the distinguished name of the application directory partition.
2. Bind to the **crossRef** object with delegation enabled. This is required because the **crossRef** object can only be modified on the Domain-Naming FSMO role holder. With delegation enabled, the domain controller can contact the Domain-Naming FSMO role holder using the same credentials.
3. Add the distinguished name of the **nTDSSDA** object for the domain controller that will host the new replica to the **msDS-NC-Replica-Locations** attribute of the **crossRef** object.

When the new **msDS-NC-Replica-Locations** attribute value is replicated to the new domain controller that will host a replica of the application directory partition, the KCC will be triggered to replicate the application directory partition to the target domain controller.

To remove a replica for an application directory partition, perform the same steps above to locate the **crossRef** object that represents the application directory partition and remove the value that corresponds to the domain controller from the **msDS-NC-Replica-Locations** attribute of the **crossRef** object.

When the new **msDS-NC-Replica-Locations** attribute value is replicated to the domain controller that will no longer host a replica of the application directory partition, the KCC will be triggered to remove the replica of the application directory partition on the target domain controller.

If a domain controller that is hosting an application directory partition replica is removed or demoted, the Active Directory server will automatically remove the value corresponding to the domain controller from the **msDS-NC-Replica-Locations** attribute of all **crossRef** objects.

Enumerating Replicas of an Application Directory Partition

6/3/2022 • 2 minutes to read • [Edit Online](#)

When a replica of an application directory partition is added, the distinguished name of the **nTDSDSA** object for the domain controller that will contain the replica is added to the **msDS-NC-Replica-Locations** attribute of the **crossRef** object. The **crossRef** object used represents the application directory partition.

To enumerate the replicas for an application directory partition

1. Search the Partitions container for a **crossRef** object that has an **nCName** attribute value that is equal to the distinguished name of the application directory partition.
2. Use each value of the **msDS-NC-Replica-Locations** attribute of the **crossRef** object to bind to the **nTDSDSA** object of the server.
3. Obtain the ADsPath for the parent of each **nTDSDSA** object. This is an object that represents the domain controller server. Use the ADsPath to bind to the server object.
4. Obtain the **dNSHostName** attribute value of the server object. This is a single-value property that contains the DNS name of the server.

Due to replication latency and scheduled KCC run delays, it is possible the actual active replicas for an application directory partition may not match the list of domain controllers indicated by the **msDS-NC-Replica-Locations** attribute of the **crossRef** object. A more accurate, but less efficient way to determine the actual active replicas of an application directory partition is to search for all **nTDSDSA** objects in the forest that have a **msDS-hasMasterNCs** attribute that contains the distinguished name of the application directory partition. The **msDS-hasMasterNCs** attribute contains the distinguished names of all writable directory partitions that the domain controller hosts, including application directory partitions.

Modifying Application Directory Partition Configuration

6/3/2022 • 2 minutes to read • [Edit Online](#)

An application directory partition can be modified by changing certain attributes of the **crossRef** object that represents the application directory partition. To locate the **crossRef** object that represents a particular application directory partition, search the Partitions container for a **crossRef** object that has an **nCName** attribute value that is equal to the distinguished name of the application directory partition. The distinguished name of the **crossRef** object can then be used to bind to the **crossRef** object.

The following table lists attributes of the **crossRef** object that can be modified.

ATTRIBUTE	DESCRIPTION
msDS-Replication-Notify-First-DSA-Delay	Specifies the delay, in seconds, after an originating object change before the first replication partner is notified. For more information, see Application Directory Partition Replication .
msDS-Replication-Notify-Subsequent-DSA-Delay	Specifies the delay, in seconds, between subsequent notifications to the second, third, and so on replication partners. For more information, see Application Directory Partition Replication .
msDS-SDReferenceDomain	Identifies the domain that the security subsystem uses to interpret local domain references in the nTSecurityDescriptor attributes of objects in that application directory partition. For more information, see Application Directory Partition Security .
msDS-NC-Replica-Locations	Contains the distinguished name of the nTDSDSA object for each domain controller that hosts a replica of the application directory partition. For more information, see Adding or Deleting an Application Directory Partition Replica .

Detecting the Operation Mode of a Domain

6/3/2022 • 2 minutes to read • [Edit Online](#)

In Windows 2000, a domain can run in two operation modes: mixed and native. Mixed mode should be used to include domain controllers running Windows NT 4.0 in a Windows 2000 domain. Mixed mode does not support universal groups or nested groups. If all domain controllers in the domain are running Windows 2000, you can use native mode.

To programmatically detect the operation mode of a Windows 2000 domain, read the **ntMixedDomain** property of the **domainDNS** object for that domain. A value of zero (0) means that the domain is in native mode. A value of one (1) indicates that the domain is in mixed mode. You can also use the **DsRoleGetPrimaryDomainInformation** function to get the operation mode as well as other data about the domain and its state.

To bind to the **domainDNS** object of the domain of the user account under which your application is running, use serverless binding and rootDSE to get the distinguished name for the domain and then use that distinguished name to bind to the **domainDNS** object that represents that domain. For more information about serverless binding and rootDSE, see [Serverless Binding and RootDSE](#).

For more information and a code example that shows how to programmatically detect the operation mode of a domain, see [Example Code for Determining the Operation Mode](#).

Example Code for Determining the Operation Mode

6/3/2022 • 3 minutes to read • [Edit Online](#)

This topic includes a code example that reads the **ntMixedDomain** property of a domain and determines the operation mode.

The following C++ code example reads the **ntMixedDomain** property of a domain and determines the operation mode.

```
HRESULT GetDomainMode(IADs *pDomain, BOOL *bIsMixed)
{
    HRESULT hr = E_FAIL;
    VARIANT var;
    if (pDomain)
    {
        VariantClear(&var);

        // Get the ntMixedDomain attribute.
        hr = pDomain->Get(CComBSTR("ntMixedDomain"), &var);
        if (SUCCEEDED(hr))
        {

            // Type should be VT_I4.
            if (var.vt==VT_I4)
            {

                // Zero means native mode.
                if (var.lVal == 0)
                    *bIsMixed = FALSE;

                // One means mixed mode.
                else if (var.lVal == 1)
                    *bIsMixed = TRUE;
                else
                    hr=E_FAIL;
            }
        }
        VariantClear(&var);
    }
    return hr;
}
```

The following Visual Basic code example reads the **ntMixedDomain** property of a domain and determines the operation mode.

```
' Example: Detects if a domain is in mixed mode or native mode.

Dim IADsRootDSE As IADs
Dim IADsDomain As IADs

On Error Resume Next
sComputer = InputBox("This script detects if a Windows 2000 domain " _
    & "is running in mixed or native mode." _
    & vbCrLf & vbCrLf _
    & "Specify the domain name or the name of a" _
    & " domain controller in the domain :")
```

```

sPrefix = "LDAP://" & sComputer & "/"

.....
' Bind to a ds server
.....
Set IADsRootDSE = GetObject(sPrefix & "rootDSE")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
    Exit Sub
End If
sDomain = IADsRootDSE.Get("defaultNamingContext")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get method"
    Exit Sub
End If
Set IADsDomain = GetObject(sPrefix & sDomain)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
    Exit Sub
End If

.....
' Get ntMixedDomain property
.....
sMode = IADsDomain.Get("ntMixedDomain")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get ntMixedDomain property"
    Exit Sub
End If
.....
' Get name property
.....
sName = IADsDomain.Get("name")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get name property"
    Exit Sub
End If

If (sMode = 0) Then
    sModeString = "Native"
ElseIf (sMode = 1) Then
    sModeString = "Mixed"
Else
    BailOnFailure sMode, "Invalid ntMixedDomain value: " & sMode
    Exit Sub
End If

strText = "The domain " & sName & " is in " _
    & sModeString & " mode.

MsgBox strText
.....
' Display subroutines
.....
Sub show_groups(strText, strName)
    MsgBox strText, vbInformation, "Groups on " & strName
End Sub

Sub BailOnFailure(ErrNum, ErrText)
    strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub

```

The following Visual Basic Scripting Edition code example reads the **ntMixedDomain** property of a domain and determines the operation mode.

```

' Example: Detects if a domain is in mixed mode or native mode.

Dim IADsRootDSE
Dim IADsDomain

On Error Resume Next
sComputer = InputBox("Specify the domain name or the name " _
& "of a domain controller in the domain :")

sPrefix = "LDAP://" & sComputer & "/"

.....
' Bind to a ds server
.....
Set IADsRootDSE = GetObject(sPrefix & "rootDSE")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
    Exit Sub
End If
sDomain = IADsRootDSE.Get("defaultNamingContext")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get method"
    Exit Sub
End If
Set IADsDomain = GetObject(sPrefix & sDomain)
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on GetObject method"
    Exit Sub
End If

.....
' Get ntMixedDomain property
.....
sMode = IADsDomain.Get("ntMixedDomain")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get ntMixedDomain property"
    Exit Sub
End If
.....
' Get name property
.....
sName = IADsDomain.Get("name")
If (Err.Number <> 0) Then
    BailOnFailure Err.Number, "on Get name property"
    Exit Sub
End If

If (sMode = 0) Then
    sModeString = "Native"
ElseIf (sMode = 1) Then
    sModeString = "Mixed"
Else
    BailOnFailure sMode, "Invalid ntMixedDomain value: " & sMode
    Exit Sub
End If

strText = "The domain " & sName & " is in " & sModeString & " mode."

MsgBox strText
.....
' Display subroutines
.....
Sub BailOnFailure(ErrNum, ErrText)
    strText = "Error 0x" & Hex(ErrNum) & " " & ErrText
    MsgBox strText, vbInformation, "ADSI Error"
    WScript.Quit
End Sub

```


Active Directory Domain Services Reference

6/3/2022 • 2 minutes to read • [Edit Online](#)

This section includes reference materials for programming in Microsoft Active Directory Domain Services. The topics include:

- [Constants](#)
- [Structures](#)
- [Enumerations](#)
- [Functions](#)
- [Interfaces](#)
- [Messages](#)
- [Return Values](#)
- [User Interface Mappings](#)
- [WMI Provider Reference](#)

To view the online reference pages for the Active Directory schema, see the [Active Directory Schema](#). For overview information about the Active Directory Schema from an Active Directory Domain Services perspective, see [Active Directory Schema](#).

Constants in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

This section defines flags, messages, and GUIDS used for user interfaces in Active Directory Domain Services.

- [GUIDs of User Interface Elements](#)
- [Messages Communicated through User Interfaces](#)
- [CFSTR_DS_DISPLAY_SPEC_OPTIONS](#)
- [CFSTR_DS_MULTISELECTPROPPAGE](#)
- [CFSTR_DS_PARENTHWND](#)
- [CFSTR_DS_PROPSHEETCONFIG](#)
- [CFSTR_DSOBJECTNAMES](#)
- [CFSTR_DSOP_DS_SELECTION_LIST](#)
- [CFSTR_DSPROPERTYPAGEINFO](#)
- [CFSTR_DSQUERYPARAMS](#)
- [CFSTR_DSQUERYSCOPE](#)
- [BFT Constants](#)

GUIDs of User Interface Elements

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following table lists the class identifier, expressed as a GUID, for user interface objects implemented by a directory service.

CLASS IDENTIFIER	DESCRIPTION
CLSID_DsFolderProperties	Reserved. Declared in Dsclient.h.
CLSID_DsPropertyPages	This object provides the IShellExtInit , IShellPropSheetExt , and IContextMenu interfaces for use with system-supplied directory service objects. Declared in Dsclient.h.

Messages Communicated through User Interfaces

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic lists messages that a directory service can send from a user interface.

Common Query Page Messages

The following messages are sent to a directory service query form extension page in the [CQPageProc](#) callback function:

- [CQPM_CLEARFORM](#)
- [CQPM_ENABLE](#)
- [CQPM_GETPARAMETERS](#)
- [CQPM_HANDLERSPECIFIC](#)
- [CQPM_HELP](#)
- [CQPM_INITIALIZE](#)
- [CQPM_PERSIST](#)
- [CQPM_RELEASE](#)
- [CQPM_SETDEFAULTPARAMETERS](#)

Miscellaneous Messages

The following table lists messages that a directory service can send.

MESSAGE	VALUE	DESCRIPTION
DSPROP_ATTRCHANGED_MSG	"DsPropAttrChanged"	A message sent for synchronizing property pages and the directory service administration tools, declared in Dsclient.h.
DSQPM_GETCLASSLIST	CQPM_HANDLERSPECIFIC+0	A page message sent to the form pages for retrieving a list of classes for query, used by the field selector and property well to build its list of display classes. wParam = flags and lParam = LPLPDSQUERYCLASSLIST, declared in Dsquery.h.
DSQPM_HELPTOPICS	(CQPM_HANDLERSPECIFIC+1)	A page message sent to the form pages for handling the "Help Topics" request. wParam = 0, lParam = hWndParent, declared in Dsquery.h.

CQPM_CLEARFORM message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **CQPM_CLEARFORM** message is sent to the [CQPageProc](#) callback function of a query for extension page when the contents of the page should be reset to the default values.

Parameters

wParam

Not used.

lParam

Not used.

Return value

Returns **S_OK** if successful or a standard **HRESULT** error code otherwise.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

CQPM_ENABLE message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **CQPM_ENABLE** message is sent to the [CQPageProc](#) callback function of a query form extension page to enable or disable the page.

Parameters

wParam

Contains zero to disable the page or nonzero to enable the page.

lParam

Not used.

Return value

The return value for this message is ignored.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

CQPM_GETPARAMETERS message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **CQPM_GETPARAMETERS** message is sent to the [CQPageProc](#) callback function of a query form extension page to retrieve data about the query performed by the page.

Parameters

wParam

Not used.

lParam

Pointer to a [LPDSQUERYPARAMS](#) value that receives data about the query performed by the page. If this parameter is **NULL**, the DSQUERYPARAMS structure must be allocated by the extension using the [CoTaskMemAlloc](#) function.

Return value

Returns **S_OK** if successful or a standard **HRESULT** error code otherwise.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

[DSQUERYPARAMS](#)

[CoTaskMemAlloc](#)

CQPM_HANDLERSPECIFIC message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The CQPM_HANDLERSPECIFIC message is the base value used for messages that are private to the query handler. The query handler should add this value to private messages to ensure that message collisions do not occur.

Parameters

wParam

Contains additional message data. The content of this parameter is defined by the query handler.

lParam

Contains additional message data. The content of this parameter is defined by the query handler.

Return value

The meaning of the return value is defined by the query handler.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

CQPM_HELP message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The CQPM_HELP message is sent to the [CQPageProc](#) callback function of a query form extension page to allow the page extension to display context-sensitive help for the page. If possible, the query page extension should display context-sensitive help in response to this event.

Parameters

wParam

Not used.

lParam

Pointer to a [HELPINFO](#) structure that contains additional data about the menu item, control, dialog box, or window for which context-sensitive help is requested.

Return value

The return value for this message is ignored.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

[HELPINFO](#)

CQPM_INITIALIZE message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The CQPM_INITIALIZE message is sent to the [CQPageProc](#) callback function of a query form extension page when the page is added to a form.

Parameters

wParam

Not used.

lParam

Not used.

Return value

The return value for this message is ignored.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

CQPM_PERSIST message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The CQPM_PERSIST message is sent to the [CQPageProc](#) callback function of a query form extension page to allow the page to read or write query data from persistent memory.

Parameters

wParam

Contains nonzero to read the query data or zero to write the query data.

lParam

Pointer to an [IPersistQuery](#) interface that the query data should be read from or written to.

Return value

Returns S_OK if successful or a standard HRESULT error code otherwise.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

[IPersistQuery](#)

CQPM_RELEASE message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The CQPM_RELEASE message is sent to the [CQPageProc](#) callback function of a query form extension page when the page is about to be unloaded.

Parameters

wParam

Not used.

lParam

Not used.

Return value

The return value for this message is ignored.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

CQPM_SETDEFAULTPARAMETERS message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **CQPM_SETDEFAULTPARAMETERS** message is sent to the [CQPageProc](#) callback function of a query form extension page to set alternate default parameters for the page.

Parameters

wParam

Contains nonzero if the page is the default page or zero otherwise.

lParam

Pointer to an [OPENQUERYWINDOW](#) structure that contains data about the directory service query dialog box.

Return value

The return value for this message is ignored.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Cmnquery.h

See also

[CQPageProc](#)

[OPENQUERYWINDOW](#)

CFSTR_DS_DISPLAY_SPEC_OPTIONS

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DS_DISPLAY_SPEC_OPTIONS

"DsDisplaySpecOptions"

The Active Directory Users and Computers, the Active Directory Sites and Services, and the Active Directory Domains and Trusts snap-ins support the **CFSTR_DS_DISPLAY_SPEC_OPTIONS** clipboard format. The **CFSTR_DS_DISPLAY_SPEC_OPTIONS** clipboard format provides an **HGLOBAL** that contains a **DSDISPLAYSPECOPTIONS** structure. The **DSDISPLAYSPECOPTIONS** contains configuration data for use by the extension.

