Windows 2000 Server

Chapter 4 - Active Directory Schema

Information about the schema for Active Directory™, the directory service that is included with Microsoft® Windows® 2000 Server, is essential for anyone who is managing the directory service or who is developing directory-aware applications for Microsoft® Windows® 2000–based servers. The material in this chapter covers how the schema for Active Directory is defined, how to modify ("extend") the schema, what safeguards are implemented in the service to protect the schema from being corrupted during the process of modifying it, what automatic checks are included to ensure schema consistency, and what precautions to take when you install new applications on a domain controller.

In This Chapter

Related Information in the Resource Kit

- For an overview of the Active Directory physical structure, see "Active Directory Data Storage" in this book.
- For more information about Active Directory tools, see "Active Directory Diagnostics, Troubleshooting, and Recovery" in this book.
- For information about service publication, see "Service Publication in Active Directory" in this book.

Introduction to the Active Directory Schema

In Active Directory, the schema contains definitions for the universe of objects that can be stored in the directory, and it enforces the rules that govern both the structure and the content of the directory. The schema consists of a set of classes, attributes, and syntaxes that represent an instance of one or more classes in the schema. A *class* is a category of objects that share a set of common characteristics. It is a formal description of a discrete, identifiable type of object that can be stored in the directory. Each object in the directory is an instance of one or more classes in the schema. An *attribute* describes the characteristics of some aspect of an object. Attributes define the types of information that an object can hold. For each class, the schema specifies the mandatory attributes and optional attributes that constitute the set of shared characteristics of the class. The values assigned to attributes define specific characteristics. A *syntax* is the data type of a particular attribute. Syntaxes determine what data type an attribute can have. Active Directory uses a set of standard syntaxes. The predefined syntaxes do not actually appear in the directory, and you cannot add new syntaxes. An everyday example of an object is a vehicle, which can belong to the class of trucks, the class of motorcycles, or the class of cars, and so forth. A car can be described by its make, model, and color. These are some of the attributes of the car. In the example of the car, the possible values for the color of the car might be red, blue, or gray. The syntax for color might be the nomenclature (such as 2B1R2Y) that denotes specific combinations of primary colors that comprise what one sees as the colors of automotive paints.

The schema specifies the relationships between classes of objects. Each object stored in the directory is an instance of one or more classes in the schema. *User*, *Computer*, and *printQueue* are examples of classes in Active Directory. For example, if the schema contains a class called *User*, the user accounts, Sue and Mary, are two objects in the directory that are instances of the class *User*. The object Mary might contain an optional attribute defined for this class called *phoneNumber*. This attribute for the object Mary of the class *User* might have the value 555-0100.

For example, the attribute *phoneNumber* can be defined to take values of the syntax String(numeric), which means that the value can contain only the digits 0 through 9.

The base schema that ships in Microsoft Windows 2000 contains all of the class and attribute definitions that are used by Windows 2000 and Windows 2000 components.

The schema itself is represented in Active Directory by a set of objects known as "schema objects." For each class in the schema, there is a schema object that defines the class. This object is called a *classSchema* object. For each attribute in the schema, there is also a schema object that defines the attribute. This object is called an *attributeSchema* object. Therefore, every class is actually an instance of the *classSchema* class, and every attribute is an instance of the *attributeSchema* class. Storing the schema in the directory has many advantages. One example is that when user applications locate the schema in the directory, they can read the schema to discover what types of objects and properties are available.

Administrators and applications can extend the schema by adding new attributes and classes or by modifying existing ones. Schema definitions are required by applications that need to create or modify objects in Active Directory. Applications that are "directory-enabled" are programmed to recognize the attributes and syntaxes that are required to interact with the directory.

Location of the Schema in Active Directory

The objects stored in Active Directory are arranged in a logical hierarchy called the *Directory Information Tree (DIT)*. Active Directory includes a preconfigured database (commonly referred to as the *base DIT*) that contains the information that is required to install and run Windows 2000 and Active Directory. The base DIT is installed during a fresh install of a Windows 2000 domain controller. One section of the base DIT is the base schema.

The Directory Information Tree is divided into directory partitions. A *directory partition* is a tree of directory objects that forms a unit of replication in Active Directory.

Schema objects are located in the Schema container. The Schema container is not a container in the sense of a special type of Active Directory object that contains other objects; the Schema container is a special purpose object class. The Schema container (cn=schema,cn=configuration,dc=< forest root *domainName*>) contains all of the class and attribute definitions that are required to locate objects in Active Directory and to create new objects. It is the topmost object of the schema directory partition.

The relationship of the schema partition and the Configuration and Schema containers is illustrated in Figure 4.1.

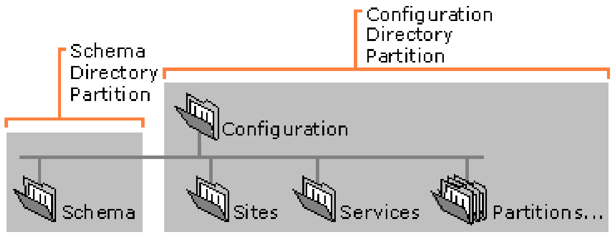


Figure 4.1 Location of the Schema Container

Note The schema is a directory partition in its own right to prevent potential dependency problems that can arise because of the way directory objects are replicated. For more information about the schema directory partition and why the schema is a separate directory partition, see "Name Resolution in Active Directory" in this book.

Finding the Schema Container

Every Active Directory object can be referenced by a unique and unambiguous name known as the *distinguished name* (also known as a "DN"). The distinguished name identifies the *domain* that holds the object as well as the complete path through the container hierarchy by which the object is reached. The distinguished name of the Schema container can be expressed as follows:

```
cn=schema,cn=configuration,dc=< forest root domainname>
```

For more information about the distinguished name, see "Active Directory Logical Structure" in this book.

You can view the contents of the Schema container by using the Active Directory Schema console in Microsoft Management Console (MMC). You also can bind to the schema directory partition and view schema objects by using the Active Directory Service Interfaces (ADSI) Edit MMC console or the Ldp tool.

Note The ADSI Edit snap-in is not one of the default MMC snap-ins that is provided with Windows 2000 Server. To use ADSI Edit and Ldp, install the Support Tools that are located in the Support\Tools folder on the Windows 2000 Server operating system CD. To install the tools, double-click the **Setup** icon in that folder. For more information about using ADSI Edit and Ldp, see Microsoft® Windows® 2000 Support Tools Help. For information about installing and using the Windows 2000 Support Tools and Support Tools Help, see the file Sreadme.doc in the Support\Tools folder of the Windows 2000 operating system CD. For information about diagnosing and troubleshooting problems using the Ldp tool, see "Active Directory Diagnostics, Troubleshooting, and Recovery" in this book.)

It is possible to locate the Schema container without knowing the domain name. Installation scripts and other applications that might not know what domain they are to be used in are able to gain access to the schema because they bind to a special entry at the top of the logical namespace called rootDSE, which provides the schema location. The *rootDSE* (DSA-specific Entry) represents the top of the logical namespace and, therefore, the top of the Lightweight Directory Access Protocol (LDAP) search tree. The attributes of rootDSE identify, among other things, the directory partitions — that is, the domain, schema, and configuration directory partitions — as well as the forest root domain directory partition. One attribute, *schemaNamingContext*, provides the location of the schema so that applications that are connecting to any domain controller can find and read the schema. (For more information about the rootDSE, see "Name Resolution in Active Directory" in this book.)

To identify the Schema directory partition by using ADSI Edit

1. Start the ADSI Edit console in MMC.
2. Right-click **ADSI Edit**, and then click **Connect to**.

   The **Connection dialog box** is displayed.
3. In the **Connection Point** check box, make sure **Naming Context** is selected.
4. Select **RootDSE** from the **Naming Context** box, and then click **OK**.
5. In the Console Tree, double-click **My Connection**.

   The **RootDSE folder** is displayed.
6. Right-click the **RootDSE** folder, and then click **Properties**.
7. In the **Select property to view** dialog box, select **schemaNamingContext** from the list of properties ("attributes").
8. In **Attribute Values**, view the **Value(s)** box to see the distinguished name of the schema directory partition.

Note The Schema Management snap-in is not one of the default MMC snap-ins that is provided with Windows 2000 Server. To make it appear in the list of available snap-ins, you must install the admin tools package (Adminpak.msi). To register the Schema Management snap-in, open your %SystemRoot%\System32 folder and run **Regsvr32** Schmmgmt.dll from the command prompt or from the **Run** command on the **Start** menu.

Subschema Subentry

The rootDSE also carries a mandatory attribute called the *subSchemaSubEntry*. Its value is the distinguished name of a *subSchema* object in the directory in which the server makes available the attributes (in *attributeTypes*) and classes (in *objectClasses*) of which the Active Directory schema is comprised. This special object, an instance of the unique *subSchema* class, is used for administering information about the schema, in particular the object classes and attribute types that are supported. This enables client applications to retrieve the information by querying the *subSchema* entry. Clients must only retrieve attributes from a *subSchema* entry by requesting a base object search of the entry, where the LDAP search filter is "(*objectClass=subSchema*)." The location of the *subSchemaSubEntry* container is as follows:

```
CN=Aggregate,CN=Schema,CN=Configuration,DC=<DomainName>,DC=<DomainRoot>
```

Schema Files

Active Directory data is distributed among all domain controllers in the forest. No single domain controller stores all Active Directory data for the entire forest, but every domain controller does hold a copy of the schema. The Active Directory data that is in use on a particular domain controller is stored in a file named Ntds.dit. The location of the Ntds.dit file is an option that is set during the promotion process while you create the directory. The default location for the database and database log files is %SystemRoot%\Ntds. (For more information about the Ntds.dit file, see "Active Directory Data Storage" in this book.)

Another file, the Schema.ini initialization file contains the information that is necessary for creating the default directory objects and the default security for the DIT, as well as the Active Directory display specifiers. For information about display specifiers, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources . Although this file is named Schema.ini, the schema itself is actually preloaded and is contained in the base version of Ntds.dit that is installed by the Active Directory Installation wizard.

Active Directory Schema Objects

The attributes and classes in Active Directory are stored in the Schema container as directory objects called *schema objects*. The Schema container itself is represented in Active Directory by an object that is an instance of the Directory Management Domain (*dMD*) class.

For more information about Active Directory *attributeSchema* and classSchema objects, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

attributeSchema Objects

Attributes are data items that are used to describe the classes that are defined in the schema. They are defined in the schema separately from the classes, which allows a single attribute definition to be applied to many classes.

Attributes are *attributeSchema* objects. Each *attributeSchema* object is an instance of the *attributeSchema* class. The *attributeSchema* object lists, among other things, the following information:

- The LDAP display name of the attribute.
- The object identifier for the attribute.
- The globally unique identifier (GUID) for the attribute.
- The syntax of the attribute.
- The range for the attribute. For integers, range defines the minimum and maximum value; for strings, range defines the minimum and maximum length.
- Whether the attribute is a multivalue attribute. Note that multivalue attributes hold a set of values with no particular order. There is no guarantee that multivalue attributes are ever going to be returned in the order in which they were stored (or in any other order).
- Whether and how the attribute is indexed.