CFSTR_DSDISPLAYSPECOPTIONS

The **CFSTR_DSDISPLAYSPECOPTIONS** clipboard format is identical to the **CFSTR_DS_DISPLAY_SPEC_OPTIONS** clipboard format.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	DSClient.h

See also

[DSDISPLAYSPECOPTIONS](#)

CFSTR_DS_MULTISELECTPROPPAGE

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DS_MULTISELECTPROPPAGE

"DsAdminMultiSelectClipFormat"

The CFSTR_DS_MULTISELECTPROPPAGE clipboard format provides a global memory handle (**HGLOBAL**) that contains a null-terminated Unicode string that contains a unique name that is used to create the notification object for a multi-selection property sheet extension.

NOTE

This clipboard format value is not defined in a published header file. To use this clipboard format value, you must define it yourself in the exact format shown.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

See also

[Implementing the Property Page COM Object](#)

CFSTR_DS_PARENTHWND

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DS_PARENTHWND

"DsAdminParentHwndClipFormat"

The CFSTR_DS_PARENTHWND clipboard format provides a global memory handle (HGLOBAL) that contains the window handle (HWND) of the main MMC window.

NOTE

This clipboard format value is not defined in a published header file. To use this clipboard format value, define it in the exact format shown.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

CFSTR_DS_PROPSHEETCONFIG

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DS_PROPSHEETCONFIG

"DsPropSheetCfgClipFormat"

The CFSTR_DS_PROPSHEETCONFIG clipboard format provides a global memory handle (**HGLOBAL**) that contains a [PROPSHEETCFG](#) structure.

NOTE

This clipboard format value is not defined in a published header file. To use this clipboard format value, you must define it yourself in the exact format shown.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

See also

[PROPSHEETCFG](#)

CFSTR_DSOBJECTNAMES

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DSOBJECTNAMES

"DsObjectNames"

The CFSTR_DSOBJECTNAMES clipboard format is supported by the [IDataObject](#) provided by the [ICommonQuery::OpenQueryWindow](#) method. The CFSTR_DSOBJECTNAMES clipboard format provides a global memory handle (HGLOBAL) that contains [DSOBJECTNAMES](#) structure.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	DSClient.h

See also

[IDataObject](#)

[ICommonQuery::OpenQueryWindow](#)

[DSOBJECTNAMES](#)

CFSTR_DSOP_DS_SELECTION_LIST

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DSOP_DS_SELECTION_LIST

"CFSTR_DSOP_DS_SELECTION_LIST"

The CFSTR_DSOP_DS_SELECTION_LIST clipboard format is provided by the [IDataObject](#) obtained by calling [IDsObjectPicker::InvokeDialog](#). The CFSTR_DSOP_DS_SELECTION_LIST clipboard format provides an HGLOBAL that contains a [DS_SELECTION_LIST](#) structure. The DS_SELECTION_LIST structure contains data about the items selected in a [Directory Object Picker](#) dialog box.

Requirements

REQUIREMENT	VALUE
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Objsel.h

See also

[DS_SELECTION_LIST](#)

[IDsObjectPicker::InvokeDialog](#)

[Directory Object Picker](#)

CFSTR_DSPROPERTYPAGEINFO

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DSPROPERTYPAGEINFO

"DsProp PageInfo"

The CFSTR_DSPROPERTYPAGEINFO clipboard format provides an HGLOBAL that contains a **DSPROPERTYPAGEINFO** structure. The **DSPROPERTYPAGEINFO** structure contains the optional string that the extension added to the **adminPropertySheet** and/or **shellPropertySheet** parameter values when the extension was registered. For more information about how this string is set, see [Registering the Property Page COM Object in a Display Specifier](#).

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	DSClient.h

See also

[DSPROPERTYPAGEINFO](#)

[Registering the Property Page COM Object in a Display Specifier](#)

CFSTR_DSQUERYPARAMS

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DSQUERYPARAMS

"DsQueryParameters"

The CFSTR_DSQUERYPARAMS clipboard format is supported by the [IDataObject](#) provided by the [ICommonQuery::OpenQueryWindow](#) method. The CFSTR_DSQUERYPARAMS clipboard format provides a global memory handle (HGLOBAL) that contains a [DSQUERYPARAMS](#) structure.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	DSQuery.h

See also

[IDataObject](#)

[ICommonQuery::OpenQueryWindow](#)

[DSQUERYPARAMS](#)

CFSTR_DSQUERYSCOPE

6/3/2022 • 2 minutes to read • [Edit Online](#)

CFSTR_DSQUERYSCOPE

"DsQueryScope"

The CFSTR_DSQUERYSCOPE clipboard format is supported by the [IDataObject](#) provided by the [ICommonQuery::OpenQueryWindow](#) method. The CFSTR_DSQUERYSCOPE clipboard format provides a global memory handle (HGLOBAL) that contains a string that specifies the query scope.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	DSQuery.h

See also

[IDataObject](#)

[ICommonQuery::OpenQueryWindow](#)

BFT Constants

6/3/2022 • 2 minutes to read • [Edit Online](#)

The BFT constants are used as bit flags to identify different file types in an Active Directory Domain Services backup.

BFT_LOG_DIRECTORY

0x20

The file belongs in the log directory.

BFT_DATABASE_DIRECTORY

0x40

The file belongs in the database directory.

BFT_DIRECTORY

0x80

The path specified is a directory and not a file name.

BFT_LOG

0x21

Specifies a log file that belongs in the log directory.

BFT_LOG_DIR

0x22

The file is the path of the log directory.

BFT_CHECKPOINT_DIR

0x23

The file is the path of the checkpoint directory.

BFT_NTDS_DATABASE

0x44

The file is a directory service database that belongs in the database directory.

BFT_PATCH_FILE

0x25

The file is a patch file that belongs in the log directory.

BFT_UNKNOWN

0x0F

The file cannot be recognized. The file does not coincide with the known file names and file types.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h

Structures in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following are categories of structures in Active Directory Domain Services:

- [Admin Structures in Active Directory Domain Services](#)
- [Display Structures in Active Directory Domain Services](#)
- [MMC Property Page Structures in Active Directory Domain Services](#)
- [Domain Controller and Replication Management Structures](#)
- [Directory Service Structures](#)
- [Directory Backup Structures](#)
- [Object Picker Dialog Box Structures](#)

Admin Structures in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following structures are used with the Active Directory Domain Services administrative MMC snap-ins.

- [DSA_NEWORDISPINFO](#)

Display Structures in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Display functions and interfaces in Active Directory Domain Services use the following structures to contain display data about directory user interface elements:

- [CQFORM](#)
- [CQPAGE](#)
- [DSA_SEC_PAGE_INFO](#)
- [DOMAINDESC](#)
- [DOMAINTREE](#)
- [DSBITEM](#)
- [DSBROWSEINFO](#)
- [DSCLASSCREATIONINFO](#)
- [DSCOLUMN](#)
- [DSDISPLAYSPECOPTIONS](#)
- [DSOBJECT](#)
- [DSOBJECTNAMES](#)
- [DSPROPERTYPAGEINFO](#)
- [DSQUERYCLASSTLIST](#)
- [DSQUERYINITPARAMS](#)
- [DSQUERYPARAMS](#)
- [OPENQUERYWINDOW](#)
- [PROPSHEETCFG](#)

DSA_SEC_PAGE_INFO structure

6/3/2022 • 2 minutes to read • [Edit Online](#)

The DSA_SEC_PAGE_INFO structure is used with the [WM_ADSPROP_SHEET_CREATE](#) and [WM_DSA_SHEET_CREATE_NOTIFY](#) messages to define a secondary property sheet in an Active Directory MMC snap-in.

NOTE

This structure is not defined in a published header file. To use this structure, define it in the exact format shown.

Syntax

```
typedef struct _DSA_SEC_PAGE_INFO {  
    HWND      hwndParentSheet;  
    DWORD     offsetTitle;  
    DSOBJECTNAMES dsObjectNames;  
} DSA_SEC_PAGE_INFO, *PDSA_SEC_PAGE_INFO;
```

Members

hwndParentSheet

Contains the window handle of the parent of the secondary property sheet.

offsetTitle

Contains the offset, in bytes, from the start of the DSA_SEC_PAGE_INFO structure to a NULL-terminated, Unicode string that contains the title of the secondary property sheet.

dsObjectNames

Contains a [DSOBJECTNAMES](#) structure that defines the secondary property sheet. Only one secondary property sheet can be created at a time, so the DSOBJECTNAMES structure can only contain one [DSOBJECT](#) structure.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

See also

[WM_ADSPROP_SHEET_CREATE](#)

[WM_DSA_SHEET_CREATE_NOTIFY](#)

DSOBJECTNAMES

DSOBJECT

PROPSHEETCFG structure

6/3/2022 • 2 minutes to read • [Edit Online](#)

The PROPSHEETCFG structure is used to contain property sheet configuration data. This structure is contained in the [CFSTR_DS_PROPSHEETCONFIG](#) clipboard format.

NOTE

This structure is not defined in a published header file. To use this structure, you must define it yourself in the exact format shown.

Syntax

```
typedef struct {
    LONG_PTR lNotifyHandle;
    HWND      hwndParentSheet;
    HWND      hwndHidden;
    WPARAM    wParamSheetClose;
} PROPSHEETCFG, *PPROPSHEETCFG;
```

Members

INotifyHandle

Contains the notification handle. This is identical to the handle passed for the *handle* parameter in the [IExtendPropertySheet2::CreatePropertyPages](#) method.

hwndParentSheet

Contains the window handle of the parent property sheet.

hwndHidden

Contains the handle of the hidden window.

wParamSheetClose

Contains an application-defined 32-bit value. This value is passed back to the application in the *wParam* of the [WM_DSA_SHEET_CLOSE_NOTIFY](#) message.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

See also

[CFSTR_DS_PROPSHEETCONFIG](#)

WM_DSA_SHEET_CLOSE_NOTIFY

MMC Property Page Structures in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory Manager MMC snap-in uses property sheets to display the attributes of objects in Active Directory Domain Services. The following structures help provide the infrastructure to control the operations performed on the pages of these property sheets:

- [ADSPROPERERROR](#)
- [ADSPROPINITPARAMS](#)

Domain Controller and Replication Management Structures

6/3/2022 • 2 minutes to read • [Edit Online](#)

The domain controller and replication management functions use the following structures:

- [DS_DOMAIN_CONTROLLER_INFO_1](#)
- [DS_DOMAIN_CONTROLLER_INFO_2](#)
- [DS_DOMAIN_CONTROLLER_INFO_3](#)
- [DS_NAME_RESULT](#)
- [DS_NAME_RESULT_ITEM](#)
- [DS_REPL_ATTR_META_DATA](#)
- [DS_REPL_ATTR_META_DATA_2](#)
- [DS_REPL_ATTR_META_DATA_BLOB](#)
- [DS_REPL_ATTR_VALUE_META_DATA](#)
- [DS_REPL_ATTR_VALUE_META_DATA_2](#)
- [DS_REPL_ATTR_VALUE_META_DATA_EXT](#)
- [DS_REPL_CURSOR](#)
- [DS_REPL_CURSOR_2](#)
- [DS_REPL_CURSOR_3](#)
- [DS_REPL_CURSOR_BLOB](#)
- [DS_REPL_CURSORS](#)
- [DS_REPL_CURSORS_2](#)
- [DS_REPL_CURSORS_3](#)
- [DS_REPL_KCC_DSA_FAILURE](#)
- [DS_REPL_KCC_DSA_FAILURES](#)
- [DS_REPL_KCC_DSA_FAILUREW_BLOB](#)
- [DS_REPL_NEIGHBOR](#)
- [DS_REPL_NEIGHBORS](#)
- [DS_REPL_NEIGHBORW_BLOB](#)
- [DS_REPL_OBJ_META_DATA](#)
- [DS_REPL_OBJ_META_DATA_2](#)
- [DS_REPL_OP](#)
- [DS_REPL_OPW_BLOB](#)
- [DS_REPL_PENDING_OPS](#)
- [DS_REPL_QUEUE_STATISTICSW](#)
- [DS_REPL_QUEUE_STATISTICSW_BLOB](#)
- [DS_REPL_VALUE_META_DATA](#)
- [DS_REPL_VALUE_META_DATA_2](#)
- [DS_REPL_VALUE_META_DATA_EXT](#)
- [DS_REPL_VALUE_META_DATA_BLOB](#)
- [DS_REPL_VALUE_META_DATA_BLOB_EXT](#)
- [DS_REPSYNCALL_ERRINFO](#)
- [DS_REPSYNCALL_SYNC](#)

- [DS_REPSYNCALL_UPDATE](#)
- [DS_SCHEMA_GUID_MAP](#)
- [DS_SITE_COST_INFO](#)
- [SCHEDULE](#)
- [SCHEDULE_HEADER](#)

Related topics

[Structures in Active Directory Domain Services](#)

Directory Service Structures

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic lists structures used by functions in Active Directory Domain Services.

- [DOMAIN_CONTROLLER_INFO](#)
- [DS_DOMAIN_TRUSTS](#)
- [DSROLE_OPERATION_STATE_INFO](#)
- [DSROLE_PRIMARY_DOMAIN_INFO_BASIC](#)
- [DSROLE_UPGRADE_STATUS_INFO](#)

Related topics

[Structures in Active Directory Domain Services](#)

Directory Backup Structures

6/3/2022 • 2 minutes to read • [Edit Online](#)

The directory backup functions use the following structure:

- [EDB_RSTMAP](#)

Related topics

[Active Directory Structures](#)

EDB_RSTMAP structure

6/3/2022 • 2 minutes to read • [Edit Online](#)

The **EDB_RSTMAP** structure is used with the [DsRestoreRegister](#) function to map a backed up database to a new database.

Syntax

```
typedef struct {
    LPTSTR szDatabaseName;
    LPTSTR szNewDatabaseName;
} EDB_RSTMAP, *PEDB_RSTMAP;
```

Members

szDatabaseName

Pointer to a null-terminated string that contains the name of the database when it was backed up.

szNewDatabaseName

Pointer to a null-terminated string that contains the new name of the database, including its new location, when applicable.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Unicode and ANSI names	EDB_RSTMAPW (Unicode) and EDB_RSTMAPA (ANSI)

See also

[DsRestoreRegister](#)

[Directory Backup Structures](#)

Object Picker Dialog Box Structures

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following structures are used with the object picker dialog boxes:

- [DS_SELECTION](#)
- [DS_SELECTION_LIST](#)
- [DSOP_FILTER_FLAGS](#)
- [DSOP_INIT_INFO](#)
- [DSOP_SCOPE_INIT_INFO](#)
- [DSOP_UPLEVEL_FILTER_FLAGS](#)

Related topics

[CFSTR_DSOP_DS_SELECTION_LIST](#)

[IDsObjectPicker::InvokeDialog](#)

[Directory Object Picker](#)

Enumerations in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following enumerations are used for Microsoft Active Directory Domain Services.

- [DS_MANGLE_FOR](#)
- [DS_NAME_ERROR](#)
- [DS_NAME_FLAGS](#)
- [DS_NAME_FORMAT](#)
- [DS_REPL_INFO_TYPE](#)
- [DS_REPL_OP_TYPE](#)
- [DS_REPSYNCALL_ERROR](#)
- [DS_REPSYNCALL_EVENT](#)
- [DS_SPN_NAME_TYPE](#)
- [DS_SPN_WRITE_OP](#)
- [DSROLE_MACHINE_ROLE](#)
- [DSROLE_OPERATION_STATE](#)
- [DSROLE_PRIMARY_DOMAIN_INFO_LEVEL](#)
- [DSROLE_SERVER_STATE](#)

Functions in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following are categories of functions in Active Directory Domain Services:

- [Display Functions in Active Directory Domain Services](#)
- [MMC Property Page Functions in Active Directory Domain Services](#)
- [Domain Controller and Replication Management Functions](#)
- [Directory Service Functions](#)
- [Directory Backup Functions](#)

Display Functions in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following functions work with display features provided in Active Directory Domain Services:

- [BFFCallBack](#)
- [CQAddFormsProc](#)
- [CQAddPagesProc](#)
- [CQPageProc](#)
- [DsBrowseForContainer](#)
- [DSEnumAttributesCallback](#)
- [DsGetFriendlyClassName](#)
- [DsGetIcon](#)

MMC Property Page Functions in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

A property sheet extension in Active Directory Domain Services uses the following functions to create and work with the notification object. For more information, see [Implementing the Property Page COM Object](#).

- [ADsPropCheckIfWritable](#)
- [ADsPropCreateNotifyObj](#)
- [ADsPropGetInitInfo](#)
- [ADsPropSendErrorMessage](#)
- [ADsPropSetHwnd](#)
- [ADsPropSetHwndWithTitle](#)
- [ADsPropShowErrorDialog](#)

Domain Controller and Replication Management Functions

6/3/2022 • 2 minutes to read • [Edit Online](#)

The domain controller (DC) and replication management functions provide tools for finding data about a DC, converting the names of network objects between different formats, manipulating service principal names (SPNs) and directory service agents (DSAs), and managing replication of servers. The following functions enable developers to work with domain controllers, replication, and the directory service:

- [DsAddSidHistory](#)
- [DsBind](#)
- [DsBindingSetTimeout](#)
- [DsBindToISTG](#)
- [DsBindWithCred](#)
- [DsBindWithSpn](#)
- [DsBindWithSpnEx](#)
- [DsClientMakeSpnForTargetServer](#)
- [DsCrackNames](#)
- [DsCrackSpn](#)
- [DsCrackUnquotedMangledRdn](#)
- [DsFreeDomainControllerInfo](#)
- [DsFreeNameResult](#)
- [DsFreePasswordCredentials](#)
- [DsFreeSchemaGuidMap](#)
- [DsFreeSpnArray](#)
- [DsGetDomainControllerInfo](#)
- [DsGetRdnW](#)
- [DsGetSpn](#)
- [DsInheritSecurityIdentity](#)
- [DsIsMangledDn](#)
- [DsIsMangledRdnValue](#)
- [DsListDomainsInSite](#)
- [DsListInfoForServer](#)
- [DsListRoles](#)
- [DsListServersForDomainInSite](#)
- [DsListServersInSite](#)
- [DsListSites](#)
- [DsMakePasswordCredentials](#)
- [DsMakeSpn](#)
- [DsMapSchemaGuids](#)
- [DsQuerySitesByCost](#)
- [DsQuerySitesFree](#)
- [DsQuoteRdnValue](#)
- [DsRemoveDsDomain](#)

- [DsRemoveDsServer](#)
- [DsReplicaAdd](#)
- [DsReplicaConsistencyCheck](#)
- [DsReplicaDel](#)
- [DsReplicaFreeInfo](#)
- [DsReplicaGetInfo](#)
- [DsReplicaGetInfo2](#)
- [DsReplicaModify](#)
- [DsReplicaSync](#)
- [DsReplicaSyncAll](#)
- [DsReplicaUpdateRefs](#)
- [DsReplicaVerifyObjects](#)
- [DsServerRegisterSpn](#)
- [DsUnBind](#)
- [DsUnquoteRdnValue](#)
- [DsWriteAccountSpn](#)
- [SyncUpdateProc](#)

Most of these functions require a handle bound to the directory service. The [DsBind](#) and [DsBindWithCred](#) functions start an RPC session with a particular domain controller, then they bind a handle to the directory service and return the handle. When the handle is no longer required, use the [DsUnBind](#) function to end the RPC session and unbind the handle.

Replication occurs between a source server and a destination server. A source server maintains a list of destination servers to which it should replicate, and a destination server maintains a list of source servers from which it receives replication. Use the [DsReplicaAdd](#) function to add to the list of source servers on a destination server, and use the [DsReplicaDel](#) function to remove references from the source server list on a destination server. The [DsReplicaModify](#) function may be used to change an existing source server reference on a destination server. To change the list of destination servers on a source server, use the [DsReplicaUpdateRefs](#) function.

Actual replication is performed by the [DsReplicaSync](#) and [DsReplicaSyncAll](#) functions. The [DsReplicaSync](#) function synchronizes a specific destination server with a single source server. Use the [DsReplicaSyncAll](#) function to synchronize a destination server with all other servers in the site.

Directory Service Functions (AD DS)

6/3/2022 • 2 minutes to read • [Edit Online](#)

The directory service functions provide a utility for locating a domain controller (DC) in a Windows NT or Windows 2000 domain. The architecture interacts with clients as well as servers in all versions of Windows NT and Windows 2000. The following functions enable developers to work with the domain controller and domain membership in the directory service:

- [DsAddressToSiteNames](#)
- [DsAddressToSiteNamesEx](#)
- [DsDeregisterDnsHostRecords](#)
- [DsEnumerateDomainTrusts](#)
- [DsGetDcClose](#)
- [DsGetDcName](#)
- [DsGetDcNext](#)
- [DsGetDcOpen](#)
- [DsGetDcSiteCoverage](#)
- [DsGetForestTrustInformationW](#)
- [DsGetSiteName](#)
- [DsMergeForestTrustInformationW](#)
- [DsRoleFreeMemory](#)
- [DsRoleGetPrimaryDomainInformation](#)
- [DsValidateSubnetName](#)

The DC locator, [DsGetDcName](#), is implemented by the Netlogon service. Each DC registers its DNS name on the DNS server and its NetBIOS name using a transport-specific mechanism, for example, in WINS. The DC locator looks up the name, then sends a datagram to, or pings, the DC that registered the name. For NetBIOS domain names, the datagram is a mailslot message. For DNS domain names, the datagram is an LDAP UDP search. Each such DC responds indicating that it is currently operational. The first DC to respond is returned to the caller.

The returned DC is cached, so that subsequent callers need not repeat the preceding algorithm, and to encourage all callers to use that same DC. This ensures that a single client has a consistent view of the contents of the DC.

When searching for a DC by DNS domain name, the DC locator attempts to find a DC in the "closest" site. Each DC registers additional DNS records indicating what site that the DC is in and what sites the DC includes. The DC locator first searches for this site-specific DNS record before searching for the DNS record that is not site-specific, thus preferring a DC in that site. When the DC locator sends a datagram to the DC, the DC looks up the IP address of the client in the Configuration/Sites/Subnet container of the DS to find a subnet object. The [siteObject](#) property of the subnet object defines the name of the site that contains the client. The DC responds to the ping with the name of the site that contains the client, along with an indicator of whether this DC covers that site. If the DC does not include that site and the DC locator has not yet attempted to find a DC in that site, the DC locator tries again to find a DC in the site.

To find the name of the site containing the client, use the [DsGetSiteName](#) function. The names of the objects in the Configuration/Sites/Subnet container must be valid subnet names. The [DsValidateSubnetName](#) function indicates whether a specified subnet name is valid.

Directory Backup Functions

6/3/2022 • 2 minutes to read • [Edit Online](#)

[These functions are available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The following functions are used with the backup and restore operations in Active Directory Domain Services:

- [DsBackupClose](#)
- [DsBackupEnd](#)
- [DsBackupFree](#)
- [DsBackupGetBackupLogs](#)
- [DsBackupGetDatabaseNames](#)
- [DsBackupOpenFile](#)
- [DsBackupPrepare](#)
- [DsBackupRead](#)
- [DsBackupTruncateLogs](#)
- [DsIsNTDSOnline](#)
- [DsRestoreEnd](#)
- [DsRestoreGetDatabaseLocations](#)
- [DsRestorePrepare](#)
- [DsRestoreRegister](#)
- [DsRestoreRegisterComplete](#)
- [DsSetAuthIdentity](#)
- [DsSetCurrentBackupLog](#)

DsBackupClose function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupClose** function closes a backup file opened with the **DsBackupOpenFile** function. For each backup handle, only one file can be opened at a time, so this function closes the currently open file.

Syntax

```
HRESULT DsBackupClose(  
    _In_ HBC hbc  
)
```

Parameters

hbc [in]

Contains the backup context handle obtained with the **DsBackupPrepare** function.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists other possible error codes.

ERROR_INVALID_PARAMETER

hbc is not valid.

hrInvalidHandle

No file is currently open.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib

REQUIREMENT	VALUE
DLL	Ntdsbcli.dll

See also

[DsBackupOpenFile](#)

[DsBackupPrepare](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsBackupEnd function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupEnd** function is called to terminate a backup operation.

Syntax

```
HRESULT DsBackupEnd(  
    _In_ HBC hbc  
)
```

Parameters

hbc [in]

Contains the backup context handle obtained with the [DsBackupPrepare](#) function.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists other possible error codes.

ERROR_INVALID_PARAMETER

hbc is not valid.

Remarks

The **DsBackupEnd** function closes outstanding binding handles and performs a cleanup after a successful or unsuccessful backup attempt.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib

REQUIREMENT	VALUE
DLL	Ntdsbcli.dll

See also

[DsSetAuthIdentity](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsBackupFree function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupFree** function releases memory allocated by the Active Directory Domain Services backup and restore functions. The following functions allocate memory that must be released with the **DsBackupFree** function.

- [DsBackupGetBackupLogs](#)
- [DsBackupGetDatabaseNames](#)
- [DsBackupPrepare](#)
- [DsRestoreGetDatabaseLocations](#)

Syntax

```
VOID NTDSBCLI_API DsBackupFree(
    _In_ PVOID pvBuffer
);
```

Parameters

pvBuffer [in]

Pointer to the memory buffer to be freed.

Return value

This function does not return a value.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll

See also

[DsBackupGetBackupLogs](#)

[DsBackupGetDatabaseNames](#)

[DsBackupPrepare](#)

[DsRestoreGetDatabaseLocations](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsBackupGetBackupLogs function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupGetBackupLogs** function obtains the list of log files that must be backed up for the given backup context.

Syntax

```
HRESULT DsBackupGetBackupLogs(
    _In_ HBC      hbc,
    _Out_ LPTSTR  *pszBackupLogFiles,
    _Out_ LPDWORD pcbSize
);
```

Parameters

hbc [in]

Contains the backup context handle obtained with the [DsBackupPrepare](#) function.

pszBackupLogFiles [out]

Pointer to a string pointer that receives the list of log file names as UNC paths. Initialize this value to **NULL** before calling **DsBackupGetBackupLogs**.

This list receives a double null-terminated list of single null-terminated strings.

This buffer is allocated by the **DsBackupGetBackupLogs** function and must be freed when no longer required by calling the [DsBackupFree](#) function.

The first character of each of the file names contains one of the [BFT Constants](#) that identifies the type of name.

pcbSize [out]

Pointer to **DWORD** value that receives the size, in bytes, of the *pszBackupLogFiles* buffer.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists other possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

ERROR_INVALID_PARAMETER

hbc, *pszBackupLogFiles*, or *pcbSize* is invalid.

ERROR_NOT_ENOUGH_MEMORY

A memory allocation failure occurred.

Remarks

The **DsBackupGetBackupLogs** function provides a list of the log files necessary for a backup. A full backup consists of the database files provided by the **DsBackupGetDatabaseNames** function and the log files. Incremental backups of Active Directory servers are not supported.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsBackupGetBackupLogsW (Unicode) and DsBackupGetBackupLogsA (ANSI)

See also

[DsBackupFree](#)

[DsBackupGetDatabaseNames](#)

[BFT Constants](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsBackupGetDatabaseNames function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupGetDatabaseNames** function obtains the list of database files to be backed up for the given backup context.

Syntax

```
HRESULT DsBackupGetDatabaseNames(
    _In_ HBC     hbc,
    _Out_ LPTSTR *pszAttachmentInfo,
    _Out_ LPDWORD pcbSize
);
```

Parameters

hbc [in]

Contains the backup context handle obtained with the [DsBackupPrepare](#) function.

pszAttachmentInfo [out]

Pointer to a string pointer that receives the list of database file names as UNC paths. This value must be initialized to **NULL** prior to calling **DsBackupGetDatabaseNames**.

This list receives a double null-terminated list of single null-terminated strings.

This buffer is allocated by the **DsBackupGetDatabaseNames** function and must be freed when it is no longer required by calling the [DsBackupFree](#) function.

The first character of each file name contains one of the [BFT Constants](#) that identifies the type of name. The [DsRestoreGetDatabaseLocations](#) function only supplies the following types of names.

BFT_NTDS_DATABASE

The file is an NTDS database file. This file should be copied to the file identified as **BFT_NTDS_DATABASE** when the data is restored.

BFT_LOG

The file is a log file. All log files are copied to the directory identified as **BFT_LOG_DIR** when the data is restored.

BFT_PATCH_FILE

The file is a patch file. All patch files are copied to the directory identified as **BFT_CHECKPOINT_DIR** when the data is restored.

pcbSize [out]

Pointer to **DWORD** value that receives the size, in bytes, of the *pszAttachmentInfo* buffer.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists other possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

ERROR_INVALID_PARAMETER

hbc, *pszAttachmentInfo*, or *pcbSize* are invalid.

ERROR_NOT_ENOUGH_MEMORY

A memory allocation failure occurred.

Remarks

The [DsBackupGetDatabaseNames](#) function provides a list of the database files necessary for a backup. A full backup consists of the database files and the log files provided by the [DsBackupGetBackupLogs](#) function. Incremental backups of Active Directory servers are not supported.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsBackupGetDatabaseNamesW (Unicode) and DsBackupGetDatabaseNamesA (ANSI)

See also

[DsBackupPrepare](#)

[DsBackupFree](#)

[DsBackupGetBackupLogs](#)

[BFT Constants](#)

Backing Up an Active Directory Server

Directory Backup Functions

DsBackupOpenFile function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupOpenFile** function opens the specified file and performs the client and server operations necessary to prepare the file for backup.

Syntax

```
HRESULT DsBackupOpenFile(
    _In_ HBC          hbc,
    _In_ LPCTSTR      szAttachmentName,
    _In_ DWORD        cbReadHintSize,
    _Out_ LARGE_INTEGER *pliFileSize
);
```

Parameters

hbc [in]

Contains the backup context handle obtained with the [DsBackupPrepare](#) function.

szAttachmentName [in]

Pointer to a null-terminated string that specifies the name of the backup file to open.

cbReadHintSize [in]

Contains the possible size, in bytes, of the buffer passed as the *pvBuffer* argument in the [DsBackupRead](#) function. The backup functions use this value as a hint to optimize the network traffic. This value must be a multiple of 8192 and must be greater than or equal to 24576.

pliFileSize [out]

Pointer to a **LARGE_INTEGER** value that receives the size, in bytes, of the backup file opened.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists other possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

ERROR_INVALID_PARAMETER

hbc, *szAttachmentName*, or *pliFileSize* are invalid.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsBackupOpenFileW (Unicode) and DsBackupOpenFileA (ANSI)

See also

[DsBackupRead](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsBackupPrepare function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupPrepare** function prepares the directory on the specified server for the online backup and returns a backup context handle used in subsequent calls to other backup functions.

Syntax

```
HRESULT DsBackupPrepare(
    _In_   LPCTSTR szBackupServer,
    _In_   ULONG    grbit,
    _In_   ULONG    btBackupType,
    _Out_  PVOID    *ppvExpiryToken,
    _Out_  LPDWORD  pcbExpiryTokenSize,
    _Out_  HBC      *phbc
);
```

Parameters

szBackupServer [in]

Pointer to a null-terminated string that contains the name of the server to backup. Preceding backslashes are optional. The server must be the same computer that this function is called from. The server name cannot contain an underscore (_) character. An example of a server name is "\\server1".

grbit [in]

Determines if the log files will be backed up. This value should always be 0 because incremental backups are not supported.

btBackupType [in]

Specifies the type of backup. This can be one of the following values.

BACKUP_TYPE_FULL

Specifies a full backup. The complete directory (DIT, log files, and update files) are backed up. All data is backed up and transaction log files are truncated. Only full backups are supported.

BACKUP_TYPE_LOGS_ONLY

This value is not supported. Specifies that only the database logs, and not the database itself, will be backed up. This is normally used when performing a differential or incremental backup.

BACKUP_TYPE_INCREMENTAL

This value is not supported. **DsBackupPrepare** returns **ERROR_INVALID_PARAMETER**.

ppvExpiryToken [out]

Pointer to a **PVOID** value that receives a pointer to an expiry token associated with this backup. *pcbExpiryTokenSize* receives the size, in bytes, of this data. The caller must save the contents of this token with the backup because the token must be passed to [DsRestorePrepare](#) when attempting to restore data. After the token has been stored and is no longer required, the caller should free the allocated memory using [DsBackupFree](#).

pcbExpiryTokenSize [out]

Pointer to a **DWORD** value that receives the size, in bytes, of the token in *ppvExpiryToken*.

phbc [out]

Pointer to an **HBC** value that receives the handle for the backup. This handle is used when calling other Directory Service backup functions, such as [DsBackupOpenFile](#) and [DsBackupEnd](#).

Return value

Returns **S_OK** if the function is successful or an error code otherwise. The following list lists other possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

ERROR_INVALID_PARAMETER

szBackupServer or *phbcBackupContext* are not valid.

ERROR_NOT_ENOUGH_MEMORY

A memory allocation failure occurred.

hrCouldNotConnect

The server in *szBackupServer* could not be found, is not a domain controller or *szBackupServer* is not formatted correctly. This value is defined in ntdsbmsg.h.

hrInvalidParam

ppvExpiryToken and/or *pcbExpiryTokenSize* are invalid. This value is defined in Ntdsbmsg.h.

RPC_S_INVALID_BINDING

The function is called remotely or the server in *szServerName* is not a domain controller.

Remarks

This function requires that the caller has the **SE_BACKUP_NAME** privilege. The [DsSetAuthIdentity](#) function can be used to change the security context under which this function is called.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

Requirement	Value
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsBackupPrepareW (Unicode) and DsBackupPrepareA (ANSI)

See also

[DsRestorePrepare](#)

[DsBackupFree](#)

[DsBackupOpenFile](#)

[DsBackupEnd](#)

[DsSetAuthIdentity](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsBackupRead function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupRead** function reads a block of data from the current open file, into a buffer. The client application is expected to call this function repeatedly until the entire backup file has been received. The [DsBackupOpenFile](#) function provides the entire size of the backup file.