Single-Value or Multivalue Attributes

Attributes might be single-value or multivalue. Single-value and multivalue attributes are defined by the *singleValued* attribute being set to TRUE or FALSE. The Active Directory Schema console reports this as "single-valued" or "multivalued" rather than as an attribute-value pair.

A multivalue attribute can contain multiple values, all of uniform syntax. Note that multivalue attributes hold a set of values with no particular order. There is no guarantee that multivalue properties are ever going to be returned in the order in which they were stored (or any other order).

Note The LDAP protocol reads a multivalue attribute as a single entity. This can be inconvenient or even impossible when the number of values in a multivalue attribute becomes large. An Internet draft titled "Incremental Retrieval of Multivalued Properties" defines an option called Range that can be specified as part of an attribute description to retrieve the values of a multivalue attribute incrementally. Servers might or might not honor the range option. Servers that support the range option include the object identifier 1.2.840.113556.1.4.802 in the *supportedControls* operational attribute on the rootDSE. Clients must not use the range option unless this object identifier is present. The range option is a constant, case-insensitive string value (Range=), followed by a range-specifier that lists the initial and terminal values in the range.

For more information about the retrieval of multivalue attributes, see the Internet Engineering Task Force (IETF) link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources . Follow the links to Internet Drafts, and then use a keyword search.

Indexed Attributes

Making an attribute indexed means that directory searches involving that attribute are going to be more efficient than if the attribute had no index. Attributes are indexed when the least significant bit in their *searchFlags* attribute is set to the value 1. Changing the value of the bit to 1 dynamically builds an index; changing the value to 0 or deleting it drops an index for the attribute in question. The index is built automatically by a background thread on the directory server.

Ideally, indexed attributes are single value with highly unique values that are evenly distributed across the set of instances. Multivalue attributes can be indexed, but the cost to build the index is larger in terms of storage and updating. Even with single-value attributes, keep in mind that the more indexed attributes a class has, the longer it takes to modify or create instances of the class.

Attributes for attributeSchema Class Objects

Attributes for the *attributeSchema* class are described in Table 4.1.

Table 4.1 Attributes for the *attributeSchema* Class

Attribute	Syntax	Mandatory	Multi-value	Description
cn	Unicode	Yes	No	Descriptive relative distinguished name for the schema object.
attributeID	Object identifier	Yes	No	Object identifier that uniquely identifies this attribute.
lDAPDisplayName	Unicode	Yes, but filled in automatically	No	Name by which LDAP clients identify this attribute.
schemaIDGUID	String(Octet)	Yes	No	GUID that uniquely identifies this attribute.
mAPIID	Integer	No	No	Integer by which Messaging Application Programming Interface (MAPI) clients identify this attribute.
attributeSecurityGUID	GUID	No	No	GUID by which the security system identifies the property set of this attribute.
attributeSyntax	Object identifier	Yes	No	Syntax object identifier of this attribute.
oMSyntax	Integer	Yes	No	Syntax of this attribute as defined by the XAPIA X/Open Object Model (XOM) specification.
isSingleValued	BOOL	Yes	No	Indicates whether this attribute is a single-value or multivalue attribute. Note that multivalue attributes hold a set of values with no particular order. There is no guarantee that multivalue attributes are ever going to be returned in the order in which they were stored (or in any other order).
extendedCharsAllowed	BOOL	No	No	Indicates whether extended characters are allowed in the value of this attribute. Only applies to attributes of syntax String(teletex).
rangeLower	Integer	No	No	Lower range of values that are allowed for this attribute.[2]
rangeUpper	Integer	No	No	Upper range of values that are allowed for this attribute.[2]
systemFlags	Integer	No	No	Flags that determine specific system operations. Note: this attribute cannot be set or modified. The systemFlags that are relevant to the schema

				objects are the following: Attribute is required to be a member of the partial set = 0x00000002. Attribute is not replicated = 0x00000001. Attribute is a constructed attribute = 0x00000004.
searchFlags	Integer	No	No	The searchFlags property of each property's *attributeSchema* object defines whether a property is indexed. The four currently defined bits for this attribute are as follows: 1 = Index over attribute only; 2 = Index over container and attribute; 4 = Add this attribute to the Ambiguous Name Resolution (ANR) set (should be used in conjunction with 1); 8 = Preserve this attribute on logical deletion (that is, make this attribute available on tombstones).
isMemberof PartialAttributeSet	BOOL	No	No	A Boolean value that defines whether the attribute is replicated to the global catalog (if replicated to the global catalog, it has a value of TRUE, if not, its value is FALSE). For more information, see "Active Directory Replication" in this book.
SystemOnly	BOOL	No	No	System-only attributes are those attributes on which Windows 2000 and Active Directory depend for normal operations. If TRUE, only the system can modify this attribute. No user-defined attribute must ever have the *systemOnly* flag set.
objectClass	Object identifier	Yes	Yes	Class of this object, which is always *attributeSchema*.
nTSecurityDescriptor	NT-Sec-Des	Yes	No	Security descriptor on the *attributeSchema* object itself.
oMObjectClass	String(Octet)	No	No	For object-syntaxed attributes (OM-syntax = 127), the Basic Encoding Rules (BER) encoded object identifier of the XOM object class. For more information about BER encoding, see RFC 2251.
LinkID	Integer	No	No	Whether a linked attribute or not, an even integer denotes a forward link, an odd integer a back link. A forward link is a pointer to another object in the directory; a back link points back to the first object that has a forward link to it. (For more information about links, see "Active Directory Data Storage" in this book.)

1. Unicode is a 16-bit character set that contains all of the characters commonly used in information processing.
^{2}. When *rangeLower* and *rangeUpper* are defined for attributes that are integers, they define the limits of the value held by the attribute. When they are defined for attributes that are strings, they define the number of characters that can be held in the string.

classSchema Objects

The *classSchema* object specifies the various attributes of the class with which it is associated and, among other things, defines the following constraints of objects that are instances of the class:

- The list of mandatory attributes that must be present on any object that is an instance of this class.
- The list of optional attributes that, in addition to the *mustContain* attributes, can and might be found on an object that is an instance of this class.
- Hierarchy rules that determine the possible parents in the Directory Information Tree of an object that is an instance of the class.

An object can have only attributes that belong to either the *mustContain* or the *mayContain* list for the class.

The *classSchema* object is essentially a template that contains the "rules" for creating objects in an Active Directory class. When a new object is created in a class, the *classSchema* object ensures that this new object has the same properties ("attributes") as all other objects in the class. After an object has been created, the object's class can never be changed.

The *classSchema* object contains, among other things, the following information:

- The LDAP display name of the class.
- The object identifier for the class.
- The GUID for the class.
- The attributes that must be present for an instance of the class.
- Other attributes that can be present for an instance of the class.
- The classes to which the parent of instances of this class may belong.
- The superclass from which this class inherits characteristics.
- Other Auxiliary classes from which this class inherits attributes.
- The type of class (Abstract, Structural, or Auxiliary).
- The default hiding state for the class. If you do not want instances of your class displayed by the end-user user interface, you can define the class as hidden by default.

Categories of Object Classes

The X.500 1993 specification requires that object classes be assigned to one of four categories:

- Structural
- Abstract
- Auxiliary
- 88

Different categories of classes allow for defining structure in the directory. The four categories of classes are applied as follows:

Structural Classes Structural classes are the only classes that can have instances in the directory. That is, you can create directory objects whose class is one of the Structural classes. A Structural class can be used in defining the structure of the directory and is derived from either an Abstract class or another Structural class. A Structural class can include any number of Auxiliary classes in its definition. This type of class is specified by a value of 1 in the *objectClassCategory* attribute.

Abstract Classes Abstract classes are templates that are used only to derive new Structural classes. Abstract classes cannot be instantiated in the directory. This means that no object can belong only to an Abstract class; each object of an Abstract class also belongs to some nonabstract subclass of that class. A new Abstract class can be derived from an existing Abstract class. This type of class is specified by a value of 2 in the *objectClassCategory* attribute. Classes of the abstract category have the sole function of providing attributes for subordinate classes, called subclasses. A subclass contains all mandatory and optional attributes of the class from which it is derived, called its superclass, in addition to those specific to the class itself. Likewise, the subclass of that class contains all attributes of both superclasses, and so forth.

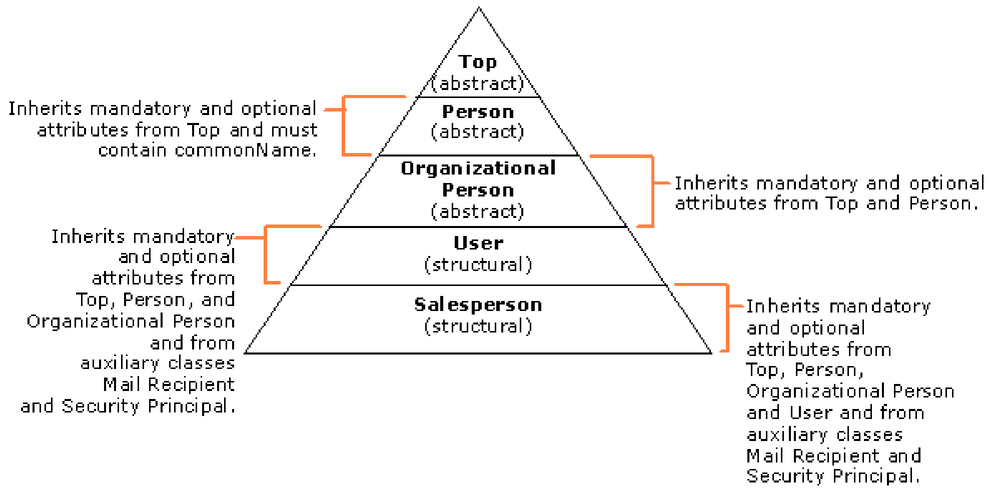
Auxiliary Classes Auxiliary classes are like "include" files; they contain a list of attributes. Adding the Auxiliary class to the definition of a Structural or Abstract class adds the Auxiliary class's attributes to the definition. An Auxiliary class cannot be instantiated in the directory, but new Auxiliary classes can be derived from existing Auxiliary classes. This type of class is specified by a value of 3 in the *objectClassCategory* attribute. For example, the *securityPrincipal* class is an Auxiliary class, and it derives its attributes from the parent abstract class called *top*. Although you cannot create a security principal object in the directory (because Auxiliary classes cannot have instances), you can create an object of the Structural class *user*, which has the *securityPrincipal* class as an auxiliary. The attributes of the *securityPrincipal* class contribute to making the user object recognizable to the system as a security account. Similarly, the *group* class has *securityPrincipal* as an Auxiliary class.