Syntax

```
HRESULT DsBackupRead(
    _In_ HBC hbc,
    _In_ PVOID pvBuffer,
    _In_ DWORD cbBuffer,
    _Out_ PDWORD pcbRead
);
```

Parameters

hbc [in]

Contains the backup context handle obtained with the [DsBackupPrepare](#) function.

pvBuffer [in]

Pointer to a buffer that receives the data. This buffer must be at least *cbBuffer* bytes in size.

cbBuffer [in]

Contains the size, in bytes, of the buffer at *pvBuffer*. This value must be a multiple of 8192 and must be greater than or equal to 24576.

pcbRead [out]

Pointer to a **DWORD** value that receives the actual number of bytes read. This may be less than the number of bytes requested because some transports fragment the buffer being transmitted instead of filling the entire buffer with data.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. Possible error codes include the following.

ERROR_INVALID_PARAMETER

One or more parameters are not valid.

ERROR_HANDLE_EOF

The end of the backup file was reached.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll

See also

[DsBackupOpenFile](#)

[DsBackupPrepare](#)

[DsBackupFree](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsBackupTruncateLogs function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsBackupTruncateLogs** function truncates the previously read backup logs.

Syntax

```
HRESULT DsBackupTruncateLogs(
    _In_ HBC hbc
);
```

Parameters

hbc [in]

Contains the backup context handle obtained with the [DsBackupPrepare](#) function.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists other possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

ERROR_INVALID_PARAMETER

hbc is invalid.

Remarks

Use the **DsBackupTruncateLogs** function when a full or incremental backup has completed successfully.

Caution

If this function is called after a differential backup, all of the incremental backup information will be lost.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

Requirement	Value
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll

See also

[DsBackupPrepare](#)

[DsBackupGetBackupLogs](#)

[DsSetCurrentBackupLog](#)

[Backing Up an Active Directory Server](#)

[Directory Backup Functions](#)

DsIsNTDSOnline function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsIsNTDSOnline** function determines if Active Directory Domain Services are online on the specified server.

Syntax

```
HRESULT DsIsNTDSOnline(
    _In_  LPCTSTR szServerName,
    _Out_ BOOL     *pfNTDSOnline
);
```

Parameters

szServerName [in]

Pointer to a null-terminated string that contains the name of the server to test. Preceding backslashes are optional. The server must be the same computer that this function is called from. The server name cannot contain any underscore (_) characters. An example of a server name is "\\server1".

pfNTDSOnline [out]

Pointer to **BOOL** value that receives the result. Receives **TRUE** if the directory service is online or **FALSE** if the directory service is offline.

Return value

Returns **S_OK** if the function is successful or an error code otherwise. The following list lists possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

hrCouldNotConnect

The server in *szServerName* cannot be found, is not a domain controller, or *szServerName* is not formatted correctly. This value is defined in Ntdsbmsg.h.

RPC_S_INVALID_BINDING

The [DsIsNTDSOnline](#) function is being called remotely or the server in *szServerName* is not a domain controller.

Remarks

Call this function before calling any of the directory backup or restore functions. The directory must be online in

order to perform a backup. The directory must be offline to perform a restore.

This function can only be called from a domain controller that is also the target server specified in *szServerName*. This function cannot be called remotely.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsIsNTDSOnlineW (Unicode) and DsIsNTDSOnlineA (ANSI)

See also

[DsSetAuthIdentity](#)

[Directory Backup Functions](#)

[Backing Up and Restoring an Active Directory Server](#)

DsRestoreEnd function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsRestoreEnd** function is called to terminate a restore operation.

Syntax

```
HRESULT DsRestoreEnd(  
    _In_ HBC hbc  
)
```

Parameters

hbc [in]

Contains the restoration context handle obtained with the [DsRestorePrepare](#) function.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists possible error codes.

ERROR_INVALID_PARAMETER

hbc is not valid.

Remarks

The **DsRestoreEnd** function closes outstanding binding handles and performs a cleanup operation after a restore attempt.

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	None supported
End of client support	None supported
End of server support	None supported
Header	Ntdsbcli.h

REQUIREMENT	VALUE
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll

See also

[DsRestorePrepare](#)

[Restoring an Active Directory Server](#)

[Directory Backup Functions](#)

DsRestoreGetDatabaseLocations function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsRestoreGetDatabaseLocations** function obtains the locations where backup files should be copied during a restore operation.

Syntax

```
HRESULT DsRestoreGetDatabaseLocations(
    _In_ HBC      hbc,
    _Out_ LPWSTR  *pszDatabaseLocationList,
    _Out_ LPDWORD pcbSize
);
```

Parameters

hbc [in]

Contains the restoration context handle obtained with the [DsRestorePrepare](#) function.

pszDatabaseLocationList [out]

Pointer to a string pointer that receives the list of database locations as UNC paths. This list receives a double null-terminated list of single null-terminated strings.

This buffer is allocated by the **DsRestoreGetDatabaseLocations** function and must be freed when it is no longer required by calling the [DsBackupFree](#) function.

The first character of each of the file names contains one of the [BFT Constants](#) that identifies the name type. The **DsRestoreGetDatabaseLocations** function only supplies the following name types.

BFT_NTDS_DATABASE

The NTDS database file should be copied to this file. This is the file that was identified as **BFT_NTDS_DATABASE** when the backup was performed.

BFT_LOG_DIR

All log files are copied to this directory. The log files were identified as **BFT_LOG** when the backup was performed.

BFT_CHECKPOINT_DIR

All patch files are copied to this directory. The patch files were identified as **BFT_PATCH_FILE** when the backup was performed.

pcbSize [out]

Pointer to **DWORD** value that receives the size, in bytes, of the *pszDatabaseLocationList* buffer.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

ERROR_INVALID_PARAMETER

hbc, *pszDatabaseLocationList*, or *pcbSize* are invalid.

ERROR_NOT_ENOUGH_MEMORY

A memory allocation failure occurred.

Remarks

The [DsRestoreGetDatabaseLocations](#) function can be used to obtain the restoration directories without access to the backed up data. To do this, call [DsRestorePrepare](#) with **NULL** for the *pvExpiryToken* parameter. This causes [DsRestorePrepare](#) to return a restricted context handle which can only be used with the [DsRestoreGetDatabaseLocations](#) function.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsRestoreGetDatabaseLocationsW (Unicode) and DsRestoreGetDatabaseLocationsA (ANSI)

See also

[DsRestorePrepare](#)

[DsBackupFree](#)

[Directory Backup Functions](#)

[Restoring Active Directory](#)

DsRestorePrepare function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsRestorePrepare** function connects to the specified directory server and prepares it for the restore operation.

Syntax

```
HRESULT DsRestorePrepare(
    _In_    LPCWSTR szServerName,
    _In_    ULONG    rtFlag,
    _In_    PVOID    pvExpiryToken,
    _In_    DWORD    cbExpiryTokenSize,
    _Out_   HBC      *phbc
);
```

Parameters

szServerName [in]

Pointer to a null-terminated string that contains the name of the server to restore. Preceding backslashes are optional. The server must be the same computer that this function is called from. The server name cannot contain any underscore (_) characters. An example of a server name is "\\server1".

rtFlag [in]

Specifies the type of restoration to perform. This can be zero or one of the following values.

RESTORE_TYPE_CATCHUP

Default. The restored version is reconciled through the standard reconciliation logic so that the restored DIT can synchronize with other enterprise server computers.

RESTORE_TYPE_AUTHORATATIVE

Not Supported.

RESTORE_TYPE_ONLINE

Not Supported. Restoration is performed when NTDS is online.

pvExpiryToken [in]

Pointer to the expiry token associated with the backup being restored. This token was obtained from the [DsBackupPrepare](#) function when the directory was backed up.

If this parameter is **NULL**, the handle returned in *phbc* can only be used to obtain the restoration directories with the [DsRestoreGetDatabaseLocations](#) function. The handle cannot be used for any other restoration functions.

cbExpiryTokenSize [in]

Contains the size, in bytes, of the expiry token in *pvExpiryToken*.

phbc [out]

Pointer to an **HBC** value that receives the handle for the restore. This handle is used when calling other Directory Service restore functions, such as [DsBackupOpenFile](#) and [DsRestoreEnd](#).

Return value

If successful, returns a standard **HRESULT** codes; otherwise, a failure code is returned.

Remarks

The [DsRestorePrepare](#) function requires that the caller is a member of the Administrators group on the server.

[DsRestorePrepare](#) may be used with or without a token provided. If the token is provided, it is checked for expiration, and all operations are allowed on the context returned. If the token is not provided, the context returned is restricted, and may be used only for the [DsRestoreGetDatabaseLocations](#) function. It may not be used for the [DsRestoreRegister](#) function.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsRestorePrepareW (Unicode) and DsRestorePrepareA (ANSI)

See also

[Restoring an Active Directory Server](#)

[Directory Backup Functions](#)

[DsRestoreGetDatabaseLocations](#)

[DsRestoreRegister](#)

[DsRestoreRegisterComplete](#)

[DsRestoreEnd](#)

DsRestoreRegister function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsRestoreRegister** function registers a restore operation. This function interlocks all subsequent restore operations and prevents the restore target from starting until the **DsRestoreRegisterComplete** function is called.

Syntax

```
HRESULT DsRestoreRegister(
    _In_ HBC          hbc,
    _In_ LPCTSTR      szCheckPointFilePath,
    _In_ LPCTSTR      szLogPath,
    _In_ EDB_RSTMAP* rgrstmap[],
    _In_ LONG         crstmap,
    _In_ LPCTSTR      szBackupLogPath,
    _In_ ULONG        genLow,
    _In_ ULONG        genHigh
);
```

Parameters

hbc [in]

Contains the restoration context handle obtained with the **DsRestorePrepare** function.

szCheckPointFilePath [in]

Pointer to a null-terminated string that contains the path to the checkpoint file. This path is provided by the **DsRestoreGetDatabaseLocations** function and has a BFT value of **BFT_CHECKPOINT_DIR**. Typically this is the same as the system database path. This path is required for proper backup restore function. This parameter cannot be **NULL**. Passing **NULL** in this parameter will cause an error during the restore process.

szLogPath [in]

Pointer to a null-terminated string that contains the path where the log files will be restored. This path is provided by the **DsRestoreGetDatabaseLocations** function and has a BFT value of **BFT_LOG_DIR**. If the path points to an empty directory, new log files are created there. This parameter cannot be **NULL**.

rgrstmap [in]

An array of **EDB_RSTMAP** structures that contains the old and new paths for each database. There is one structure for each database. For the directory, there is a structure for the system database and another structure for the directory database. The order of the elements in the array does not matter. The *crstmap* parameter contains the number of elements in the array.

crstmap [in]

Contains the number of elements in the *rgrstmap* array.

szBackupLogPath [in]

Pointer to a null-terminated string that contains the path where the backed up log files currently reside. This parameter cannot be **NULL**.

genLow [in]

Contains the lowest log number to restore in this restore session. This is a hexadecimal number in the range from 0x00000 to 0xFFFF.

genHigh [in]

Contains the highest log number to restore in this restore session. This is a hexadecimal number in the range from 0x00000 to 0xFFFF.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

ERROR_INVALID_PARAMETER

One or more parameters are invalid.

hrMissingExpiryToken

The expiry token supplied to [DsRestorePrepare](#) was invalid. This value is defined in Ntdsbmsg.h.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsRestoreRegisterW (Unicode) and DsRestoreRegisterA (ANSI)

See also

[DsRestoreRegisterComplete](#)

[**DsRestorePrepare**](#)

[**DsRestoreGetDatabaseLocations**](#)

[**DsRestoreEnd**](#)

[**EDB_RSTMAP**](#)

[**Restoring Active Directory**](#)

[**Directory Backup Functions**](#)

DsRestoreRegisterComplete function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsRestoreRegisterComplete** function is called to unlock an Active Directory server after a restore operation is complete. This function is counterpart to the [DsRestoreRegister](#) function.

Syntax

```
HRESULT DsRestoreRegisterComplete(
    _In_ HBC      hbc,
    _In_ HRESULT hrRestoreState
);
```

Parameters

hbc [in]

Contains the restoration context handle obtained with the [DsRestorePrepare](#) function.

hrRestoreState [in]

Contains the final status of the restore operation. This parameter should contain S_OK if the restore operation was successful or an error code otherwise.

Return value

Returns S_OK if the function is successful or a Win32 or RPC error code otherwise. The following list lists possible error codes.

ERROR_ACCESS_DENIED

The caller does not have the proper access privileges to call this function. The [DsSetAuthIdentity](#) function can be used to set the credentials to use for the backup and restore functions.

Remarks

Before you restart the domain controller, call this function to provide the status of the restore operation. If the status is not successful, the directory service will not start until a valid database has been restored. This function completes the restore operation and allows Active Directory Domain Services to start.

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	None supported

Requirement	Value
End of client support	None supported
End of server support	None supported
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll

See also

[DsRestoreRegister](#)

[DsRestorePrepare](#)

[DsSetAuthIdentity](#)

[Restoring an Active Directory Server](#)

[Directory Backup Functions](#)

DsSetAuthIdentity function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsSetAuthIdentity** function sets the security context under which the directory backup APIs are called.

Syntax

```
HRESULT DsSetAuthIdentity(
    _In_ LPCTSTR szUserName,
    _In_ LPCTSTR szDomainName,
    _In_ LPCTSTR szPassword
);
```

Parameters

szUserName [in]

The null-terminated string that specifies the user name.

szDomainName [in]

The null-terminated string that specifies the name of the domain that the user belongs to.

szPassword [in]

The null-terminated string that specifies the password of the user in the specified domain.

Return value

If successful, returns a standard **HRESULT** success codes; otherwise, a failure code is returned.

Remarks

If **DsSetAuthIdentity** is not called, security context of the current process is assumed.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h

Requirement	Value
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsSetAuthIdentityW (Unicode) and DsSetAuthIdentityA (ANSI)

See also

[Backing Up and Restoring an Active Directory Server](#)

[Directory Backup Functions](#)

DsSetCurrentBackupLog function

6/3/2022 • 2 minutes to read • [Edit Online](#)

[This function is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Beginning with Windows Vista, use [Volume Shadow Copy Service \(VSS\)](#) instead.]

The **DsSetCurrentBackupLog** function sets the current backup log number after a successful restore. Because Active Directory Domain Services only support circular logging, this function is not normally used.

Syntax

```
HRESULT DsSetCurrentBackupLog(
    _In_  LPCWSTR szServerName,
    _In_  DWORD   dwCurrentLog
);
```

Parameters

szServerName [in]

Pointer to a null-terminated string that contains the name of the server to set the backup log number for. Preceding backslashes are optional. The server must be the same computer that this function is called from. The server name cannot contain any underscore (_) characters. An example of a server name is "\\server1".

dwCurrentLog [in]

Contains the backup log number to set.

Return value

Returns **S_OK** if the function is successful or a Win32 or RPC error code otherwise. The following list lists possible error codes.

ERROR_INVALID_PARAMETER

One or more parameters are invalid.

ERROR_NOT_ENOUGH_MEMORY

A memory allocation failure occurred.

Remarks

It is not normally required to call the **DsSetCurrentBackupLog** function. The backup functions automatically determine and set the last log number backed up. Use **DsSetCurrentBackupLog** to prevent another incremental backup from succeeding until a full backup is performed.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Ntdsbcli.h
Library	Ntdsbcli.lib
DLL	Ntdsbcli.dll
Unicode and ANSI names	DsSetCurrentBackupLogW (Unicode) and DsSetCurrentBackupLogA (ANSI)

See also

[Backing Up and Restoring an Active Directory Server](#)

[Directory Backup Functions](#)

Interfaces for Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following are categories of interfaces for Active Directory Domain Services:

- [Admin Interfaces in Active Directory Domain Services](#)
- [Display Interfaces in Active Directory Domain Services](#)
- [Object Picker Dialog Box Interfaces](#)

Admin Interfaces in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

This category of interfaces includes the following.

- [IDsAdminCreateObj](#)
- [IDsAdminNewObj](#)
- [IDsAdminNewObjPrimarySite](#)
- [IDsAdminNewObjExt](#)
- [IDsAdminNotifyHandler](#)

See Also

[Display Interfaces in Active Directory Domain Services](#), [Object Picker Dialog Box Interfaces](#)

Display Interfaces in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

This category of interfaces includes the following:

- [ICommonQuery](#)
- [IDsBrowseDomainTree](#)
- [IDsDisplaySpecifier](#)
- [IPersistQuery](#)
- [IQueryForm](#)

See Also

[Active Directory Admin Interfaces](#), [Object Picker Dialog Box Interfaces](#)

Object Picker Dialog Box Interfaces

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following interface is used with object picker dialog boxes:

- [IDsObjectPicker](#)
- [IDsObjectPickerCredentials](#)

Related topics

[Directory Object Picker](#)

Messages in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Messages in Active Directory Domain Services are functional in Windows 2000 and Windows NT 4.0 with Service Pack 6a (SP6a) and later with DSClient. They are also functional in Windows 98/95 workstations with IE4.01 and later and DSClient. However, if multiselect property pages are launched from the shell, then the [WM_ADSPROP_NOTIFY_ERROR](#) message and its corresponding helper functions, [ADsPropSendMessage](#) and [ADsPropShowErrorDialog](#) are not functional and will not be displayed. If multiselect property pages are launched within an admin tool, for example, AD Users and Computers, then all messages are functional and available to be displayed within the multiselect property pages.

Active Directory Domain Services use the following messages:

- [WM_ADSPROP_NOTIFY_APPLY](#)
- [WM_ADSPROP_NOTIFY_CHANGE](#)
- [WM_ADSPROP_NOTIFY_ERROR](#)
- [WM_ADSPROP_NOTIFY_EXIT](#)
- [WM_ADSPROP_NOTIFY_FOREGROUND](#)
- [WM_ADSPROP_NOTIFY_PAGEHWND](#)
- [WM_ADSPROP_NOTIFY_PAGEINIT](#)
- [WM_ADSPROP_NOTIFY_SETFOCUS](#)
- [WM_ADSPROP_SHEET_CREATE](#)
- [WM_DSA_SHEET_CLOSE_NOTIFY](#)
- [WM_DSA_SHEET_CREATE_NOTIFY](#)

WM_ADSPROP_NOTIFY_APPLY message

6/3/2022 • 2 minutes to read • [Edit Online](#)

An Active Directory directory service property sheet extension sends the WM_ADSPROP_NOTIFY_APPLY message to the notification object if the property page PSN_APPLY handler succeeds.

WM_ADSPROP_NOTIFY_APPLY

WPARAM wParam
LPARAM lParam

Parameters

hwnd

The handle of the notification object. To obtain this handle, call [ADsPropCreateNotifyObj](#).

wParam

Not used.

lParam

Not used.

Return value

This message has no return value.

Remarks

When adding pages to the Active Directory Manager MMC snap-in, Active Directory MMC property sheets create the notification objects by a call to the [ADsPropCreateNotifyObj](#) function, and then passes the notification object handle to each property page.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

[ADsPropCreateNotifyObj](#)

Messages in Active Directory Domain Services

WM_ADSPROP_NOTIFY_CHANGE message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The WM_ADSPROP_NOTIFY_CHANGE message is used internally by the notification object.

`WM_ADSPROP_NOTIFY_CHANGE`

`WPARAM wParam`
`LPARAM lParam`

Parameters

hwnd

Not used.

wParam

Not used.

lParam

Not used.

Return value

This message has no return value.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

[Messages in Active Directory Domain Services](#)

WM_ADSPROP_NOTIFY_ERROR message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The WM_ADSPROP_NOTIFY_ERROR message adds an error message to a list of error messages that are displayed by calling the [ADsPropShowErrorDialog](#) function.

```
WM_ADSPROP_NOTIFY_ERROR  
  
WPARAM wParam  
LPARAM lParam
```

Parameters

hwnd

Handle of the notification object. This is the *hNotifyObject* parameter obtained by [ADsPropCreateNotifyObj](#).

wParam

Not used.

lParam

Pointer to an [ADSPROPERROR](#) structure that contains error message data.

Return value

This message has no return value.

Remarks

The [ADsPropSendErrorMessage](#) function is the preferred method of sending this message.

The error messages added by the WM_ADSPROP_NOTIFY_ERROR message are accumulated until [ADsPropShowErrorDialog](#) is called. [ADsPropShowErrorDialog](#) combines and displays the accumulated error messages. When the error dialog is dismissed, the accumulated error messages are deleted.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

Messages in Active Directory Domain Services

ADSPROPERROR

ADsPropSendErrorMessage

ADsPropShowErrorDialog

WM_ADSPROP_NOTIFY_EXIT message

6/3/2022 • 2 minutes to read • [Edit Online](#)

An Active Directory property sheet extension sends the WM_ADSPROP_NOTIFY_EXIT message to the notification object when the notification object is no longer required.

WM_ADSPROP_NOTIFY_EXIT

WPARAM wParam
LPARAM lParam

Parameters

hwnd

The handle of the notification object. To obtain this handle, call [ADsPropCreateNotifyObj](#).

wParam

Not used.

lParam

Not used.

Return value

This message has no return value.

Remarks

The notification object will delete itself in response to this message. When this message has been sent, the notification object handle should be considered invalid.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

[ADsPropCreateNotifyObj](#)

[Messages in Active Directory Domain Services](#)

WM_ADSPROP_NOTIFY_FOREGROUND message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The WM_ADSPROP_NOTIFY_FOREGROUND message is used internally by the notification object.

`WM_ADSPROP_NOTIFY_FOREGROUND`

`WPARAM wParam`
`LPARAM lParam`

Parameters

hwnd

Not used.

wParam

Not used.

lParam

Not used.

Return value

This message has no return value.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

[Messages in Active Directory Domain Services](#)

WM_ADSPROP_NOTIFY_PAGEHWND message

6/3/2022 • 2 minutes to read • [Edit Online](#)

An Active Directory directory service property sheet extension calls the [ADsPropSetHwnd](#) to inform the notification object of the property page window handle. The [ADsPropSetHwnd](#) function sends the **WM_ADSPROP_NOTIFY_PAGEHWND** message to the notification object, which informs the notification object of the property page window handle.

WM_ADSPROP_NOTIFY_PAGEHWND

WPARAM *hPage*
LPARAM *ptzTitle*

Parameters

hNotifyObj

The handle of the notification object. To obtain this handle, call [ADsPropCreateNotifyObj](#).

hPage

A window handle of the property page.

ptzTitle

Pointer to a NULL-terminated **WCHAR** buffer that contains the title of the property page.

Return value

This message has no return value.

Remarks

An Active Directory property sheet extension normally calls the [ADsPropSetHwnd](#) function while processing the **WM_INITDIALOG** message.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

[Messages in Active Directory Domain Services](#)

[ADsPropSetHwnd](#)

[ADsPropCreateNotifyObj](#)

[WM_INITDIALOG](#)

WM_ADSPROP_NOTIFY_PAGEINIT message

6/3/2022 • 2 minutes to read • [Edit Online](#)

An Active Directory property sheet extension calls the [ADsPropGetInitInfo](#) to obtain data about regarding the directory object that the property sheet extension applies to. The [ADsPropGetInitInfo](#) function sends the **WM_ADSPROP_NOTIFY_PAGEINIT** message to the notification object to achieve this result.

```
WM_ADSPROP_NOTIFY_PAGEINIT  
  
WPARAM wParam  
LPARAM pADsPropInitParams
```

Parameters

hwnd

Handle of the notification object. This is the *hNotifyObject* parameter obtained by a call to [ADsPropCreateNotifyObj](#).

wParam

Not used.

pADsPropInitParams

Pointer to an [ADSPROPINITPARAMS](#) structure that receives the directory object information. This is the *pInitParams* parameter passed to [ADsPropCreateNotifyObj](#).

Return value

This message has no return value.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

[Messages in Active Directory Domain Services](#)

[ADsPropGetInitInfo](#)

[ADSPROPINITPARAMS](#)

WM_ADSPROP_NOTIFY_SETFOCUS message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The WM_ADSPROP_NOTIFY_SETFOCUS message is used internally by the notification object.

`WM_ADSPROP_NOTIFY_SETFOCUS`

`WPARAM wParam`
`LPARAM lParam`

Parameters

hwnd

Not used.

wParam

Not used.

lParam

Not used.

Return value

This message has no return value.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008
Header	Adsprop.h

See also

[Messages in Active Directory Domain Services](#)

WM_ADSPROP_SHEET_CREATE message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The WM_ADSPROP_SHEET_CREATE message is sent to the notification object to create a secondary property sheet in an Active Directory MMC snap-in.

NOTE

This message value is not defined in a published header file. To use this message value, you must define it yourself in the exact format shown.

```
#define WM_ADSPROP_SHEET_CREATE (WM_USER + 1108)
LRESULT SendMessage( (HWND) hwnd,
                     (UINT) WM_ADSPROP_SHEET_CREATE,
                     (WPARAM) wParam,
                     (LPARAM) lParam);
```

Parameters

hwnd

The handle of the notification object. To obtain this handle, call [ADsPropCreateNotifyObj](#).

wParam

Contains a pointer to a [DSA_SEC_PAGE_INFO](#) structure that defines the secondary page to create. Only one secondary property sheet can be created with the WM_ADSPROP_SHEET_CREATE message, so the [DSOBJECTNAMES](#) structure can only contain one [DSOBJECT](#) structure.

lParam

Not used. Must be NULL.

Return value

The return value for this message is always zero.

Remarks

The caller must allocate the [DSA_SEC_PAGE_INFO](#) structure, the title string and all [DSOBJECT](#) strings using a single call to the [LocalAlloc](#) function. The WM_ADSPROP_SHEET_CREATE message is an asynchronous message, so it will return before the secondary sheet is created. Because of this, the memory must remain intact after the message returns. The receiver will free this memory with a single call to the [LocalFree](#) function after the secondary sheet is created.

Requirements

Requirement	Value
Minimum supported client	Windows Vista

Requirement	Value
Minimum supported server	Windows Server 2008

See also

[CFSTR_DS_PARENTHWN](#)

[DSA_SEC_PAGE_INFO](#)

[DSOBJECTNAMES](#)

[DSOBJECT](#)

[LocalAlloc](#)

[LocalFree](#)

WM_DSA_SHEET_CLOSE_NOTIFY message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The WM_DSA_SHEET_CLOSE_NOTIFY message is posted to the Active Directory MMC snap-in when a secondary property sheet is destroyed.

NOTE

This message value is not defined in a published header file. To use this message value, you must define it yourself in the exact format shown.

```
#define WM_DSA_SHEET_CLOSE_NOTIFY (WM_USER + 5)
```

```
LRESULT SendMessage( (HWND)    hwnd,
                     (UINT)     WM_DSA_SHEET_CLOSE_NOTIFY,
                     (WPARAM)   wParam,
                     (LPARAM)   lParam);
```

Parameters

hwnd

The window handle of the snap-in window that will process this message. This handle is obtained from the **hwndHidden** member of the [PROPSHEETCFG](#) structure.

wParam

Contains an application-defined 32-bit value. This is obtained from the **wParamSheetClose** member of the [PROPSHEETCFG](#) structure.

lParam

Not used.

Return value

The return value for this message is not used.

Requirements

REQUIREMENT	VALUE
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

See also

[PROPSHEETCFG](#)

WM_DSA_SHEET_CREATE_NOTIFY message

6/3/2022 • 2 minutes to read • [Edit Online](#)

The WM_DSA_SHEET_CREATE_NOTIFY message is posted to the Active Directory MMC snap-in to create a secondary property sheet.

NOTE

This message value is not defined in a published header file. To use this message value, define it in the exact format shown.

```
#define WM_DSA_SHEET_CREATE_NOTIFY (WM_USER + 6)
LRESULT SendMessage(
    (HWND) hwnd,
    WM_DSA_SHEET_CREATE_NOTIFY,
    (WPARAM) wParam,
    (LPARAM) lParam);
```

Parameters

hwnd

The window handle of the snap-in window that will process this message. This handle is obtained from the **hwndHidden** member of the [PROPSHEETCFG](#) structure.

wParam

Contains a pointer to a [DSA_SEC_PAGE_INFO](#) structure that defines the secondary property sheet to create. The message receiver must free this memory with the [LocalFree](#) function when it is no longer required.

lParam

Not used.

Return value

Not used.

Requirements

Requirement	Value
Minimum supported client	Windows Vista
Minimum supported server	Windows Server 2008

See also

[PROPSHEETCFG](#)

[DSA_SEC_PAGE_INFO](#)

Function Return Values in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following are categories of return values for functions in Active Directory Domain Services:

- [Success](#)
- [Backup Errors in Active Directory Domain Services](#)
- [System Errors in Active Directory Domain Services](#)
- [Directory Manager Errors](#)
- [Logging and Recovery Errors in Functions in Active Directory Domain Services](#)

Success

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic describes success return values from functions in Active Directory Domain Services.

VALUE	DESCRIPTION
hrNone	The operation was successful.

Related topics

[Backup Errors in Active Directory Domain Services](#)

[Logging and Recovery Errors in Active Directory Domain Services](#)

[Return Values in Active Directory Domain Services](#)

[System Errors in Active Directory Domain Services](#)

[Directory Manager Errors](#)

Backup Errors in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic lists backup error return values for functions in Active Directory Domain Services.

VALUE	DESCRIPTION
hrNyi	The function is not implemented.
hrInvalidParam	The parameter is not valid.
hrError	An internal error occurred.
hrInvalidHandle	The handle is not valid.
hrRestoreInProgress	The restore process is in progress.
hrAlreadyOpen	The specified file is open.
hrInvalidRecips	The recipients are not valid.
hrCouldNotConnect	Unable to backup. A connection to the specified backup server was not detected or the service you are attempting to back up is not running.
hrRestoreMapExists	A restore map exists for the specified component. A restore map can be specified when performing a full restore.
hrIncrementalBackupDisabled	Another application has modified the specified Windows NT/Windows 2000 Directory Service database such that any subsequent backups will fail. To fix this, perform a full backup.
hrLogFileNotFound	Unable to perform an incremental backup because a required Windows NT/Windows 2000 Directory Service database log file could not be found.
hrCircularLogging	The Windows NT/Windows 2000 Directory Service component specified is configured to use circular database logs. It cannot be backed up incrementally. Perform a full backup.
hrNoFullRestore	The databases have not been restored to this computer. You cannot restore an incremental backup until a full backup has been restored.
hrCommunicationError	A communications error occurred while attempting to perform a local backup.
hrFullBackupNotTaken	You must perform a full backup before you can perform an incremental backup.

VALUE	DESCRIPTION
hrMissingExpiryToken	Expiry token is missing. Cannot restore without the expiry data.
hrUnknownExpiryTokenFormat	Expiry token is in an unrecognizable format.
hrContentsExpired	DS Contents in the backup are out of date. Restore with a recent copy.

See Also

[Success](#), [System Errors in Active Directory Domain Services](#), [Directory Manager Errors](#), [Logging and Recovery Errors](#) in [Functions in Active Directory Domain Services](#), [Return Values for Functions in Active Directory Domain Services](#)

System Errors in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic contains system error return values in functions of Active Directory Domain Services.

VALUE	DESCRIPTION
<code>hrFileClose</code>	Unable to close the DOS file.
<code>hrOutOfThreads</code>	Unable to start a thread because there are none available.
<code>hrTooManyIO</code>	The system is busy because there are too many I/Os.
<code>hrBFNotSynchronous</code>	The buffer page has been evicted.
<code>hrBFPageNotFound</code>	Unable to find the page.
<code>hrBFInUse</code>	Unable to abandon the buffer.

Related topics

[Success](#)

[Backup Errors in Active Directory Domain Services](#)

[Directory Manager Errors](#)

[Logging and Recovery Errors in Functions in Active Directory Domain Services](#)

[Return Values for Functions in Active Directory Domain Services](#)

Directory Manager Errors

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic contains the return values for Directory Manager errors from functions in Active Directory Domain Services.

VALUE	DESCRIPTION
<code>hrPMRecDeleted</code>	The record was deleted.
<code>hrRemainingVersions</code>	There is idle work remaining.
<code>hrFLDKeyTooBig</code>	The key was truncated because it is more than 255 bytes.
<code>hrFLDTooManySegments</code>	There are too many key segments.
<code>hrFLDNullKey</code>	The key is NULL .

Related topics

[Backup Errors in Active Directory Domain Services](#)

[Logging and Recovery Errors Active Directory Domain Services](#)

[Return Values for functions in Active Directory Domain Services](#)

[System Errors in Active Directory Domain Services](#)

[Success](#)

Logging and Recovery Errors in Functions in Active Directory Domain Services

6/3/2022 • 6 minutes to read • [Edit Online](#)

This topics lists logging and recovery error return values for functions in Active Directory Domain Services.

VALUE	DESCRIPTION
hrLogFileCorrupt	The log file is damaged.
hrNoBackupDirectory	No backup directory was given.
hrBackupDirectoryNotEmpty	The backup directory is not empty.
hrBackupInProgress	Backup is active.
hrMissingPreviousLogFile	A log file for the checkpoint is missing.
hrLogWriteFail	Unable to write to the log file.
hrBadLogVersion	The version of the log file is not compatible with the version of the Windows NT/Windows 2000 Directory Service database (NTDS).
hrInvalidLogSequence	The time stamp in the next log does not match what was expected.
hrLoggingDisabled	The log is not active.
hrLogBufferTooSmall	The log buffer is too small to be recovered.
hrLogSequenceEnd	The maximum number of log files has been exceeded.
hrNoBackup	There is no backup in progress.
hrInvalidBackupSequence	The backup call is out of sequence.
hrBackupNotAllowedYet	Unable to perform a backup now.
hrDeleteBackupFileFail	Unable to delete the backup file.
hrMakeBackupDirectoryFail	Unable to make a backup temporary directory.
hrInvalidBackup	An incremental backup cannot be performed when circular logging is enabled.
hrRecoveredWithErrors	Errors were encountered during the repair process.
hrMissingLogFile	The current log file is missing.