88 Classes Classes defined before 1993 are not required to fall into one of the preceding categories; assigning classes to categories was not required in the X.500 1988 specification. Classes that were defined prior to the X.500 1993 standards, default to the 88 class. This type of class is specified by a value of 0 in the *objectClassCategory* attribute. Do not define new 88 classes.

Note Active Directory does not return an error for 88 classes; it only performs looser semantic checking. For example, an 88 class can be used as an abstract superclass, but it can also be directly instantiated. When you define new schema classes, you need to use one of the X.500 1993 categories.

Inheritance

Inheritance, which is also referred to as derivation, is the ability to build new object classes from existing object classes. The new object is defined as a *subclass* of the parent object. A subclass is a class that inherits from some other class; for example, a subclass inherits structure and content rules from the parent. The parent object becomes a *superclass* of the new object. A superclass is a class from which one or more other classes inherit information. The inherited information includes mandatory and optional attributes (*systemMustContain*, *mustContain*, *systemMayContain*, and *mayContain*) and its parent classes in the directory hierarchy (*systemPossSuperiors* and *possSuperiors*). The diagram in Figure 4.2 illustrates an object class hierarchy.


Top (abstract)
Person (abstract)
Organizational Person (abstract)
User (structural)
Salesperson (structural)
Inherits mandatory and optional attributes from Top and must contain commonName.
Inherits mandatory and optional attributes from Top and Person.
Inherits mandatory and optional attributes from Top, Person, and Organizational Person and from auxiliary classes Mail Recipient and Security Principal.
Inherits mandatory and optional attributes from Top, Person, Organizational Person and User and from auxiliary classes Mail Recipient and Security Principal.

If your browser does not support inline frames, click here to view on a separate page.

Figure 4.2 Object Class Hierarchy

For example, you can specify a *Salesperson* class that defines information about the salespeople in your company, including specialized information, such as commission rate and travel route. You can specify the *Salesperson* class as *subClassOf* of the *User* class. This would cause the *Salesperson* class to inherit all the mandatory and optional attributes and directory-parent classes of the *User* class after the schema cache is updated. You would not have to define these attributes for each salesperson in your company.

All structural object classes are subclasses, directly or indirectly, of a single abstract object class, which is called *top*. Every object represented in the directory belongs to *top* and, as a result, every entry must have an *objectClass* attribute. When you create a new class, you must specify the superclass: If you are not creating a subclass of an existing class, the new class is a subclass of *top*.

A new class can inherit mandatory and optional attributes from more than one existing class. However, any additional classes must be specified by the *auxiliaryClass* attribute.

Note If you add another attribute later to a class that has subclasses or auxiliary subclasses, the new attribute is automatically added to the subclasses after the schema cache has been updated.

To view a graphical representation of the Active Directory class hierarchy, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

System and Changeable Attribute Pairs

Some aspects of a class-definition object are contained in pairs of attributes, where the value of one of these attributes can be changed by administrators and the other cannot. These attribute pairs are *mustContain/systemMustContain*, *mayContain/systemMayContain*, *possSuperiors/systemPossSuperiors*, and *auxiliaryClass/systemAuxiliaryClass*.

In each of these pairs, the value of the attribute that begins with the word *system* cannot be changed by administrators. This enables Active Directory to protect certain key attributes of certain classes and ensure that the schema stays consistent and usable. System-only properties can only be changed by the directory system agent (DSA). *System-only properties* are those properties on which Windows 2000 and Active Directory depend for normal operations. For example, the *attributeID* and *governsID* attributes in the schema are system-only attributes. The value of the other (nonsystem) attributes in each pair can be changed by administrators.

Mandatory Attributes

The term *mandatory attributes* refers to object attributes for which values must be specified. If you do not specify a value for a mandatory attribute, one of the following happens:

- The attribute takes on a default value.
- The object is not created until you specify a value for the attribute.

Which of the object's attributes are mandatory is determined by the class to which the object belongs.

Some mandatory attributes are inherited. Because every *schemaClass* object belongs to a subclass called *top* in the class hierarchy, each *schemaClass* object inherits the mandatory attributes that belong to *top*. Table 4.2 is a list of the mandatory attributes that every object inherits from *top*. To see a graphical representation of the Active Directory class hierarchy, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

Table 4.2 Mandatory Attributes That All *schemaClass* Objects Inherit

Inherited Mandatory Attribute	Default Status
nTSecurityDescriptor	Defaults if not specified. The default value depends on the default security descriptor for the classSchema class.
objectCategory	Defaults to the value of the default object category of the class (which is usually the class itself). Can be changed after the class is created.
objectClass	No default. Administrator must specify the class.

Note You can view an object's mandatory attributes by using the Active Directory Schema snap-in. (The attributes are displayed on the **Attributes** tab in the **Properties** dialog box.) Because some of an object's mandatory attributes are inherited from its parent class, you might need to view the attributes of the parent class in order to identify all of the mandatory attributes of your object. The Active Directory Schema snap-in is an MMC tool that is provided by Windows 2000 to enable administrators to modify the schema by using a graphical interface.

Attributes for classSchema Objects

Table 4.3 is a list of the attributes a *classSchema* object can have.

Table 4.3 Attributes of a *classSchema* Object

Attribute	Syntax	Mandatory?	Multi-value?	Description
cn	Unicode	Yes	No	Descriptive relative distinguished name for the schema object.
GovernsID	Object identifier	Yes	No	Object identifier that uniquely identifies this class.
LDAPDisplayName	Unicode	Yes	No	Name by which LDAP clients identify this class.
SchemaIDGUID	String(Octet)	Yes, but defaulted.	No	GUID that uniquely identifies this class.
RDNAttID	Object identifier	No	No	Relative-distinguished-name-type of instances of this class (OU, CN).
SubClassOf	Object identifier	Yes	No1	The class from which this object inherits attributes.
SystemMustContain	Object identifier	No	Yes	The list of mandatory attributes for instances of this class. This list cannot be changed.
MustContain[2]	Object identifier	No	Yes	The mandatory attributes for instances of this class.
SystemMayContain	Object identifier	No	Yes	The optional attributes for instances of this class.
MayContain[2]	Object identifier	No	Yes	The optional attributes for instances of this class.
SystemPossSuperiors[2]	Object identifier	No	Yes	The classes that can be parents of this class in the directory hierarchy. After creation of the class, this property cannot be changed.
PossSuperiors[2]	Object identifier	No	Yes	The classes that can be parents of this class in the directory hierarchy. For an existing *classSchema* object, values can be added to this property but not removed.
systemAuxiliaryClass[2]	Object identifier	No	Yes	The Auxiliary classes from which this class inherits its optional (*mayContain*) and mandatory (*mustContain*) attributes. After creation of the class, this property cannot be changed.
AuxiliaryClass[2]	Object identifier	No	Yes	The Auxiliary classes from which this class inherits its optional (*mayContain*) and mandatory (*mustContain*) attributes. A multivalue property that specifies the auxiliary classes that this class inherits from. For an existing *classSchema* object, values can be added to this property but not removed.

DefaultHidingValue	BOOL	No	No	The default hiding state for the class. If you do not want instances of your class displayed in the user interface, you can define the class as hidden.
DefaultSecurity Descriptor	String(Octet)	No	No	The default security descriptor that is assigned to new instances of this class if no security descriptor is specified during creation of the class or is merged into a security descriptor if one is specified.
ObjectClassCategory	Integer	Yes	No	Class types are defined as follows: 88 Class = 0; Structural = 1; Abstract = 2; Auxiliary = 3.
SystemOnly	BOOL	No	No	If TRUE, only the system can create and modify instances of this class.
ObjectClass	Object Identifier	Yes	Yes	This object's class, which is always *classSchema*.
NTSecurityDescriptor	NT-Sec-Desc	Yes	No	Security descriptor on the classSchema object.
DefaultObjectCategory	Distinguished name	Yes	No	The default object category of new instances of this class if none has been specified.

[1] Objects cannot inherit from more than one class by using this attribute. Use the *auxiliaryClass* attribute to define additional parent classes.
[2] Each value is the lDAPDisplayName of a class that is a class object identifier. Note that you must ensure that the classes exist or will exist when the new class is written to the directory. If one of the classes does not exist, the *classSchema* object is not added to the directory.

Note When you look at the attributes in a *classSchema* object's *mustContain* attribute list, you are not seeing the complete set of attributes that must be present for an instance of a class to exist. For example, in the class A, the *classSchema* object B specifies a list of *mustContain* attributes that an instance of A must have through the *systemMustContain* and *mustContain* attributes. However, because mandatory attributes are also inherited, the complete list of attributes for an instance of class A includes the inherited *mustContain* attributes from all classes from which B inherits — that is, all classes in the *subClassOf* and *auxiliaryClass* lists for the *classSchema* object B. The *mayContain* attributes for object B are also defined this way. The *possSuperiors* are defined this way as well, except that *possSuperiors* are inherited only from classes in the *subClassOf* list, not from the classes in the *auxiliaryClass* list.

Syntaxes

The *syntax* for an attribute defines the storage representation, byte ordering, and matching rules for comparisons of property types. Whether the attribute value must be a string, a number, or a unit of time is also defined. Every attribute of every object is associated with exactly one syntax. The syntaxes are not represented as objects in the schema, but they are programmed to be understood by Active Directory. The allowable syntaxes in Active Directory are predefined. You cannot add new syntaxes.

When you define a new attribute, you must specify both the *attributeSyntax* and the *oMSyntax* numbers of the syntax you want for the attribute. The *attributeSyntax* number is an object identifier and *oMSyntax* number is an integer. The *oMSyntax* is defined by the XOM specification. This model provides a relatively fine-grained definition of syntax. For example, there are distinct *oMSyntax* attributes to distinguish among several types of printable strings, according to factors such as the supported character set and whether case is significant. Table 4.4 is a list of the valid syntaxes for attributes in the Active Directory schema.

Table 4.4 Valid Syntaxes for Attributes in the Active Directory Schema

Syntax[1]	***attribute*Syntax**	**oM Syntax**	**ASN 1-Encoded Object Identifier**	**Description**
Undefined	2.5.5.0		\x550500	Not a legal syntax.
Object(DN-DN)	2.5.5.1	127	\x550501	The fully qualified name of an object in the directory.
String(Object-Identifier)	2.5.5.2	6	\x550502	The object identifier.
Case-Sensitive String	2.5.5.3	27	\x550503	General String. Differentiates uppercase and lowercase.
CaseIgnoreString(Teletex)	2.5.5.4	20	\x550504	Teletex. Does not differentiate uppercase and lowercase.
String(Printable), String (IA5)	2.5.5.5	19, 22	\x550505	Printable string or IA5-String. Both character sets are case-sensitive.
String(Numeric)	2.5.5.6	18	\x550506	A sequence of digits.
Object(DN-Binary)	2.5.5.7	127	\x550507	A distinguished name plus a binary large object.
Boolean	2.5.5.8	1	\x550508	TRUE or FALSE values.
Integer, Enumeration	2.5.5.9	2, 10	\x550509	A 32-bit number or enumeration.
String(Octet)	2.5.5.10	4	\x55050A	A string of bytes.
String(UTC-Time), String (Generalized-Time)	2.5.5.11	23, 24	\x55050B	UTC Time or Generalized-Time.
String(Unicode)	2.5.5.12	64	\x55050C	Unicode string.
Object(Presentation-Address)	2.5.5.13	127	\x55050D	Presentation address.
Object(DN-String)	2.5.5.14	127	\x55050E	A DN-String plus a Unicode string.
String(NT-Sec-Desc)	2.5.5.15	66	\x55050F	A Microsoft® Windows NT® Security descriptor.

LargeInteger	2.5.5.16	65	\x550510	A 64-bit number.
String(Sid)	2.5.5.17	4	\x550511	Security identifier (SID).

[1]The *oMSyntax* names are specified against the syntax numbers to enable correct choice.

Note A complete syntax specification consists of both the attribute-syntax and the *oMSyntax*. Whenever more than one *oMSyntax* can be used with an attribute-syntax, the correct *oMSyntax* must be used.

Active Directory does not currently enforce character set restrictions for string syntaxes, so if you use attributes with string syntax, use only characters in the standard character set.

Object Identifiers

Object identifiers are unique numeric values that are granted by various issuing authorities to identify data elements, syntaxes, and other parts of distributed applications. Because they are globally unique, object identifiers ensure that the objects that are defined by these issuing authorities do not conflict with one another when different directories, such as Active Directory and Novell Directory Services, are brought together in a global directory namespace.

Object identifiers are found in Open Systems Interconnection (OSI) applications, X.500 directories, Simple Network Management Protocol (SNMP), and other applications in which uniqueness is important. Object identifiers are based on a tree structure in which a superior issuing authority allocates a branch of the tree to a subordinate authority, which in turn allocates sub-branches of the tree.

LDAP requires a directory service, like Active Directory, to identify object classes and attributes with an object identifier syntax. The object identifier is the value for the *governsID* attribute in a class-schema object and for the *attributeID* attribute in an *attributeSchema* object. These are required attributes; therefore, object identifiers are necessary when you create new classes or attributes.

Object identifiers in the Active Directory base schema include some issued by the International Standards Organization (ISO) for X.500 classes and attributes and some issued by Microsoft. Object identifier notation is a dotted string of non-negative numbers (for example, 1.2.840.113556.1.5.4), the components of which are shown in Table 4.5.

Table 4.5 Components of a Sample Object Identifier (1.2.840.113556.1.5.4)

Numerical Values of the Sample Object Identifier	What the Numerical Values Denote
1	ISO ("root" authority) Issued 1.2 to ANSI, which in turn . . .
2	ANSI Issued 1.2.840 to USA, which in turn . . .
840	USA Issued 1.2.840.113556 to Microsoft, which . . .
113556	Microsoft Internally manages several object identifier branches under 1.2.840.113556 that include . . . .
1	Active Directory A branch called Active Directory that includes . .
5	Classes A branch called Classes that includes . . . .
4	Builtin-Domain A class called Builtin-Domain.

Object identifiers ensure that every object is interpreted appropriately — for example, that a telephone number is not mistaken for an employee number. A series of widely used objects and attributes is standardized for use in object identifiers. New object identifiers are issued by standards authorities, and they form a hierarchy below which new object identifiers can be managed internally. An object identifier is represented as a dotted decimal string (for example, 1.2.3.4). Enterprises (and individuals) can obtain a root object identifier from an issuing authority and use it to allocate additional object identifiers internally. For example, Microsoft Corporation has been issued the root object identifier 1.2.840.113556. Microsoft manages further branches from this root internally. One of these branches is used to allocate object identifiers for Active Directory classes, another for Active Directory attributes, and so forth.

Most countries and regions in the world have an identified National Registration Authority (NRA) responsible for issuing object identifiers to enterprises. In the United States, the NRA is the American National Standards Institute (ANSI). The NRA issues root object identifiers. An enterprise can register a name for the object identifier as well. There is a fee associated with registering the root object identifiers and registered names. Contact the NRA for your country or region for details. The International Standards Organization (ISO) recognizes NRAs and maintains a list of contacts on their Web site.

The issuing authority assigns an object identifier space that is a branch of the ISO-International Telecommunications Union (ITU) object identifier tree. Assume that your company is assigned the space 1.2.840.111111. You can extend this space internally as you want (within the constraints of the structure of an object identifier). For example, you can subdivide this space further (by appending dotted decimals to the object identifier root) and assign these subspaces to various divisions within your company. Each division, in turn, can further subdivide the subspace allotted to it. For example, by using the sample object identifier 1.2.840.111111, your company might have the subspace 1.2.840.111111.1.4 for attributes and 1.2.840.111111.1.5 for classes. An internal issuing authority within the company, using an Administrator account, might then allocate object identifiers from this space on request. The *governsID* attribute on every *classSchema* object and the *attributeID* attribute on every *attributeSchema* object are mandatory attributes that contain an object identifier string. In this example, all of your company-created *classSchema* objects have a *governsID* of the form 1.2.840.111111.1.5.*x*, where *x* is a decimal number. Similarly, all of your company-created *attributeSchema* objects have an *attributeID* of the form 1.2.840.111111.1.4.*x*.

Structure and Content Rules

The schema enforces rules that govern both the structure and the content of Active Directory. When you add, delete, or modify objects, validation takes place by using these schema rules to ensure the integrity of the directory. Structure rules define the possible tree structures. When you create a new object, structure rules determine the validity of the object class to which you designate the new object. You cannot create an object that belongs to a nonexistent class. You must first create the new class. Conversely, these rules do not allow you to delete or modify an object that has already been deleted. In Active Directory, the structure rules are completely expressed by the *possSuperiors* and *systemPossSuperiors* attributes that are present on each *classSchema* object. These attributes specify the possible classes that can be parents of an object instance of the class in question. In other words, the *possSuperiors* and *systemPossSuperiors* attribute values determine the object classes and, hence, the location in the Directory Information Tree under which objects of the class in question can be instantiated.

Content rules determine the mandatory and optional attributes of the class instances that are stored in the directory. New objects must contain all of the mandatory attributes that are specified by the *classSchema* object in the schema and can contain any of the optional attributes. In Active Directory, the content rules are completely expressed by the *mustHave, mayHave, mayContain*, *systemMustContain,* and *systemMayContain* attributes of the schema definitions for each class. In addition, specific marked attributes have additional restrictions imposed by the Security Account Manager (SAM). SAM read-only objects consist of the following:

revision, objectSID, domainReplica, creationTime modifiedCount, modifiedCountAtLastPromotion, nextRID, serverState, samAccountType, isCriticalSystemObject, dbcsPwd, ntPwdHistory,lmPwdHistory, lastLogon, lastLogoff, badPasswordTime, badPwdCount ,logonCount, supplementalCredentials

Below are some other attributes on which SAM enforces special checks:

sAMAccountName. Domain-wide uniqueness, without replication latency, 20-character limit for user objects (not groups).

Member. Membership rules as defined in Windows 2000 groups.

userWorkstations. Must be valid computer names.

primaryGroupID. For a user/computer account, must point to a group and the user /computer account must be a member of the group; the group and the user must be in the same domain. If the computer is a domain controller, the primary group must be the domain controllers group.

LockoutTime. For a user or computer object. Only legal value that can be written is 0 to clear an account.

LockoutPasswordLastset. The system normally writes to it, but two special values can be written 0 and -1 to expire /unexpire a password.

For more information about these attributes, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

Schema Cache

All changes made to Active Directory are validated first against the schema. For performance reasons, this validation takes place against a version of the schema that is held in memory on the domain controllers. This "in-memory version," called the *schema cache*, is updated automatically after the on-disk version has been updated. The schema cache provides mapping between attribute identifiers such as a database column identifier or a MAPI identifier and the in-memory structures that describe those attributes. The schema cache also provides lookups for class identifiers to get in-memory structures describing those classes.

When the computer is started, the schema cache is loaded from the underlying database and updated automatically whenever the on-disk version is updated. When changes are made to the schema, the schema cache is automatically updated within five minutes after the first change was applied. During the interval before the schema updates are copied to the schema cache, objects that reference a new or modified class or attribute cannot be added. This behavior keeps the cache consistent, but it can be confusing because changes are not apparent until the cache is updated, even though they were applied on disk.

There is also a mechanism for updating the schema cache on demand. You can use this when you modify the schema. You can add the *schemaUpdateNow* attribute to the rootDSE with a value of 1. The value is not used; it acts as a trigger or operational attribute. Writing this attribute starts a cache reload.

The rootDSE is a DSA-specific entry that holds the attributes that pertain to the local domain controller, such as directory partitions, server name, and supported LDAP version numbers. The *schemaUpdateNow* attribute is defined as an *operational attribute,* used only for administering the directory server itself. It is an artifact attribute that is never defined in the schema and does not require any storage. Generally, when you set an operational attribute, you trigger some action on the server.

Adding the *schemaUpdateNow* attribute causes a schema cache update to start immediately. The call is blocking, which means that if the call returns with no error, the cache is updated and all schema updates are ready to be used. An error return, however, indicates that the cache update is not successful. It is recommended that applications that want to take advantage of this feature be designed to accommodate the blocking write, particularly in giving the user feedback, if the program or script runs interactively.

Important It is recommended that you force an immediate schema cache update only once and only after all required schema updates are finished because cache loads are expensive in terms of memory.

Default Security of Active Directory Objects

The default security descriptor for an Active Directory object is specified in the schema. Essentially there are two segments to the default Active Directory security configuration or default access rights granted.

- Initial security for all objects created while installing Active Directory.
- Default security for objects created after installing Active Directory.

For information about the default security descriptors for Active Directory objects, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources . For information about permissions and security descriptors, see "Access Control" in this book.

Note There are special cases where default security is not applied on newly created objects. For more information about these situations, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

Default Security of the Domain Directory Partition

The domain directory partition object is derived from the object class *domainDNS*; therefore, the default security is equivalent to the default security for the object class *domainDNS.*

The default security descriptor for the domain directory partition comprises the following:

- Full control permissions to the Domain Administrators group and the System group, and Read permissions to the Authenticated Users group.
- Read property on all properties to the Everyone group. This permission provides backward compatibility for application programming interfaces (APIs).
- Replicating Directory Changes, Replication Synchronize, and Manage Replication Topology permissions to the Enterprise Domain Controllers group. These permissions allow members of the Enterprise Domain Controllers group to manage replication automatically.
- Replicating Directory Changes, Replication Synchronize, and Manage Replication Topology permissions to the Builtin Administrators group. Administrators of individual domain controllers can use these permissions to troubleshoot replication problems.
- Inheritable Full Control to the Enterprise Administrators group. Enterprise Administrators, by definition, have complete control of each domain.
- Inheritable List Contents to the Pre-Windows 2000 Compatible Access group.
- Inheritable Read Property on RAS Information, General Information, Membership, User Account Restrictions, and User Logon on all User Objects permissions to the Pre-Windows 2000 Compatible Access group.
- Inheritable Read on all Group objects.
- Inheritable Auditing successful/failed Writes to the Everyone group.