VALUE	DESCRIPTION
hrLogDiskFull	The log disk is full.
hrBadLogSignature	A log file is damaged.
hrBadDbSignature	A database file is damaged.
hrBadCheckpointSignature	A checkpoint file is damaged.
hrCheckpointCorrupt	A checkpoint file either could not be found or is damaged.
hrDatabaseInconsistent	The database is damaged.
hrConsistentTimeMismatch	There is a mismatch in the database's last consistent time.
hrPatchFileMismatch	The patch file is not generated from this backup.
hrRestoreLogTooLow	The starting log number is too low for the restore.
hrRestoreLogTooHigh	The starting log number is too high for the restore.
hrGivenLogFileHasBadSignature	The log file downloaded from the tape is damaged.
hrGivenLogFileIsNotContiguous	Unable to find a mandatory log file after the tape was downloaded.
hrMissingRestoreLogFile	The data is not fully restored because some log files are missing.
hrExistingLogFileHasBadSignature	The log file in the log file path is damaged.
hrExistingLogFileIsNotContiguous	Unable to find a mandatory log file in the log file path.
hrMissingFullBackup	The database missed a previous full backup before the incremental backup.
hrBadBackupDatabaseSize	The backup database size must be a multiple of 4000 (4096 bytes).
hrTermInProgress	The database is being shut down.
hrFeatureNotAvailable	The feature is not available.
hrInvalidName	The name is invalid.
hrInvalidParameter	The parameter is invalid.
hrColumnNull	The value of the column is null.
hrBufferTruncated	The buffer is too small for data.
hrDatabaseAttached	The database is already attached.

VALUE	DESCRIPTION
hrInvalidDatabaseId	The database ID is invalid.
hrOutOfMemory	The computer is out of memory.
hrOutOfDatabaseSpace	The database has reached the maximum size of 16 GB.
hrOutOfCursors	Out of table cursors.
hrOutOfBuffers	Out of database page buffers.
hrTooManyIndexes	There are too many indexes.
hrTooManyKeys	There are too many columns in an index.
hrRecordDeleted	The record has been deleted.
hrReadVerifyFailure	A read verification error occurred.
hrOutOfFileHandles	Out of file handles.
hrDiskIO	A disk I/O error occurred.
hrInvalidPath	The path to the file is invalid.
hrRecordTooBig	The record has exceeded the maximum size.
hrTooManyOpenDatabases	There are too many open databases.
hrInvalidDatabase	The file is not a database file.
hrNotInitialized	The database was not yet called.
hrAlreadyInitialized	The database was already called.
hrFileAccessDenied	Unable to access the file.
hrBufferTooSmall	The buffer is too small.
hrSeekNotEqual	Either SeekLE or SeekGE cannot find an exact match.
hrTooManyColumns	There are too many columns defined.
hrContainerNotEmpty	The container is not empty.
hrInvalidFilename	The filename is invalid.
hrInvalidBookmark	The bookmark is invalid.
hrColumnInUse	The column is used in an index.

VALUE	DESCRIPTION
hrInvalidBufferSize	The data buffer does not match the column size.
hrColumnNotUpdatable	Unable to set the column value.
hrIndexInUse	The index is in use.
hrNullKeyDisallowed	Null keys are not allowed on an index.
hrNotInTransaction	The operation must be within a transaction.
hrNoIdleActivity	No idle activity occurred.
hrTooManyActiveUsers	There are too many active database users.
hrInvalidCountry	The country or region code is either not known or is invalid.
hrInvalidLanguageId	The language ID is either not known or is invalid.
hrInvalidCodePage	The code page is either not known or is invalid.
hrNoWriteLock	There is no write lock at transaction level 0.
hrColumnSetNull	The column value is set to null.
hrVersionStoreOutOfMemory	The lMaxVerPages exceeded (XJET only).
hrCurrencyStackOutOfMemory	Out of cursors.
hrOutOfSessions	Out of sessions.
hrWriteConflict	The write lock failed due to an outstanding write lock.
hrTransTooDeep	The transactions are nested too deeply.
hrInvalidSesid	The session handle is invalid.
hrSessionWriteConflict	Another session has a private version of the page.
hrInTransaction	The operation is not allowed within a transaction.
hrDatabaseDuplicate	The database already exists.
hrDatabaseInUse	The database is in use.
hrDatabaseNotFound	The database does not exist.
hrDatabaseInvalidName	The database name is invalid.
hrDatabaseInvalidPages	The number of pages is invalid.

VALUE	DESCRIPTION
hrDatabaseCorrupted	The database file is either damaged or cannot be found.
hrDatabaseLocked	The database is locked.
hrTableEmpty	An empty table was opened.
hrTableLocked	The table is locked.
hrTableDuplicate	The table already exists.
hrTableInUse	Unable to lock the table because it is already in use.
hrObjectNotFound	The table or object does not exist.
hrCannotRename	Unable to rename the temporary file.
hrDensityInvalid	The file/index density is invalid.
hrTableNotEmpty	Unable to define the clustered index.
hrInvalidTableId	The table ID is invalid.
hrTooManyOpenTables	Unable to open any more tables.
hrIllegalOperation	The operation is not supported on tables.
hrObjectDuplicate	The table or object name is already being used.
hrInvalidObject	The object is invalid for operation.
hrIndexCantBuild	Unable to build a clustered index.
hrIndexHasPrimary	The primary index is already defined.
hrIndexDuplicate	The index is already defined.
hrIndexNotFound	The index does not exist.
hrIndexMustStay	Unable to delete a clustered index.
hrIndexInvalidDef	The index definition is illegal.
hrIndexHasClustered	The clustered index is already defined.
hrCreateIndexFailed	Unable to create the index because an error occurred while creating a table.
hrTooManyOpenIndexes	Out of index description blocks.
hrColumnLong	The column value is too long.

VALUE	DESCRIPTION
hrColumnDoesNotFit	The field will not fit in the record.
hrNullInvalid	The value cannot be null.
hrColumnIndexed	Unable to delete because the column is indexed.
hrColumnTooBig	The length of the field exceeds the maximum length of 255 bytes.
hrColumnNotFound	Unable to find the column.
hrColumnDuplicate	The field is already defined.
hrColumn2ndSysMaint	Only one auto-increment or version column is allowed per table.
hrInvalidColumnType	The column data type is invalid.
hrColumnMaxTruncated	The column was truncated because it exceeded the maximum length of 255 bytes.
hrColumnCannotIndex	Unable to index a long value column.
hrTaggedNotNull	Tagged columns cannot be null.
hrNoCurrentIndex	The entry is invalid without a current index.
hrKeyIsMade	The key is complete.
hrBadColumnId	The column ID is incorrect.
hrBadItagSequence	There is a bad instance identifier for a multivalued column.
hrCannotBeTagged	AutoIncrement and Version cannot be multivalued.
hrRecordNotFound	Unable to find the key.
hrNoCurrentRecord	The currency is not on a record.
hrRecordClusteredChanged	A clustered key cannot be changed.
hrKeyDuplicate	The key already exists.
hrAlreadyPrepared	The current entry has already been copied or cleared.
hrKeyNotMade	No key was made.
hrUpdateNotPrepared	Update was not prepared.
hrwrnDataHasChanged	Data has changed.

VALUE	DESCRIPTION
hrerrDataHasChanged	The operation was abandoned because data has changed.
hrKeyChanged	Moved to a new key.
hrTooManySorts	There are too many sort processes.
hrInvalidOnSort	An operation that is invalid occurred in the sort.
hrTempFileOpenError	Unable to open the temporary file.
hrTooManyAttachedDatabases	There are too many databases open.
hrDiskFull	The disk is full.
hrPermissionDenied	Permission is denied.
hrFileNotFound	Unable to find the file.
hrFileOpenReadOnly	The database file is read-only.
hrAfterInitialization	Unable to restore after initialization.
hrLogCorrupted	The database log files are damaged.
hrInvalidOperation	The operation is invalid.
hrAccessDenied	Access is denied.

Related topics

[Success](#)

[Backup Errors in Active Directory Domain Services](#)

[System Errors in Active Directory Domain Services](#)

[Directory Manager Errors](#)

[Return Values for Functions in Active Directory Domain Services](#)

User Interface Mappings in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The user interface for Active Directory Domain Services includes property sheets and other user interfaces for setting and retrieving the attributes of commonly used classes. The topics in this section contain tables that map labels that appear in the user interfaces with the corresponding attributes in the Active Directory Schema.

- [Mappings for the Active Directory Users and Computers Snap-in](#)

Mappings for the Active Directory Users and Computers Snap-in

6/3/2022 • 2 minutes to read • [Edit Online](#)

The Active Directory Users and Computers snap-in includes property sheets and other user interfaces for the following object classes.

The Object property sheet is common to all object classes.

- [Computer Object User Interface Mapping](#)
- [Domain Object User Interface Mapping](#)
- [Group Object User Interface Mapping](#)
- [Object Property Sheet](#)
- [Organizational Unit User Interface Mapping](#)
- [Printer Object User Interface Mapping](#)
- [Shared Folder Object User Interface Mapping](#)
- [User Object User Interface Mapping](#)

Computer Object User Interface Mapping

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following tables list elements of the Computer object property sheets in the Active Directory Users and Computers snap-in.

General Property Sheet

The following table lists the UI labels of the **General** property sheet.

UI LABEL	ACTIVE DIRECTORY DOMAIN SERVICES ATTRIBUTE	COMMENTS
Computer Name (pre-Windows 2000)	sAMAccountName	
DNS Name	dNSHostName	
Role	userAccountControl	Toggles a bit in the userAccountControl bitmask.
Description	description	
Trust Computer for delegation	userAccountControl	Toggles a bit in the userAccountControl bitmask.

Location Property Sheet

The following table lists the UI labels of the **Location** property sheet.

UI LABEL	ACTIVE DIRECTORY DOMAIN SERVICES ATTRIBUTE
Location	location

Member Of Property Sheet

The following table lists the UI labels of the **Member Of** property sheet.

UI LABEL	ACTIVE DIRECTORY DOMAIN SERVICES ATTRIBUTE	COMMENTS
Member of	memberOf	Includes all of the groups in the UI list, except the primary group, which is represented in the AD through the primaryGroupId attribute.
Set Primary Group	primaryGroupID	LDAP: Tied to primaryGroupToken of the primary group.

Operating System Property Sheet

The following table lists the UI labels of the **Operating System** property sheet.

UI LABEL	ACTIVE DIRECTORY DOMAIN SERVICES ATTRIBUTE
Name	operatingSystem
Version	operatingSystemVersion
Service Pack	operatingSystemServicePack

Domain Object User Interface Mapping

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic discusses the Domain object property sheets in the Active Directory Users and Computers snap-in.

General Property Sheet

The following table shows the UI labels on the **General** property sheet.

UI LABEL	ACTIVE DIRECTORY ATTRIBUTE
Domain Name (pre-Windows 2000)	DC*
Description	description

*The DC attribute provides access to the first of the domain components which is not necessarily the same as the NetBIOS DN.

Group Object User Interface Mapping

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic describes the Group object property sheets in the Active Directory Users and Computers snap-in.

- [General Property Sheet](#)
- [Member Of Property Sheet](#)
- [Members Property Sheet](#)
- [Managed By Property Sheet](#)

General Property Sheet

The following table shows the UI labels of the **General** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Group Name (pre-Windows 2000)	SAM-Account-Name
Description	Description
E-Mail	E-mail-Addresses
Group Scope	Group-Type
Group Type	Group-Type
Notes	Comment

Member Of Property Sheet

The following table shows the UI labels of the **Member Of** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	DESCRIPTION
Member of	Is-Member-Of-DL	Contains the distinguished names of the groups to which this group belongs. The member attribute of each of the groups in this list contains the distinguished name of this group object. The user interface does not directly modify the Is-Member-Of-DL attribute. It modifies the Member attribute on the group object of which this object is made a member of. The Active Directory server maintains the Is-Member-Of-DL attribute.

Members Property Sheet

The following table shows the UI labels of the **Members** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	DESCRIPTION
Members	Member	Contains the distinguished names of the members of this group object.

Managed By Property Sheet

The following table shows the UI labels of the **Managed By** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Name	Managed-By
Manager can update membership list	None. An ACE with the "Allow - Write Members" permission is added to the account identified by Name .
Office	The Physical-Delivery-Office-Name attribute of the account identified by Name .
Street	The Street-Address attribute of the account identified by Name .
City	The Locality-Name attribute of the account identified by Name .
State/province	The State-Or-Province-Name attribute of the account identified by Name .
Country/region	The Country-Name attribute of the account identified by Name .
Telephone number	The Telephone-Number attribute of the account identified by Name .
Fax number	The Facsimile-Telephone-Number attribute of the account identified by Name .

Object Property Sheet

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic discusses the Object property sheet, which is common to all object classes in the Active Directory Users and Computers snap-in.

Object Property Sheet

The following table shows the UI labels of the **Object** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENTS
Fully qualified domain name of object		This is the object's distinguished name in canonical form.
Object class	objectClass	
Created	whenCreated	
Modified	whenChanged	
Update Sequence Numbers: Current	uSNChanged	
Update Sequence Numbers: Original	uSNCreated	

Organizational Unit User Interface Mapping

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic describes the Organizational Unit (OU) object property sheets in the Active Directory Users and Computers snap-in.

General Property Sheet

The following table shows the UI labels of the **General** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Description	Description
Street	Street-Address
City	Locality-Name
State/Province	State-Or-Province-Name
Zip/Postal Code	Postal-Code
Country/Region	Country-Name

Managed By Property Sheet

The following table shows the UI labels of the **Managed By** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Name	Managed-By
Office	The Physical-Delivery-Office-Name attribute of the account identified by Name .
Street	The Street-Address attribute of the account identified by Name .
City	The Locality-Name attribute of the account identified by Name .
State/province	The State-Or-Province-Name attribute of the account identified by Name .
Country/region	The Country-Name attribute of the account identified by Name .

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Telephone number	The Telephone-Number attribute of the account identified by Name .
Fax number	The Facsimile-Telephone-Number attribute of the account identified by Name .

Printer Object User Interface Mapping

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic discusses the Printer object property sheets in the Active Directory Users and Computers snap-in.

General Property Sheet

The following table lists the UI labels of the **General** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Location	location
Model	driverName
Description	description
Color	printColor
Staple	printStaplingSupported
Double-sided	print DuplexSupported
Printing Menu	printRate
Maximum Resolution	printMaxResolutionSupported

Shared Folder Object User Interface Mapping

6/3/2022 • 2 minutes to read • [Edit Online](#)

This topic discusses the Shared Folder object property sheets in the Active Directory Users and Computers snap-in.

General Property Sheet

The following table lists the UI labels of the **General** property sheet.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Description	description
UNC Name	uNCName
Keywords	keywords

User Object User Interface Mapping

6/3/2022 • 3 minutes to read • [Edit Online](#)

The following tables identify the property pages supplied by the Active Directory Users and Computers snap-in. Each table identifies the user interface elements of the property page and the attribute associated with that user interface element. Because the property pages that are displayed by the Active Directory Users and Computers snap-in can be extended, it is not possible to provide this data for all of the pages displayed in a property sheet.

General Property Page

The following table lists UI labels of the **General** property page.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
First Name	<code>givenName</code>
Last Name	<code>sn</code>
Initials	<code>initials</code>
Display Name	<code>displayName</code>
Description	<code>description</code>
Office	<code>physicalDeliveryOfficeName</code>
Telephone Number	<code>telephoneNumber</code>
Telephone: Other	<code>otherTelephone</code>
E-Mail	<code>E-mail-Addresses</code>
Web Page	<code>wWWHomePage</code>
Web Page: Other	<code>url</code>

Account Property Page

The following table lists the UI labels of the **Account** property page.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENT

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENT
UserLogon Name	userPrincipalName	This field is taken from everything in the userPrincipalName attribute that precedes the last '@' character. The last '@' character and everything after it is placed in the suffix combo box.
User logon name (pre-Windows 2000)	sAMAccountname	No comment.
Logon Hours	logonHours	No comment.
Log On To	logonWorkstation	No comment.
Account is locked out	lockoutTime and lockoutDuration	If the lockoutTime attribute is not zero, the lockoutDuration attribute is added to lockoutTime and compared to the current date and time to determine if the account is locked out.
User must change password at next logon	pwdLastSet	No comment.
User cannot change password	N/A	This is the Change Password control in the ACL.
Other Account Options	userAccountControl	The remaining items in Account Options toggle bits in the userAccountControl attribute bitmask (flags in a DWORD).
Account Expires	accountExpires	The Account Expires control displays the date that the account will expire at the end of. The accountExpires attribute is stored as the date that the account expires on. Because of this, the date displayed in the Account Expires control will be displayed as one day earlier than the date contained in the accountExpires attribute.