Activating the auditing policy ensures that writes that are performed on the directory (on any object) are audited immediately without the need for any extra user intervention. Inheritable access control entry (ACE) provides a convenient way of removing auditing policy.

Default Security of the Configuration Directory Partition

The default security descriptor for the configuration directory partition comprises the following:

- Full control permissions to Domain Administrators, and System and Read permissions to the Authenticated Users.
- Replicating Directory Changes, Replication Synchronize, and Manage Replication Topology permissions to the Enterprise Domain Controllers group. These permissions enable domain controllers in the forest to replicate from each other and automatically reconfigure the replication topology on the basis of replication delays and latency for the configuration directory partition.
- Replicating Directory Changes, Replication Synchronize, and Manage Replication Topology permissions to the Builtin Administrators group. These permissions enable administrators from individual domain controllers to synchronize replication and topology management for the configuration directory partition.
- Enable Inheritable Full Control to the Enterprise Administrators group. This permission allows members of the Enterprise Administrators group exclusive control over the Configuration container. The Enable Inheritable Full Control permission is required to control the Configuration container throughout the forest.
- Enable Inheritable Auditing to the Writes by the Everyone group. Activating the auditing policy ensures that writes that are performed on the directory (on any object) are audited immediately without the need for any extra user intervention. Inheritable ACE provides a convenient way of removing auditing policy.

Default Security of the Schema Directory Partition

The default security descriptor for the schema directory partition comprises the following:

- Write property permission on the *fSMORoleOwner* attribute to the Schema Administrators group. This permission enables members of the Schema Administrators group to forcibly transfer the domain controller where schema changes are made.
- Change Schema Master control permission to the Schema Administrators group. This permission enables members of the Schema Administrators group to change (per the Flexible Single-Master Operation [FSMO] protocol) the domain controller where schema changes are made.
- Inheritable Full Control permission designated to the Schema Administrators group. By default, the Schema Administrators group is the only group that has write access to the entire schema container. A schema object does not have any exclusive control over its own security, thus the object inherits its security from the schema container.
- Replicating Directory Changes, Replication Synchronize, and Manage Replication Topology to the Enterprise Domain Controllers group. These permissions enable the members of the Enterprise Domain Controllers group to manage replication of the schema in the forest automatically.
- Replicating Directory Changes, Replication Synchronize, and Manage Replication Topology permissions to the Builtin Administrators group. These permissions enable the administrators per domain controllers to resolve replication issues.
- Read permissions designated to the Authenticated Users group. This permission enables the members of the Authenticated Users group the right to read the schema.
- Audit successful/failed Writes by the Everyone group. Activating the auditing policy ensures that writes that are performed on the directory (on any object) are audited immediately without the need for any extra user intervention. Inheritable ACE provides a convenient way of removing auditing policy.

Default Security of Attributes and Classes

All attributes and classes inherit security from the ACLs on the Schema container. This ensures that the entire schema is consistent in terms of security.

Note The initial security allows only Schema Administrators write access to the Schema container

Extending the Schema

When the existing class and attribute definitions in the schema do not meet the needs of your organization, the schema can be extended by adding or modifying schema objects. The Active Directory schema can be extended 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 that are stored in the directory. This allows you to make the objects that are meaningful to your organization available throughout the enterprise.

Note As is true for every object in Active Directory, schema objects are protected by access control lists (ACLs), so only authorized users can alter the schema. (For more information about ACLs, see "Access Control" in this book.)

Adding or modifying class or attribute definitions in the schema involves adding or modifying the corresponding *classSchema* object or *attributeSchema* object. The operations that are involved in this process are similar to adding or modifying any object in Active Directory, except that additional checks are performed to ensure that changes do not cause inconsistencies or problems in the schema in the future.

When to Extend the Schema

Modifying the schema is a major change, with implications throughout the directory. It is recommended that you modify the schema only when it is absolutely necessary. Many schema modifications cannot be reversed, so you must make sure that changes are planned and well thought out before they are implemented. Inconsistencies in the schema can cause significant problems that impair or disable Active Directory. These problems might or might not be evident immediately.

Planning for schema modification involves examining the default schema that comes with Active Directory to verify that there is no way to use the existing classes or attributes for your needs. It is then necessary to understand the types of modifications that can be made and, conversely, that cannot be changed. The following are the modifications that can be made to the schema:

- Creating classes.
- Modifying existing classes.
- Creating attributes.
- Modifying existing attributes.
- Deactivating classes and attributes.

There are three ways to effectively add a new class:

- Extending an existing class by adding attributes or additional possible parents.
- Deriving a new subclass from an existing class. The subclass has all the attributes of the original class and any additional attributes that you specify.
- Creating an entirely new class with any attributes that you want to assign.

You need to extend an existing class when the following conditions apply:

- The existing class needs additional attributes but otherwise meets your needs. For example, you might want to add a *purchasingLimit* attribute to the *User* class and add it to the user object for people who are cost center managers and have purchasing authority.
- You have no need to identify the extended class as a distinct class from the original class.
- You want to use the existing Active Directory Users and Computers console in MMC to manage the extended attributes of the objects. This requires the addition of property pages to the set defined for the object you are extending.

Derive a subclass from an existing class when the following conditions apply:

- The existing class needs additional attributes but otherwise meets your needs.
- You want to identify the extended class as a distinct class from the original class.
- You want to use the existing Active Directory Users and Computers console in MMC to manage the extended attributes of the objects.

How to Extend the Schema

After you have decided that you have to make changes to the schema and you have carefully planned the types of changes you are going to make, you can proceed. Because this is an extremely significant operation, and not without the possibility of causing serious problems, Windows 2000 has three safety features, or interlocks, that control modification of the schema:

- By default, schema modification is disabled on all domain controllers. Use the Active Directory Schema console on a domain controller to permit write access to the schema on that domain controller.
- The schema object is protected by the Windows 2000 security model. Therefore, administrators must be given explicit permissions or be a member of the Schema Administrators group (**Schema Admins** in the user interface) to effect changes to the schema.
- Only one domain controller in the enterprise, the one holding the Schema Master Role, is allowed to write to the schema. This role is one example of an FSMO role.

Installation of Schema Extensions

The recommended practice is to strictly control the schema updates at most customer sites. If a service requires schema extensions, you must be able to install them separately by using one of the following methods

- Extending the schema by using LDIF scripts. This allows customers to update the schema separately and in advance of the rest of the installation.
- Extending the schema programmatically.

In addition to providing a separate installation procedure for schema extensions, it is recommended that the nature of the schema extensions be clearly documented. The documentation needs to contain the following:

- A statement that describes the authority from which your object identifier prefix was obtained.
- The common-name (cn), the LDAP-Display-Name, the object identifier, and the description of each new class and attribute and its expected use. Also answer the following questions:
  - Is the attribute configured for replication to Global Catalog servers?
  - Is the attribute configured for indexing?
  - What are the expected update frequency and expected size of the attribute, which allows the customer to make calculations of replication impact?
  - What are the *rangeLower* and *rangeUpper* values?
- A class hierarchy showing newly created classes.
- If defined, the values for the Default-Security-Descriptor and the NT-Security-Descriptor.

The schema installation program must allow the user to exit the program prior to your making any changes to the schema.

Specify the Schema-ID-GUID

Specify the schemaIDGUID when you create attributes or classes in Active Directory. The schemaIDGUID is a globally unique identifier (GUID) that uniquely identifies all classes and attributes in the schema. Unlike object identifiers, which are issued by a central authority, a special algorithm generates GUIDs. SchemaIDGUIDs are used in ACLs to provide attribute-specific or class-specific privileges.

Naming

When you modify the schema, you must adhere to the following rules with respect to specifying the relative distinguished name attribute (which is common-name [cn]) and the LDAP display name (lDAPDisplayName).

Common-Name (cn)

- Choose a company prefix. This section of the prefix must be the registered DNS domain name of the company and the current year (four digits, separated by a hyphen (-).
- Make the next token in the cn a hyphen (-).
- Choose a product-specific prefix. This section of the name must be unique within your company and a succinct identification of the product and needs to begin with an uppercase letter. The letters in the remainder of the prefix can be uppercase or lowercase as you deem appropriate.
- Make the next token in the cn a hyphen (-).
- Make the next section of the cn the name of the attribute or class separated by hyphens.

LDAP-Display-Name

- Use the Common-Name (cn) as the starting point for the LDAP-Display-Name (lDAPDisplayName).
- Make the first character LDAP-Display-Name lowercase.
- Make the character that immediately follows each hyphen (-) uppercase.
- Remove all hyphens that follow the product-specific section of the prefix except for the hyphen that immediately follows this section.

Table 4.6 illustrates the naming rules as they are applied to the Common-Name (cn) and the LDAP-Display-Name (lDAPDisplayName):

Table 4.6 Naming Rules

Common-Name (cn)	LDAP-Display-Name (lDAPDisplayName)
Microsoft-Com-1999-MQ-Attribute-1	Microsoft-Com-1999-mQAttribute1
Microsoft-Com-1999-EXCHANGE-Attribute-2	Microsoft-Com-1999-exchangeAttribute2

Modifying the Schema

To allow a domain controller to modify the schema, use the Active Directory Schema console in MMC on the selected server.

Note Because of the serious nature of schema modification, the Active Directory Schema MMC snap-in is not listed with the default MMC snap-ins that are provided with Windows 2000 Server. To make it appear in the list of available MMC snap-ins, you must run **Regsvr32** on the dynamic-link library (DLL) (Schmmgmt.dll) from the command prompt.

To enable schema modification

1. Open the Active Directory Schema console in MMC.
2. Right-click **Active Directory Schema (Manager)**, and select **Operations Master**.
3. Select **The Schema may be modified on this server** check box, and then click **OK**.

The value of the **The Schema may be modified on this server** check box is stored in the registry in the **Schema Update Allowed** entry (in HKEY_LOCAL_MACHINE \SYSTEM \CurrentControlSet \Services \NTDS \Parameters). Active Directory adds this entry to the registry when you use the Active Directory Schema console to change the default value.

Caution Do not use a registry editor to edit the registry directly unless you have no alternative. The registry editors bypass the standard safeguards provided by administrative tools. These safeguards prevent you from entering conflicting settings or settings that are likely to degrade performance or damage your system. Editing the registry directly can have serious, unexpected consequences that can prevent the system from starting and require that you reinstall Windows 2000. To configure or customize Windows 2000, use the programs in Control Panel or MMC whenever possible.

Schema Administrators Group

To modify the schema, you must use an account that is a member of the Schema Admins group. By default, the only member in that security group is the Administrator account in the root domain of the enterprise. If you want to add other accounts, you have to add them explicitly.

Caution Membership in the Schema Admins group must be highly restricted to prevent unauthorized access to the schema because modifying the schema improperly can have serious consequences.

One way to verify that an account is a member of the Schema Admins group is to use the Active Directory Users and Computers console in MMC.