Address Property Page

The following table lists the UI labels of the **Address** property page.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENT
Street	streetAddress	No comment.
P.O.Box	postOfficeBox	No comment.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENT
City	<code>l</code>	The <code>l</code> attribute name is a lowercase "L" as in Locale.
State/Province	<code>st</code>	No comment.
Zip/Postal Code	<code>postalCode</code>	No comment.
Country/Region	<code>c, co</code> , and <code>countryCode</code>	No comment.

Member Of Property Page

The following table lists UI labels of the **Member Of** property page.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENT
Member of	<code>memberOf</code>	Includes all of the groups in the UI list, except the primary group, which is represented in the AD through the <code>primaryGroupId</code> attribute.
Set Primary Group	<code>primaryGroupId</code>	LDAP: Tied to <code>primaryGroupToken</code> of the primary group.

Organization Property Page

The following table shows the UI labels of the **Organization** property page.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENT
Title	<code>title</code>	No comment.
Department	<code>department</code>	No comment.
Company	<code>company</code>	No comment.
Manager:Name	<code>manager</code>	No comment.
Direct Reports	<code>directReports</code>	No comment.

Profile Property Page

The following table lists the UI labels of the **Profile** property page.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES	COMMENT
Profile Path	profilePath	No comment.
Logon Script	scriptPath	No comment.
Home Folder: Local Path	homeDirectory	If Local path is selected, the local path is stored in the homeDirectory attribute.
Home Folder: Connect	homeDrive	If Connect is selected, the mapped drive is stored in the homeDrive attribute.
Home Folder: To	homeDirectory	If Connect is selected, the path is stored in the homeDirectory attribute.

Telephones Property Page

The following table lists the UI elements of the **Telephones** property page and their corresponding Active Directory attributes.

UI LABEL	ATTRIBUTE IN ACTIVE DIRECTORY DOMAIN SERVICES
Home	homePhone
Home: Other	otherHomePhone
Pager	pager
Pager: Other	otherPager
Mobile	mobile
Mobile: Other	otherMobile
Fax	facsimileTelephoneNumber
Fax: Other	otherFacsimileTelephoneNumber
IP phone	ipPhone
IP phone: Other	otherIpPhone
Notes	info

Environment, Sessions, Remote Control, and Terminal Services Profile

Property Pages

The **Environment**, **Sessions**, **Remote Control**, and **Terminal Services Profile** pages are supplied for a user object to support terminal services. The UI elements for these pages do not correspond to individual attributes. Instead, the settings are stored in private data within Active Directory Domain Services. The terminal services settings can be accessed with the [IADsTsUserEx](#) interface.

WMI Provider Reference for Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

The following topics provide reference information for the WMI provider for replication in Active Directory Domain Services:

- [Classes](#)

WMI Provider Classes in Active Directory Domain Services

6/3/2022 • 2 minutes to read • [Edit Online](#)

Active Directory Domain Services have the following classes for use with WMI scripting:

- [MSAD_DomainController](#)
- [MSAD_NamingContext](#)
- [MSAD_ReplCursor](#)
- [MSAD_ReplNeighbor](#)
- [MSAD_ReplPendingOp](#)

The provider itself is represented by the [ReplicationProvider1](#) class, which is instanced with the name "ReplProv1".

ReplicationProvider1 class

6/3/2022 • 5 minutes to read • [Edit Online](#)

The base class for the provider instance.

The following syntax is simplified from MOF code and includes all inherited properties.

Syntax

```
class ReplicationProvider1 : __Win32Provider
{
    string ClientLoadableCLSID;
    string CLSID;
    sint32 Concurrency;
    string DefaultMachineName;
    boolean Enabled;
    sint32 ImpersonationLevel = 0;
    sint32 InitializationReentrancy = 0;
    datetime InitializationTimeoutInterval;
    boolean InitializeAsAdminFirst;
    string Name;
    datetime OperationTimeoutInterval;
    boolean PerLocaleInitialization = FALSE;
    boolean PerUserInitialization = FALSE;
    boolean Pure = TRUE;
    string SecurityDescriptor;
    boolean SupportsExplicitShutdown;
    boolean SupportsExtendedStatus;
    boolean SupportsQuotas;
    boolean SupportsSendStatus;
    boolean SupportsShutdown;
    boolean SupportsThrottling;
    datetime UnloadTimeout;
    uint32 Version;
    string HostingModel;
};
```

Members

The **ReplicationProvider1** class has these types of members:

- [Properties](#)

Properties

The **ReplicationProvider1** class has these properties.

ClientLoadableCLSID

Data type: **string**

Access type: Read/write

Class identifier that WMI uses to determine whether or not to load a high performance provider into the client process or the WMI process. If both the provider and client are located on the same computer, WMI loads the provider in-process to the client by using **ClientLoadableCLSID** as the class identifier. When the provider and client are located on different computers, WMI loads the provider in-process to WMI. WMI also uses

ClientLoadableCLSID to support refresh operations.

For more information, see [Registering a High-Performance Provider](#).

This property is inherited from [__Win32Provider](#).

CLSID

Data type: **string**

Access type: Read/write

GUID that represents the class identifier (CLSID) of the provider COM object. This COM object must contain an implementation of the [IWbemProviderInit](#) interface.

This property is inherited from [__Win32Provider](#).

Concurrency

Data type: **sint32**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

DefaultMachineName

Data type: **string**

Access type: Read/write

Identifies the computer on which to start the provider. If the provider runs on the local computer it is **NULL**.

This property is inherited from [__Win32Provider](#).

Enabled

Data type: **boolean**

Access type: Read/write

If **TRUE**, this instance is enabled and can be used to complete client requests.

This property is inherited from [__Win32Provider](#).

HostingModel

Data type: **string**

Access type: Read-only

Qualifiers: **override** ("HostingModel")

Contains the hosting model of the provider.

ImpersonationLevel

Data type: **sint32**

Access type: Read/write

Reserved. The default value is zero (0).

This property is inherited from [__Win32Provider](#).

InitializationReentrancy

Data type: **sint32**

Access type: Read/write

Set of flags that provide information about serialization. The default value is zero (0).

This property is inherited from [__Win32Provider](#).

0

All initialization of this provider must be serialized.

1

All initializations of this provider in the same namespace must be serialized.

2

No initialization serialization is necessary.

InitializationTimeoutInterval

Data type: **datetime**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

InitializeAsAdminFirst

Data type: **boolean**

Access type: Read/write

Windows Server 2003: This property is disabled.

This property is inherited from [__Win32Provider](#).

Name

Data type: **string**

Access type: Read/write

Qualifiers: **Key**

The provider name.

This property is inherited from [__Win32Provider](#).

OperationTimeoutInterval

Data type: **datetime**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

PerLocaleInitialization

Data type: **boolean**

Access type: Read/write

If **TRUE**, the provider is initialized for each locale when a user connects to the same namespace more than one time using different locales. The default value is **FALSE**.

This property is inherited from [__Win32Provider](#).

PerUserInitialization

Data type: **boolean**

Access type: Read/write

If **TRUE**, the provider is initialized one time for each NT LAN Manager (NTLM) user that makes requests to the provider. If **FALSE** (default), the provider is initialized one time for all users.

This property is inherited from [__Win32Provider](#).

Pure

Data type: **boolean**

Access type: Read/write

If **TRUE**, the provider agrees to prepare to unload by calling [IUnknown::Release](#) on all outstanding interface points when WMI calls the **Release** method of its primary interface. Providers that must remain clients of WMI after they do not function as providers should set **Pure** to **FALSE**. The default setting is **TRUE**. For more information, see the Remarks section of this topic.

This property is inherited from [__Win32Provider](#).

SecurityDescriptor

Data type: **string**

Access type: Read/write

Security descriptor (SD) in the Security Descriptor Definition Language (SDDL) that determines the set of users that can successfully call [IWbemDecoupledRegistrar:Register](#) for the decoupled provider. For more information, see the [Security Descriptor Definition Language](#) topic in the Security section of the Windows SDK. This security descriptor is used only for decoupled providers, and does not affect other providers. For more information, see [Incorporating a Provider in an Application](#).

WMI performs access checks for decoupled providers that use the [IWbemProviderInit](#) and [IWbemObjectSink](#) interfaces. If the security descriptor is **NULL**, then only applications or services that run under the LocalSystem, NetworkService, LocalService accounts can run a decoupled provider.

The following string shows a decoupled provider to be run only by built-in Administrators."O:BAG:BAD:(A;;0x1;;BA)"

For more information about setting the **SecurityDescriptor** property, see [Maintaining WMI Security](#).

This property is inherited from [__Win32Provider](#).

SupportsExplicitShutdown

Data type: **boolean**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

SupportsExtendedStatus

Data type: **boolean**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

SupportsQuotas

Data type: **boolean**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

SupportsSendStatus

Data type: **boolean**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

SupportsShutdown

Data type: **boolean**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

SupportsThrottling

Data type: **boolean**

Access type: Read/write

Not used.

This property is inherited from [__Win32Provider](#).

UnloadTimeout

Data type: **datetime**

Access type: Read/write

[Date and Time Format](#) that specifies how long WMI allows the provider to remain idle before it is unloaded.

Typically, providers request that WMI wait no longer than five minutes.

For the current version of WMI, the value of this property is ignored. WMI unloads the provider based on the time-out value in an internal class in the \root namespace. It is recommended that providers set **UnloadTimeout**. For more information, see [Unloading a Provider](#).

This property is inherited from [__Win32Provider](#).

Version

Data type: **uint32**

Access type: Read/write

Version of the provider. The supported versions are 1 and 2. Version 2 strengthens validity checks for all associated property registrations, specifically the **ImpersonationLevel** property.

This property is inherited from [__Win32Provider](#).

Remarks

An instance of this class represents the WMI provider for Active Directory Domain services. The defaults are as follows:

- Name = "ReplProv1"
- ClsID = "{29288F43-39B1-40db-B41F-CE899450E911}"
- HostingModel = "NetworkServiceHost"

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	Windows Server 2008
Namespace	Root\Microsoft\ActiveDirectory
MOF	Replprov.mof
DLL	Replprov.dll

See also

[__Win32Provider](#)

MSAD_DomainController class

6/3/2022 • 2 minutes to read • [Edit Online](#)

Represents the domain controller (DC) on which the WMI provider is running.

Syntax

```
[dynamic, provider("ReplProv1")]
class MSAD_DomainController
{
    String DistinguishedName;
    String CommonName;
    String SiteName;
    String NTDSaGUID;
    boolean IsGC = FALSE;
    datetime TimeOfOldestReplSync;
    datetime TimeOfOldestReplAdd;
    datetime TimeOfOldestReplDel;
    datetime TimeOfOldestReplMod;
    datetime TimeOfOldestReplUpdRefs;
    boolean IsNextRIDPoolAvailable = FALSE;
    uint32 PercentOfRIDSLeft;
    boolean IsRegisteredInDNS;
    boolean IsAdvertisingToLocator = FALSE;
    boolean IsSysVolReady = FALSE;
};
```

Members

The **MSAD_DomainController** class has these types of members:

- [Methods](#)
- [Properties](#)

Methods

The **MSAD_DomainController** class has these methods.

METHOD	DESCRIPTION
ExecuteKCC	Invokes the Knowledge Consistency Checker to verify the replication topology.

Properties

The **MSAD_DomainController** class has these properties.

CommonName

Data type: **String**

Access type: Read-only

Gets the common name of the server object.

DistinguishedName

Data type: **String**

Access type: Read-only

Qualifiers: **key**

Gets the X.500 path of the **NTDS-DSA** object.

IsAdvertisingToLocator

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the locator service on the DC is advertising correctly. **TRUE** if the locator service on the DC is advertising correctly; otherwise, **FALSE**.

IsGC

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the DC is a Global Catalog server. **TRUE** if the DC is a Global Catalog server; otherwise, **FALSE**.

IsNextRIDPoolAvailable

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the DC has obtained another RID pool. **TRUE** if the DC has obtained another RID pool and allocation of RIDs on this DC can continue even if the current pool of RIDs is exhausted; otherwise, **FALSE**.

IsRegisteredInDNS

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the DC is registered correctly in DNS. **TRUE** if the DC is registered correctly in DNS; otherwise, **FALSE**.

IsSysVolReady

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the SysVol has published correctly. **TRUE** if the SysVol has published correctly; otherwise, **FALSE**.

NTDsaGUID

Data type: **String**

Access type: Read-only

Gets the GUID that is associated with the **NTDS-DSA** object in the configuration container. The **NTDS-DSA** object represents the Active Directory Domain Service DSA process on the server.

PercentOfRIDsLeft

Data type: **uint32**

Access type: Read-only

Gets the percentage of RIDs that are left in the current RID pool on this DC.

SiteName

Data type: **String**

Access type: Read-only

Gets the site in which the DC exists.

TimeOfOldestReplAdd

Data type: **datetime**

Access type: Read-only

Gets the timestamp of the oldest replication add operation that is still in the queue on this domain controller.

TimeOfOldestReplDel

Data type: **datetime**

Access type: Read-only

Gets the timestamp of the oldest replication delete operation that is still in the queue on this domain controller.

TimeOfOldestReplMod

Data type: **datetime**

Access type: Read-only

Gets the timestamp of the oldest replication modify operation that is still in the queue on this domain controller.

TimeOfOldestReplSync

Data type: **datetime**

Access type: Read-only

Gets the timestamp of the oldest replication sync operation that is still in the queue on this domain controller.

TimeOfOldestReplUpdRefs

Data type: **datetime**

Access type: Read-only

Gets the timestamp of the oldest replication update operation that is still in the queue on this domain controller.

Requirements

Requirement	Value
Minimum supported client	None supported

REQUIREMENT	VALUE
Minimum supported server	Windows Server 2008
Namespace	Root\Microsoft\ActiveDirectory
MOF	Replprov.mof
DLL	Replprov.dll

ExecuteKCC method of the MSAD_DomainController class

6/3/2022 • 2 minutes to read • [Edit Online](#)

Calls the [DsReplicaConsistencyCheck](#) function, which invokes the Knowledge Consistency Checker (KCC) to verify the replication topology.

Syntax

```
void ExecuteKCC(  
    [in] uint32 TaskID,  
    [in] uint32 dwFlags  
>;
```

Parameters

TaskID [in]

The task that the KCC should execute. **DS_KCC_TASKID_UPDATE_TOPOLOGY** is the only value that is currently supported.

dwFlags [in]

A set of flags that modify the behavior of the **ExecuteKCC** method. This parameter can be zero or a combination of one or more of the following values.

DS_KCC_FLAG_ASYNC_OP

The task is queued and then the function returns without waiting for the task to complete.

DS_KCC_FLAG_DAMPED

The task will not be added to the queue if another queued task will run soon.

Return value

This method does not return a value.

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	Windows Server 2008
Namespace	Root\MicrosoftActiveDirectory

REQUIREMENT	VALUE
MOF	Replprov.mof
DLL	Replprov.dll

See also

[MSAD_DomainController](#)

MSAD_NamingContext class

6/3/2022 • 2 minutes to read • [Edit Online](#)

Represents a particular naming context (NC) on the domain controller.

Syntax

```
[dynamic, provider("ReplProv1")]
class MSAD_NamingContext
{
    String DistinguishedName;
    boolean IsFullReplica = FALSE;
};
```

Members

The **MSAD_NamingContext** class has these types of members:

- [Properties](#)

Properties

The **MSAD_NamingContext** class has these properties.

DistinguishedName

Data type: **String**

Access type: Read-only

Qualifiers: **key**

Gets the X.500 path of the NC.

IsFullReplica

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the NC is read/write. **TRUE** if the NC is read/write; **FALSE** if the NC is read-only.

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	Windows Server 2008
Namespace	Root\MicrosoftActiveDirectory

REQUIREMENT	VALUE
MOF	Replprov.mof
DLL	Replprov.dll

MSAD_ReplCursor class

6/3/2022 • 2 minutes to read • [Edit Online](#)

Provides inbound replication state information about all replicas of a given naming context (NC).

Syntax

```
[dynamic, provider("ReplProv1")]
class MSAD_ReplCursor
{
    String NamingContextDN;
    String SourceDsaInvocationID;
    uint64 USNAttributeFilter;
    String SourceDsaDN;
    datetime TimeOfLastSuccessfulSync;
};
```

Members

The **MSAD_ReplCursor** class has these types of members:

- [Properties](#)

Properties

The **MSAD_ReplCursor** class has these properties.

NamingContextDN

Data type: **String**

Access type: Read-only

Qualifiers: **key**

Gets the X.500 path for the naming context (NC) that holds this cursor.

SourceDsaDN

Data type: **String**

Access type: Read-only

Gets the X.500 path for the directory system agent (DSA) that represents the source domain controller (DC).

SourceDsaInvocationID

Data type: **String**

Access type: Read-only

Qualifiers: **key**

Gets the invocation identifier of the originating server to which the **USNAttributeFilter** corresponds.