To verify that an account is a member of Schema Administrators

1. Open the Active Directory Users and Computers console.
2. Expand the domain for the account by clicking the plus sign (+) next to it.
3. Double-click the **Users** folder.
4. Double-click the **Schema Admins** security group, and then click the **Members** tab.
5. If the account is not listed under **Members**, click **Add**.
6. Select an account from the displayed list, or type the name of the account.
7. Click **Add**, and then click **OK**.

Schema FSMO Role

Active Directory performs schema updates in a single-master fashion to prevent conflicts. Simultaneous schema updates on two different computers might conflict with each other. The one domain controller in the enterprise that is allowed to perform schema updates at any specific time is referred to as the *schema master*. Only one domain controller in the entire enterprise, the domain controller holding the schema master role, accepts updates to schema objects.

Note To update the schema, the domain controller holding the schema master role must be online.

You can change the domain controller that serves as the schema master at any time according to your needs. This is what is meant by the word "flexible" in FSMO. The current schema master in the enterprise is identified by the value of the *fSMORoleOwner* attribute on the Schema container of the domain. By default, the first domain controller that is installed in the enterprise is the initial schema master.

Although the domain controller that is the current FSMO Role Owner for schema operations is the only one that can make the actual schema modifications, you do not have to be connected to that domain controller when you make schema modifications. If you are connected to a domain controller that does not have that role, it generates a referral to the current FSMO Role Owner to process the modifications.

If you want to do so, you can transfer the role of schema master to another domain controller by using the Active Directory Schema console in MMC.

To view or change the current schema master by using the Active Directory Schema console in MMC

1. Open MMC, and install the Active Directory Schema snap-in.
2. Right-click the Active Directory schema, and then click **Operations Master**.
3. The Current Operations Master that is displayed is the schema master.

   To retain the current schema master, click **OK**.

   – Or –

   To change the server that is the current FSMO Role Owner for the schema, click **Change**.

   If the current domain controller (the one that is listed in **Current Focus**) is also the current operations master, you must use the Active Directory Tree console to focus on another domain controller before you can change the operations master. This is because you must be connected to the domain controller that you want to have as the FSMO Role Owner. You cannot direct the connected domain controller to make another domain controller the FSMO Role Owner.

For more information about using the Active Directory Schema console, see "Modifying the Schema" earlier in this chapter.

You can also use the command-line tool Ntdsutil to transfer the Schema FSMO. The tool resides in the \%SystemRoot%\System32 folder. For more information about transferring FSMO roles by using Ntdsutil, see "Managing Flexible Single-Master Operations" in this book.

To change the schema master by using Ntdsutil

1. Start Ntdsutil by typing **ntdsutil** at the command prompt. (Note that at any prompt in this tool, you can type a question mark (**?**) to see the list of valid commands for that prompt.)
2. At the Ntdsutil prompt, type:

   roles
3. At the **fsmo maintenance** prompt, type:

   connections
4. To display the current connection information, at the server connections prompt, type:

   info

   If necessary, type the appropriate command to connect to the server that is to become the schema master. (Use the **?** command to see a list of valid commands.)
5. To return to the **fsmo maintenance** prompt, type:

   quit
6. To do a graceful transfer of the Schema FSMO, type:

   transfer schema master

You can also perform the schema master role transfer in a program. Before a program can make changes to the schema, it must check explicitly whether the domain controller is the current schema master and, if it is not, explicitly request the transfer operation.

To understand the transfer process, consider a scenario in which computer A is the current FSMO Role Owner and computer B must perform some schema updates. To request an FSMO Role Owner transfer from computer A, a program must add the operational attribute *becomeSchemaMaster* with value of 1 to the rootDSE (that is, to the object with a blank distinguished name) on computer B. It is an operational attribute that is never defined in the schema and does not require any storage. Generally, when you set an operational attribute, you trigger some immediate action on the server.

In this case, the action taken by the server (computer B) is its sending out a request to computer A for a role transfer. Computer A, upon receiving such a request, changes the *fSMORoleOwner* attribute on its Schema container to the name of computer B and sends this new attribute value back to computer B. It also sends back any schema changes that were implemented on computer A but were not yet incorporated by computer B. (This kind of discrepancy is possible as a result of replication latencies.) Computer B, upon receiving the reply from computer A, applies all changes that were sent back from computer A and, in the process, becomes the current schema master.

Note Computer B, the new schema master, now has all previous schema updates in the enterprise and, hence, the latest version of the schema.

If the old schema master is unavailable or has crashed, you can forcibly transfer (*seize*) the schema FSMO so that a new domain controller can make schema changes. However, it is recommended that you take this step only as a last resort. When the schema master is forcibly transferred to a new domain controller, recent schema changes that were made at the old schema master might not be propagated to the new schema master and might be lost. The transfer also can result in conflicting updates at other domain controllers in the forest, which might require an extensive offline cleanup of the directory.

Caution Seizing the schema master is a drastic step that you must consider only when the current schema master is no longer able to function and is never going to be available again. Before you seize the current schema master, remove it from the network. Verify that the domain controller that seizes the role is fully up-to-date with respect to updates performed on the previous role owner.

To seize the schema master by using Ntdsutil

1. Start Ntdsutil by typing **ntdsutil** at the command prompt. (Note that at any prompt in this tool, you can type a question mark (**?)** to see the list of valid commands for that prompt.)
2. At the Ntdsutil prompt, type:

   roles
3. At the fsmo maintenance prompt, type:

   connections
4. To display the current connection information, at the server connections prompt, type:

   info

   If necessary, type the appropriate command to connect to the server that is to become the schema master. (Use the **?** command to see a list of valid commands.)
5. To return to the fsmo maintenance prompt, type:

   quit
6. To perform a forced transfer (*seizure*) of the schema master, type:

   seize schema master

For more information about FSMOs, see "Managing Flexible Single-Master Operations" in this book.

Order of Processing When Extending the Schema

If you decide to extend the schema either programmatically or by using scripts, apply updates in the following order:

1. Target your update at the FSMO Role Owner. Bind to the schema on the domain controller that is the schema master. Avoid unnecessarily changing the schema master role between domain controllers. Only one domain controller is allowed to perform critical operations like updating the schema at any one time. This domain controller is known as the FSMO Role Owner. If you have more than one Windows 2000 server on your network, your current server might not be the FSMO Role Owner. You have to ensure that you target your update at the FSMO Role Owner.

2. Ensure that you have sufficient administrative privileges to perform the schema update. Check the *allowedChildClassesEffective* property of the Schema container to see if you can create attributes 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. Only members of the Schema Administrators group are allowed to alter the contents of the schema. You must ensure that your user account is a member of this group. (The Administrator account is automatically a member of the Schema Administrators group.)
3. Create the registry entry that allows write access to the schema. By default, access to the schema is read-only. This entry, known as the safety interlock, can be found in the registry in HKEY_LOCAL_MACHINE \SYSTEM \CurrentControlSet \Services \NTDS \Parameters**Schema Update Allowed**. This entry must exist and its value must be nonzero for schema updates to take place. Check that the safety interlock is engaged before removing it. Note the value that you found for this entry, and make sure to leave the value in the same state afterward. Note that you only have to create the safety interlock on the server that holds the FSMO role.
4. Add your new attributes.
5. Add your new classes.
6. Add attributes to classes. Any new attributes need to be referenced by object identifier because their names are not going to be present in the cache yet. Unless you trigger a schema cache reload after you add new attributes, an attempt to use an attribute by name is going to fail.
7. Each domain controller updates its schema cache five minutes after a schema change. If the extensions are going to be used within five minutes, you must trigger a cache reload.
8. If you had to create the safety interlock before you added your new classes or attributes, it is recommended that you re-apply the safety interlock again after you add them.
9. If you are installing a schema extension by programmatic means (script or ADSI), you must make sure that the extension is provided as a separately installable routine. This means that you must be able to do it separately from the application installation process.
10. Before you create a program to perform a schema extension, see the Microsoft Platform SDK link on the Web resources page at http://windows.microsoft.com/windows2000/reskit/webresources . Follow the links to "Active Directory Programmer's Guide" and then to "Schema Extensibility."

    Note A cache update is not necessary if the schema extensions are not to be used immediately. Depending on system load, the extensions appear in the schema cache in approximately five minutes.

Adding and Modifying Schema Objects

Because schema objects are another kind of directory object, you can use the same methods that you would use to add or modify any directory object. Windows 2000 provides an administrative tool called Active Directory Schema that provides a straightforward user interface, and, of course, you have the option of making changes to the schema programmatically.

Adding an Attribute

It is recommended that you try to use existing attributes wherever possible. If you decide that you need to create a new attribute, however, you must adhere to the following guidelines:

- Use *cn* as the *name* (relative distinguished name) attribute; this is the default for most classes, including those derived directly from *top*. Because *cn* is an indexed attribute, it allows an efficient search for your object by name.
- Large multivalue attributes are costly to store and retrieve; it is recommended that you avoid using them. Active Directory implements an LDAP control to allow an incremental read of large multivalued attributes, but not all LDAP clients know how to use this control.
- Remember that attributes are "flat," which means that there is no implied substructure to an attribute. All attributes in a specific class must relate directly to instances of that class. This is also good data normalization practice.

To add a new attribute to the schema, you must create a new attribute object. First create the Active Directory safety interlocks as described in "How to Extend the Schema" earlier in this chapter. Then do the following:

1. Choose a name for the attribute.
2. Obtain a valid object identifier from an issuing authority.
3. Determine the syntax of the attribute.
4. Decide whether the attribute needs to be a single-value or multivalue attribute.
5. Decide whether and how the attribute needs to be indexed.
6. Decide whether the attribute needs to be replicated to the Global Catalog.

For every attribute that you define, some attributes are mandatory and some are optional; these attributes are listed in Table 4.7 and Table 4.8.

Table 4.7 Mandatory Attributes for New Attribute-Definition Objects

Mandatory *Attributes*	**Default Status**
cn	No default. Administrator must specify a name.
objectClass	No default. Administrator must specify as attributeSchema.
*attribute*ID	No default. Administrator must specify as an object identifier string.
*attribute*Syntax	No default. Administrator must specify one of the syntaxes that are recognized by Active Directory.
oMSyntax	No default. Administrator must specify an oMSyntax that matches the corresponding attribute syntax.
schemaIDGUID	It is defaulted to a value generated by uuidgen if not specified.
nTSecurityDescriptor	Defaults if the administrator does not specify. The default value depends on the defaultSecurityDescriptor attribute of the attributeSchema class.
isSingleValued	Defaults to FALSE if not specified by the administrator.
lDAPDisplayName	Defaults from the common name if not specified by the administrator.

Table 4.8 Optional Attributes for New Attribute-Definition Objects

Optional Attributes	**Default Status**

rangeLower	No default. The administrator must specify a value.
rangeUpper	No default. The administrator must specify a value.
isMemberOfPartialReplicaSet	Defaults to FALSE if not specified by the administrator.
searchFlags	No default. The four currently defined bits for this attribute are as follows: 1 = Index over attribute only; 2 = Index over container and attribute; 4 = Add this attribute to the Ambiguous Name Resolution (ANR) set (needs to be used in conjunction with 1); 8 = Preserve this attribute on logical deletion (that is, make this attribute available on tombstones).