TimeOfLastSuccessfulSync

Data type: **datetime**

Access type: Read-only

Gets the timestamp of the last successful replication sync with the source DSA. Indicates the time when this DC last saw changes made on the source DSA for this NC.

USNAttributeFilter

Data type: **uint64**

Access type: Read-only

Gets the maximum update sequence number to which the destination server can indicate that it has recorded all changes originated by the given server at update sequence numbers less than, or equal to, this update sequence number. This property is used to filter changes that the destination server has already applied at replication source servers.

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	Windows Server 2008
Namespace	Root\Microsoft\ActiveDirectory
MOF	Replprov.mof
DLL	Replprov.dll

See also

[DS_REPL_CURSOR](#)

MSAD_ReplNeighbor class

6/3/2022 • 7 minutes to read • [Edit Online](#)

Represents the **DS_REPL_NEIGHBOR** structure, which contains the inbound replication state information for a particular naming context (NC) and source server pair, as returned by the **DsReplicaGetInfo** function.

Syntax

```
[dynamic, provider("ReplProv1")]
class MSAD_ReplNeighbor
{
    String NamingContextDN;
    String SourceDsaObjGuid;
    String NamingContextObjGuid;
    String SourceDsaDN;
    String SourceDsaAddress;
    String SourceDsaInvocationID;
    String AsyncIntersiteTransportDN;
    String AsyncIntersiteTransportObjGuid;
    uint64 USNLastObjChangeSynced;
    uint64 USNAttributeFilter;
    datetime TimeOfLastSyncSuccess;
    datetime TimeOfLastSyncAttempt;
    uint32 LastSyncResult;
    uint32 NumConsecutiveSyncFailures;
    uint32 ReplicaFlags;
    boolean Writeable = FALSE;
    boolean SyncOnStartup = FALSE;
    boolean DoScheduledSyncs = FALSE;
    boolean UseAsyncIntersiteTransport = FALSE;
    boolean TwoWaySync = FALSE;
    boolean FullSyncInProgress = FALSE;
    boolean FullSyncNextPacket = FALSE;
    boolean NeverSynced = FALSE;
    boolean IgnoreChangeNotifications = FALSE;
    boolean DisableScheduledSync = FALSE;
    boolean CompressChanges = FALSE;
    boolean NoChangeNotifications = FALSE;
    String SourceDsaSite;
    String SourceDsaCN;
    String Domain;
    boolean IsDeletedSourceDsa = FALSE;
    uint32 ModifiedNumConsecutiveSyncFailures;
};
```

Members

The **MSAD_ReplNeighbor** class has these types of members:

- [Methods](#)
- [Properties](#)

Methods

The **MSAD_ReplNeighbor** class has these methods.

METHOD	DESCRIPTION
SyncNamingContext	Synchronizes a destination naming context with one of its sources.

Properties

The **MSAD_ReplNeighbor** class has these properties.

AsyncIntersiteTransportDN

Data type: **String**

Access type: Read-only

Gets the X.500 path of the [interSiteTransport](#) object that corresponds to the transport over which replication is performed. Set to **NULL** for RPC/IP replication.

AsyncIntersiteTransportObjGuid

Data type: **String**

Access type: Read-only

Gets the GUID of the intersite transport object that corresponds to the **AsyncIntersiteTransportDN** property.

CompressChanges

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_COMPRESS_CHANGES** flag has been set in the **ReplicaFlags** property.

DisableScheduledSync

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_DISABLE_SCHEDULED_SYNC** flag has been set in the **ReplicaFlags** property.

Domain

Data type: **String**

Access type: Read-only

Gets the canonical name of the domain of the replicated NC.

DoScheduledSyncs

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_DO_SCHEDULED_SYNCS** flag has been set in the **ReplicaFlags** property.

FullSyncInProgress

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the DS_REPL_NBR_FULL_SYNC_IN_PROGRESS flag has been set in the **ReplicaFlags** property.

FullSyncNextPacket

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the DS_REPL_NBR_FULL_SYNC_NEXT_PACKET flag has been set in the **ReplicaFlags** property.

IgnoreChangeNotifications

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the DS_REPL_NBR_IGNORE_CHANGE_NOTIFICATIONS flag has been set in the **ReplicaFlags** property.

IsDeletedSourceDsa

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether this connection represents a source DC that has been deleted. **TRUE** if this connection represents a source DC that has been deleted; otherwise, **FALSE**. By design, the DS will continue to replicate these connections for some time for some time after the source DC is deleted.

LastSyncResult

Data type: **uint32**

Access type: Read-only

Gets the **HRESULT** error code for the last replication attempt.

ModifiedNumConsecutiveSyncFailures

Data type: **uint32**

Access type: Read-only

Gets the number of consecutive failed replication attempts, not including the connections that are expected to fail. For example, if the **IsDeletedSourceDsa** property is set to **TRUE**, it is expected to fail.

NamingContextDN

Data type: **String**

Access type: Read-only

Qualifiers: **key**

Gets the X.500 path for the NC that is replicated by this connection.

NamingContextObjGuid

Data type: **String**

Access type: Read-only

Gets the GUID for the replicated NC.

NeverSynced

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_NEVER_SYNCED** flag has been set in the **ReplicaFlags** property.

NoChangeNotifications

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_NO_CHANGE_NOTIFICATIONS** flag has been set in the **ReplicaFlags** property.

NumConsecutiveSyncFailures

Data type: **uint32**

Access type: Read-only

Gets the number of consecutive failed replication attempts.

ReplicaFlags

Data type: **uint32**

Access type: Read-only

Gets the set of flags that specify attributes and options for the replication data. This property can be zero or a combination of one or more of the following flags.

DS_REPL_NBR_WRITEABLE (16 (0x10))

The local copy of the naming context is writable.

DS_REPL_NBR_SYNC_ON_STARTUP (32 (0x20))

Replication of this naming context from this source is attempted when the destination server is booted. This flag usually only applies to intra-site neighbors.

DS_REPL_NBR_DO_SCHEDULED_SYNCS (64 (0x40))

Perform replication on a schedule. This flag is usually set unless the schedule for this naming context or source is "never", that is, the empty schedule.

DS_REPL_NBR_USE_ASYNC_INERSITE_TRANSPORT (128 (0x80))

Perform replication indirectly through the Inter-Site Messaging Service. This flag is set only when replicating over SMTP. This flag is not set when replicating over inter-site RPC/IP.

DS_REPL_NBR_TWO_WAY_SYNC (512 (0x200))

If set, indicates that when inbound replication is complete, the destination server must tell the source server to synchronize in the reverse direction. This feature is used in dial-up scenarios where only one of the two servers can initiate a dial-up connection. For example, this option could be used in a corporate headquarters and branch office, where the branch office connects to the corporate headquarters over the Internet by means of a dial-up ISP connection.

DS_REPL_NBR_RETURN_OBJECT_PARENTS (2048 (0x800))

This neighbor is in a state where it returns parent objects before children objects. It goes into this state after it receives a child object before its parent.

DS_REPL_NBR_FULL_SYNC_IN_PROGRESS (65536 (0x10000))

The destination server is performing a full synchronization from the source server. Full synchronizations do not use vectors that create updates (such as [DS_REPL_CURSORS](#)) for filtering updates. Full synchronizations are not used as a part of the default replication protocol.

DS_REPL_NBR_FULL_SYNC_NEXT_PACKET (131072 (0x20000))

The last packet from the source indicated a modification of an object that the destination server has not yet created. The next packet to be requested instructs the source server to put all attributes of the modified object into the packet.

DS_REPL_NBR_NEVER_SYNCED (2097152 (0x200000))

A synchronization has never been successfully completed from this source.

DS_REPL_NBR_PREEMPTED (16777216 (0x1000000))

The replication engine has temporarily stopped processing this neighbor in order to service another higher-priority neighbor, either for this partition or for another partition. The replication engine will resume processing this neighbor after the higher-priority work is completed.

DS_REPL_NBR_IGNORE_CHANGE_NOTIFICATIONS (67108864 (0x4000000))

This neighbor is set to disable notification-based synchronizations. Within a site, domain controllers synchronize with each other based on notifications when changes occur. This setting prevents this neighbor from performing synchronizations that are triggered by notifications. The neighbor will still do synchronizations based on its schedule, or in response to manually requested synchronizations.

DS_REPL_NBR_DISABLE_SCHEDULED_SYNC (134217728 (0x8000000))

This neighbor is set to not perform synchronizations based on its schedule. The only way this neighbor will perform synchronizations is in response to change notifications or to manually requested synchronizations.

DS_REPL_NBR_COMPRESS_CHANGES (268435456 (0x10000000))

Changes received from this source are to be compressed. Compression usually occurs only if the source server is in a different site.

DS_REPL_NBR_NO_CHANGE_NOTIFICATIONS (536870912 (0x20000000))

No change notifications should be received from this source. Usually set only if the source server is in a different

site.

DS_REPL_NBR_PARTIAL_ATTRIBUTE_SET (1073741824 (0x40000000))

This neighbor is in a state where it is rebuilding the contents of this replica because of a change in the partial attribute set.

SourceDsaAddress

Data type: **String**

Access type: Read-only

Gets the DNS address of the source DC.

NOTE

This string contains a modified GUID, not the commonly used canonical DNS name.

SourceDsaCN

Data type: **String**

Access type: Read-only

Gets the object path component for the DSA that represents the source DC. This string is often similar to the computer name but is not always identical.

SourceDsaDN

Data type: **String**

Access type: Read-only

Gets the X.500 path for the DSA that represents the source DC.

SourceDsInvocationID

Data type: **String**

Access type: Read-only

Gets the invocation ID that was used by the source server as of the last replication.

SourceDsaObjGuid

Data type: **String**

Access type: Read-only

Qualifiers: **key**

Gets the GUID for the directory service agent (DSA) that represents the source domain controller (DC).

SourceDsaSite

Data type: **String**

Access type: Read-only

Gets the site that contains the source DC.

SyncOnStartup

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_SYNC_ON_STARTUP** flag has been set in the **ReplicaFlags** property.

TimeOfLastSyncAttempt

Data type: **datetime**

Access type: Read-only

Gets the timestamp for the last replication attempt.

TimeOfLastSyncSuccess

Data type: **datetime**

Access type: Read-only

Gets the timestamp for the last successful replication attempt.

TwoWaySync

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_TWO_WAY_SYNC** flag has been set in the **ReplicaFlags** property.

UseAsyncIntersiteTransport

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the **DS_REPL_NBR_USE_ASYNC_INERSITE_TRANSPORT** flag has been set in the **ReplicaFlags** property.

USNAttributeFilter

Data type: **uint64**

Access type: Read-only

Gets the **USNLastObjChangeSynced** property value at the end of the last successful completed replication cycle. Zero if there were no successfully completed replication cycles.

USNLastObjChangeSynced

Data type: **uint64**

Access type: Read-only

Gets the **unchanged** attribute value of the last object update that was received.

Writeable

Data type: **boolean**

Access type: Read-only

Gets the value that indicates whether the DS_REPL_NBR_WRITEABLE flag has been set in the **ReplicaFlags** property.

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	Windows Server 2008
Namespace	Root\MicrosoftActiveDirectory
MOF	Replprov.mof
DLL	Replprov.dll

SyncNamingContext method of the MSAD_ReplNeighbor class

6/3/2022 • 2 minutes to read • [Edit Online](#)

Calls the [DsReplicaSync](#) function that synchronizes a destination naming context with one of its sources.

Syntax

```
void SyncNamingContext(  
    [in] uint32 Options  
);
```

Parameters

Options [in]

Additional data that determines how the method processes the request. This parameter can be a combination of the following values.

DS_REPSYNC_ADD_REFERENCE

Causes the source directory system agent (DSA) to verify that the local DSA is present in the source replicates-to list. If the local DSA is not present, it is added. This ensures that the source sends change notifications.

DS_REPSYNC_ALL_SOURCES

This value is not supported.

Windows Server 2008 R2, Windows Server 2008 and Windows Server 2003: Synchronizes from all sources.

DS_REPSYNC_ASYNCHRONOUS_OPERATION

Performs this operation asynchronously.

Windows Server 2008 R2, Windows Server 2008 and Windows Server 2003: Required when using DS_REPSYNC_ALL_SOURCES.

DS_REPSYNC_FORCE

Synchronizes even if the link is currently disabled.

DS_REPSYNC_FULL

Synchronizes starting from the first Update Sequence Number (USN).

DS_REPSYNC_INERSITE_MESSAGING

Synchronizes using an ISM.

DS_REPSYNC_NO_DISCARD

Does not discard this synchronization request, even if a similar synchronization is pending.

DS_REPSYNC_PERIODIC

Indicates that this operation is a periodic synchronization request that is scheduled by the administrator.

DS_REPSYNC_URGENT

Indicates that this operation is a notification of an update that is marked as urgent.

DS_REPSYNC_WRITEABLE

Indicates that this replica is writable. If this flag is not set, the replica is read-only.

Return value

This method does not return a value.

Requirements

REQUIREMENT	VALUE
Minimum supported client	None supported
Minimum supported server	Windows Server 2008
Namespace	Root\Microsoft\ActiveDirectory
MOF	Replprov.mof
DLL	Replprov.dll

See also

[MSAD_ReplNeighbor](#)

MSAD_ReplPendingOp class

6/3/2022 • 2 minutes to read • [Edit Online](#)

Represents the [DS_REPL_OP](#) structure, which describes a replication task that is currently executing or pending execution. This structure is returned by the [DsReplicaGetInfo](#) function.

Syntax

```
[dynamic, provider("ReplProv1")]
class MSAD_ReplPendingOp
{
    uint32 SerialNumber;
    uint32 PositionInQ;
    datetime OpStartTime;
    datetime TimeEnqueued;
    uint32 Priority;
    uint32 OpType;
    uint32 Options;
    String NamingContextDN;
    String NamingContextObjGuid;
    String DsaDN;
    String DsaAddress;
    String DsaObjGuid;
};
```

Members

The **MSAD_ReplPendingOp** class has these types of members:

- [Properties](#)

Properties

The **MSAD_ReplPendingOp** class has these properties.

DsaAddress

Data type: **String**

Access type: Read-only

Gets the transport-specific network address of the remote server that is associated with this operation. **NULL** if there is no remote server that is associated with this operation.

DsaDN

Data type: **String**

Access type: Read-only

Gets the X.500 path of the DSA that is associated with the remote server that corresponds to this operation. **NULL** if no remote server corresponds to this operation.

DsaObjGuid

Data type: **String**

Access type: Read-only

Gets the value of the [objectGuid](#) attribute of the DSA that is identified by the **DsaDN** property.

NamingContextDN

Data type: **String**

Access type: Read-only

Gets the X.500 path of the naming context (NC) that is associated with this operation.

NamingContextObjGuid

Data type: **String**

Access type: Read-only

Gets the [objectGuid](#) attribute of the NC that is identified by the **NamingContextDN** property.

OpStartTime

Data type: **datetime**

Access type: Read-only

Gets the time when the operation was started. **NULL** if this operation is still in the queue.

Options

Data type: **uint32**

Access type: Read-only

Gets the set of flags that provides additional data about the operation. The contents of this member are determined by the value of the **OpType** property.

DS_REPL_OP_TYPE_SYNC

Contains zero or a combination of one or more of the **DS_REPSYNC_*** values as defined for the *Options* parameter in [DsReplicaSync](#).

DS_REPL_OP_TYPE_ADD

Contains zero or a combination of one or more of the **DS_REPADD_*** values as defined for the *Options* parameter in [DsReplicaAdd](#).

DS_REPL_OP_TYPE_DELETE

Contains zero or a combination of one or more of the **DS REPDEL_*** values as defined for the *Options* parameter in [DsReplicaDel](#).

DS_REPL_OP_TYPE_MODIFY

Contains zero or a combination of one or more of the **DS_REPMOD_*** values as defined for the *Options* parameter in [DsReplicaModify](#).

DS_REPL_OP_TYPE_UPDATE_REFS

Contains zero or a combination of one or more of the **DS_REPSUPD_*** values as defined for the *Options* parameter in [DsReplicaUpdateRefs](#).

parameter in [DsReplicaUpdateRefs](#).

OpType

Data type: **uint32**

Access type: Read-only

Gets the **DS_REPL_OP_TYPE** value that indicates the type of operation that this class represents.

PositionInQ

Data type: **uint32**

Access type: Read-only

Gets the position of this operation in the queue.

Priority

Data type: **uint32**

Access type: Read-only

Gets the priority of this operation. Higher-priority tasks are executed first. The priority is calculated by the server based on the type of operation that this class represents and the parameters of the operation.

SerialNumber

Data type: **uint32**

Access type: Read-only

Qualifiers: **key**

Gets the ID of the operation, which is unique per-machine and per-boot.

TimeEnqueued

Data type: **datetime**

Access type: Read-only

Gets the time at which this operation was added to the queue.

Requirements

Requirement	Value
Minimum supported client	None supported
Minimum supported server	Windows Server 2008
Namespace	Root\Microsoft\ActiveDirectory
MOF	Replprov.mof
DLL	Replprov.dll