As an example, suppose you want to add a new attribute called *userName*. Each instance of a *userName* attribute stores exactly one Unicode string of at least one character and not more than 1,000 characters. In this case, you would add the following attribute definition:

- *cn* = *userName*
- *objectClass* = *attributeSchema*
- *attributeID* = 1.2.567.8901234.5.6.879 (Valid object identifier value)
- *attributeSyntax* = 2.5.5.12 (Syntax value for Unicode string)
- *oMSyntax* = 64 (Syntax value for Unicode string)
- *isSingleValued* = TRUE (The intention is to store exactly one value.)
- *rangeLower* = 1 (Minimum length of the string)
- *rangeUpper* = 1000 (Maximum length of the string)

Modifying an Attribute

To modify an attribute, modify the existing attribute-definition object that represents the class. For reasons of consistency and security, some attributes of each attribute-definition object are designated as system-only. You cannot modify system-only attributes of an attribute object, not even for new classes that you originally created. System-only attributes are designated by having the *systemOnly* attribute of the attribute set to TRUE.

The following attributes of an attribute-definition object are *systemOnly* and, thus, cannot be modified:

- *attributeID*
- *schemaIDGUID*
- *attributeSyntax*
- *oMSyntax*
- *isSingleValued*
- *extendedCharsAllowed*
- *systemOnly*
- *objectClass*
- *instanceType*

Adding a Class

To add a new class, you add a new schema-definition object with all the desired attributes. After you remove the Active Directory safety interlocks, as described in "How to Extend the Schema" earlier in this chapter, make sure that you have done the following before you add a class:

1. Choose a name for the class.
2. Obtain a valid object identifier from an issuing authority.
3. Determine the object class category.
4. Determine the class from which this new class inherits information.

For every class, some attributes are mandatory and some are optional, as shown in Table 4.9 and Table 4.10. If you do not define values for some of these attributes, they are given default values.

Table 4.9 Mandatory Attributes for New Class-Definition Objects

Attribute	**Default Status**
cn	No default. Administrator must specify a name.
objectClassCategory	Defaults to 88 class because it is assumed to be a class with no category. Other options are Structural, Abstract, or Auxiliary.
governsID	No default. Administrator must specify an object identifier string.
possSuperiors	No default. Administrator must specify the structural class or classes that are legal parents of instances of this class.
subClassOf	No default. Administrator must specify a value.
schemaIDGUID	Defaults if not specified. The default value is automatically generated by the system.
nTSecurityDescriptor	Defaults if not specified. The default value depends on the *default SecurityDescriptor* of the *classSchema* class.
lDAPDisplayName	Defaults from the common name if not specified.

Table 4.10 Optional Attributes for New Class-Definition Objects

Optional	**Default Status**
defaultSecurityDescriptor	If there is no default security descriptor specified, the default security descriptor of the immediate superclass is used.
auxiliaryClass	The list of additional (auxiliary) classes from which this class is derived.

For a new class, you must define *cn*, *objectClass*, and *governsID*. However, to make the new class actually useful, you probably also want to define some attributes in *mustContain*, *mayContain*, and *possSuperiors*. Any attributes you specify when you add a new class must already exist. So, if you want to add a new class with new attributes, you must add the new attributes to the schema first.

When you add a new class, the object identifier specified in *governsID* must be unique, not only in your enterprise but also globally.

Note The system imposes rules that restrict the addition of schema objects.

Suppose you want to add a new class "*Friend*" to store information about a friend. Any *Friend* object must contain the name of the friend and might also contain her address or phone number. And because a friend is a person, you want objects of the *Friend* class to have the same mandatory attributes, optional attributes, and directory superiors as the *Person* class you have already defined. In this case, you add the following class definition:

- *cn = Friend*
- *objectClass = classSchema*
- *subClassOf = Person*
- *governsID* = 1.2.345.678901.2.3.45 (valid object identifier value)
- *mustContain* = givenName, sn
- *mayContain* = Address, phone-number

Modifying a Class

To modify a class, modify the existing class-definition object that represents the class. Some attributes of each class are designated as system-only, for consistency and security reasons. You cannot modify system-only attributes of a class-definition object, not even for new classes that you originally created. System-only attributes are designated by having the *systemOnly* attribute of the attribute set to TRUE.

The following attributes of a class-definition object are system-only attributes and, thus, cannot be modified:

- *governsID*
- *schemaIDGUID*
- *rDNAttID*
- *subClassOf*
- *systemMustContain*
- *systemMayContain*
- *systemPossSuperiors*
- *systemAuxiliaryClass*
- *objectClassCategory*
- *systemOnly*
- *objectClass*
- *instanceType*

System Checks and Restrictions Imposed on Schema Additions and Modifications

When you try to add or modify a class or attribute, Active Directory performs some checks to make sure that the changes do not cause inconsistencies or other problems in the schema. The checks can be divided into two classes:

- Consistency checks
- Safety checks

Consistency checks maintain the consistency of the schema. Safety checks reduce the possibility of a schema update by one application breaking another application.

Consistency Checks

For both class and attribute changes, the system makes sure that the values of *lDAPDisplayName* and *schemaIDGUID* are unique and also that *lDAPDisplayName* is valid.

The class-schema object addition and modification extensions are successful only if the new class definition passes all of the following tests as well as the normal extension checks.

- The value of *governsID* must be unique.
- All attributes that are defined in the *systemMayContain*, *mayContain*, *systemMustContain,* and *mustContain* lists must already exist.
- All classes that are defined in the *subClassOf*, *systemAuxiliaryClass*, *auxiliaryClass*, *systemPossSuperiors*, and *possSuperiors* lists must already exist.
- All classes in the s*ystemAuxiliaryClass* and *auxiliaryClass* lists must have either 88 class or Auxiliary class specified as their *objectClassCategory*.
- All classes in the *systemPossSuperiors* and *possSuperiors* lists must have either 88 class or Structural class specified as their *objectClassCategory*.
- Classes in the *subClassOf* list must follow certain X.500 specifications for inheritance hierarchies. These specifications are that Abstract classes can inherit only from other Abstract classes, Auxiliary classes cannot inherit from Structural classes, and Structural classes cannot inherit from Auxiliary classes.
- The attribute specified in the *rDNAttID* attribute must have Unicode-string as its syntax and be single-valued.

For attribute changes, the system also checks the following:

- The value of *attributeID* must be unique.
- The value of *mAPIID*, if any, must be unique.
- If *rangeLower* and *rangeUpper* are present, *rangeLowe*r must be smaller than *rangeUpper*.
- The values of *attributeSyntax* and *oMSyntax* must match, as shown in Table 4.11.
- If the attribute is object-syntaxed (*oMSyntax*=127), it must have the correct *oMObjectClass*, as shown in Table 4. 12.
- The *linkID*, if any, must be unique. In addition, a back link must have a corresponding forward link. (For more information about links, see "Active Directory Data Storage" in this book.)

Note A complete syntax specification consists of both the *attributeSyntax* and the *oMSyntax*. Hence, whenever more than one *oMSyntax* can be used with an *attributeSyntax*, the correct *oMSyntax* must be used.

Table 4.11 Values of *attributeSyntax* and Corresponding Syntaxes

***attributeSyntax* Value[1]**	**Matching *oMSyntax***
2.5.5.1	127 [Object(DN-Binary)]
2.5.5.2	6 [String(Object-Identifier)]
2.5.5.3	27 [String(Case sensitive)]
2.5.5.4	20 [String(Case insensitive)]
2.5.5.5	19 [String(Printable)], 22 [String(IA5)]
2.5.5.6	18 [String(Numeric)]
2.5.5.7	127 [Object(ORName)] or [Object(DNBinary)]. Distinction is oMObjectClass value.
2.5.5.8	1 [Boolean]
2.5.5.9	2 [Integer], 10 [Enumeration]
2.5.5.10	4 [String(Octet)]
2.5.5.11	23 [String(UTC-Time)], 24 [String(Generalized-Time)]
2.5.5.12	64 [String(Unicode)
2.5.5.13	127 [Object(Presentation-Address)]
2.5.5.14	127 [Object(Access-Point)] or [Object(DN-String)]. Distinction is oMObjectClass value
2.5.5.15	66 [String(NT-Sec-Desc)]
2.5.5.16	65 [LargeInteger)]
2.5.5.17	4 [String(Sid)]

[1]The *oMSyntax* names are specified with the syntax numbers to enable the correct choice.

For attributes with *oMSyntax*=127, the *oMObjectClass* also must be correctly specified according to the *attributeSyntax*. For attributes with any other *oMSyntax* value, it is not relevant and need not be specified. Because an *oMObjectClass*, being a binary value, is somewhat inconvenient to specify and because in most cases there is a one-to-one mapping between the *attributeSyntax* and *oMObjectClass*, the value defaults if none is specified by the user. There are a couple of cases where the mapping is not one-to-one, however, and the value defaults to the more common value. Table 4.12 is a list of the *oMObjectClass* values that correspond to the different *attributeSyntax* values for attributes with *oMSyntax*=127.

Table 4.12 Values of *attributeSyntax* and Corresponding *oMObjectClass* Values

***attribute*Syntax**	***oMObjectClass Values*[1]**
2.5.5.1	\x2B0C0287731C00854A [Object(DS-DN)].
2.5.5.7	\x56060102050B1D [Object(OR-Name)] or \x2A864886F7140101010B [Object(DN-Binary)].
	Defaults to Object(OR-Name) if none specified by the user.
2.5.5.13	\x2B0C0287731C00855C [Object(Presentation-Address)].
2.5.5.14	\x2B0C0287731C00853E [Object(Access-Point)] or \x2A864886F7140101010C [Object(DN-String)].
	Defaulted to Object(Access-Point) if none specified by the user.

[1]The syntax names are specified in brackets for easy reference.

Safety Checks

The purpose of the safety checks is to reduce the possibility of schema updates by one user or application breaking another application. These checks are necessary because multiple applications might share a schema definition.

When you modify existing schema objects, the modifications are subject to certain restrictions enforced by Active Directory. In some cases, these restrictions are determined according to whether the objects are part of the original schema or whether they have been added after the original installation. So the schema objects are really divided into two categories:

- Category 1 objects
- Category 2 objects

Category 1 objects are the default base schema objects that are included with Windows 2000 in the base schema. *Category 2 objects* are objects that are added subsequently to the schema by administrators or applications. You can determine the category in which an object is located by looking at the second bit (starting at the least significant bit) in the *systemFlags* attribute. If the bit is set, it has the value FLAG_SCHEMA_BASE_OBJECT, which indicates that the object is part of the base schema, that is, category 1. If this bit is not set or the attribute is not present, the object is category 2.

The following restrictions apply to both category 1 and category 2 schema objects:

- You cannot add a new *mustContain* attribute to a class either directly or through inheritance by adding an auxiliary class.
- You cannot add or delete any *mustContain* attribute of a class either directly or through inheritance.

The following restrictions apply to category 1 schema objects:

- You cannot change the *rangeLower* and *rangeUpper* of an attribute.
- You cannot change the *atributeSecurityGUID* of an attribute.
- You cannot deactivate a class or an attribute (make it defunct).

- You cannot change the *lDAPDisplayName* of a class or an attribute.
- You cannot rename a class or an attribute.
- You cannot change the *defaultObjectCategory* of a class.
- You cannot change the *objectClassCategory* of instances of a class.

Deactivating Schema Objects

You cannot deactivate schema objects that are part of the default schema that ships with Active Directory. You can only deactivate schema objects that have been added to the default schema.

You might want to delete schema classes or attributes that are not needed in your organization. However, deleting schema objects raises some serious issues. For instance, what would happen to any other schema objects that use the class or attribute that you have deleted? Because doing an enterprise-wide check and cleanup might prove very time-consuming and costly, Active Directory does not support the actual deletion of schema objects. Rather it provides a mechanism for deactivating schema objects, also referred to as making them defunct. When you deactivate a schema object, you make it unusable for most purposes, and you get most of the benefits of deletion.

A class or an attribute can be deactivated by setting the Boolean attribute *isDefunct* to TRUE on the schema object. At any point in time, there are a number of ways to identify the defunct schema objects in the system. Programmatically, the user can search for all schema objects that have the attribute *isDefunct* set to TRUE (or if a particular schema object has *isDefunct* set to TRUE, to check whether the object is defunct). You can also use the Search function of the Ldp tool to search the schema with a filter set to (*isDefunct*=TRUE). For more information about the Ldp tool, see "Active Directory Diagnostics, Troubleshooting, and Recovery" in this book.

Note There is currently no method in the user interface for viewing defunct schema objects. To do this, you can use only one of the methods described in the preceding paragraph.

As with additions or modifications of classes or attributes, there are some special validation checks performed when a class or an attribute is made defunct. This is to ensure the consistency of the schema. In particular, on an attempt to make a class defunct, Active Directory verifies that the class is not used in the *subClassOf*, *auxiliaryClass*, or *possSuperiors* list of any existing nondefunct class. Similarly, on an attempt to make an attribute defunct, Active Directory checks that the attribute is not used in the *mustContain* or *mayContain* of any existing nondefunct class.

A defunct schema object can be resurrected, that is, made nondefunct again, by either removing the *isDefunct* attribute from the object or by setting the value of the *isDefunct* attribute to FALSE. This can also be done easily by using the Active Directory Schema console. Because making a defunct schema object nondefunct is similar to adding a new schema object as far as subsequent schema updates go, Active Directory performs the same validation checks here as it does on the addition of a new schema object.

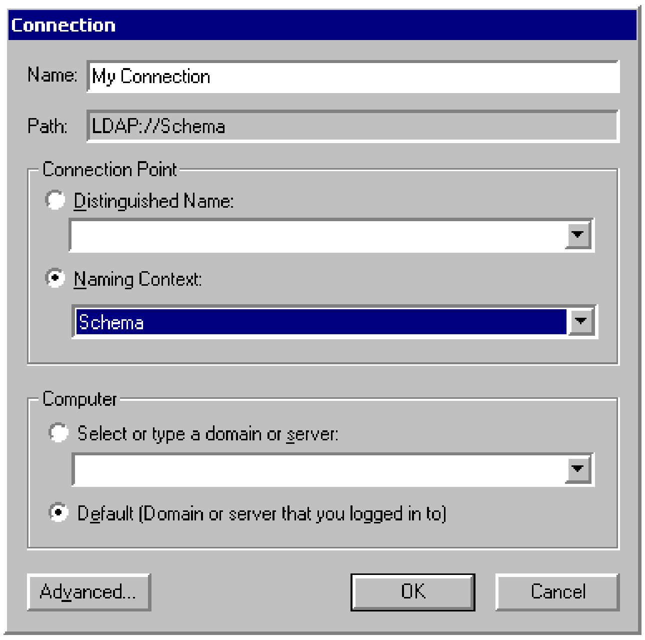
To reactivate a class or attribute by using the Active Directory Schema console

1. Open the Active Directory Schema console.
2. Double-click the **Classes** folder or **Attributes** folder to display the schema classes or attributes.
3. Right-click the class or attribute that you want, and then click **Properties**.
4. Click the **Deactivate this class (attribute)** check box to clear it, and then click **OK**.

To reactivate a class or attribute by using the ADSI Edit console

1. Open ADSI Edit.
2. Right-click **ADSI Edit**, and then click **Connect to**.

   The **Connection** dialog box is displayed, as shown in Figure 4.3.



Connection

Name: My Connection

Path: LDAP://Schema

Connection Point

○ Distinguished Name:

● Naming Context: Schema

Computer

○ Select or type a domain or server:

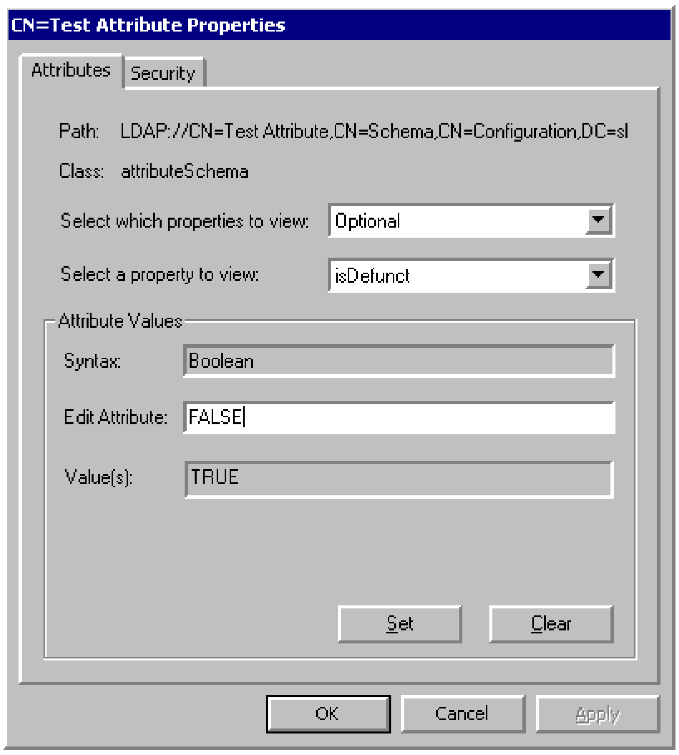
● Default (Domain or server that you logged in to)

Figure 4.3 Connection Dialog Box

3. In the **Connection Point** box, make sure that **Naming Context** is selected.
4. In the **Naming Context** box, select **Schema,** and then click **OK**.
5. In the console tree, double-click **My Connection**.

   The **Schema** folder is displayed.

6. Double-click the **Schema** folder to display a list of attributes and classes in the navigation pane. This might take a few moments.
7. Right-click the class or attribute that you want, and then click **Properties**.
8. In the **Select which properties to view** box, select **Optional,** and then select ***isDefunct*** in the **Select a property to view** box.
9. In the **Test Attribute Properties** dialog box (shown in Figure 4.4), type:
   FALSE
10. Click **Set**, and then click **OK**.



CN=Test Attribute Properties

Path: LDAP://CN=Test Attribute,CN=Schema,CN=Configuration,DC=sl

Class: attributeSchema

Select which properties to view: Optional

Select a property to view: isDefunct

Attribute Values

Syntax: Boolean

Edit Attribute: FALSE

Value(s): TRUE

If your browser does not support inline frames, click here to view on a separate page.

Figure 4.4 Test Attribute Properties Dialog Box

A schema object can be reactivated at any time. The only restriction imposed is that in any such modification, the *isDefunct* attribute is the only attribute present in the modify call. This is done to achieve clean semantics.

The only modification that is allowed on a defunct class or attribute is to modify the *isDefunct* attribute on it to make the class or attribute active again if this is necessary. No other modifications are allowed on a defunct class or attribute. The assumption is that because the object has been deactivated, it is not going to be used for any new modifications; so there is no need to modify it.

Disabling Existing Classes and Attributes

Disabling schema classes and attributes is subject to the following restrictions:

- 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 a "must have" 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 can no longer 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 can no longer be created. To re-enable the class, set the *isDefunct* attribute to FALSE.

Effect of Deactivating a Schema Object on All Objects

After a class A is made defunct, any subsequent addition or modification of instances of A fails as if A has been deleted from the system; that is, the same error codes are returned as if A never existed at all. For example, creating a new instance of A fails and trying to modify or rename an existing instance of A fails. Similarly, if an attribute B is made defunct, B is treated as nonexistent for new object creations and attempts to modify (add or replace) the value of B in an existing object fail.

However, any search or deletion in an object behaves as if no schema objects have been made defunct, the only exception being that schema objects are not allowed to be deleted. So in the preceding example, the user still is able to search for all existing instances of A and delete them if necessary. Similarly, the user can search for all instances that have a value for the attribute B and delete B from such an existing object. This behavior is retained to allow the user to clean up if necessary after a schema object is made defunct. For example, the administrator can decide that a class is not needed anymore and make it defunct so that no one can use it for any modifications. The existing instances of the class can then be cleaned up by searching for all instances and deleting them. Active Directory does not perform any cleanup after a schema object is made defunct.

Similarly, an attribute can be made defunct, and all its uses can be cleaned up. Note that you can delete only the entire attribute from the object, not the values of the attribute. For example, in the preceding example, if B is a multivalue attribute and an object had more than one value for B, trying to delete a value of B from the object fails. This behavior is enforced because there is no reason not to delete the attribute totally when cleaning up a defunct object.

Effects of Deactivating a Schema Object on Schema Updates

In addition to the effects on the instances of the schema object, there are some additional effects on subsequent schema updates when you make a schema object defunct. The additional effects arise mostly because schema updates are subject to special validation checks to which nonschema object updates are not subjected. If a class A or attribute A is made defunct, subsequent schema updates show the following behaviors:

- No modifications are allowed on defunct classes or attributes. The only exception that is allowed is to modify the *isDefunct* attribute on a defunct class A to make the class active again if required. The assumption is that because the class or attribute is made defunct, it is not used for any new modifications. So there is not any need to modify it, except to make it active again if the administrator decides later that it is needed.
- Validation checks that are performed when you add a new class or attribute or modify an existing nondefunct class or attribute treat A as nonexistent. For example, if A is an attribute, trying to modify an existing nondefunct class B by adding *mayContain*=A fails because the validation checks that are performed at schema modification time fail as if A did not exist. Or if A is a class, trying to add a new class with *subClassOf*=A fails because A is treated as nonexistent by the validation checks performed during the addition of the class. The exception is when you try to add or modify a class or attribute to have the same distinguished name, object identifier, *lDAPDisplayName*, *mAPIID*, or *schemaIDGUID* as the defunct class A or attribute A; the operation fails. In this case, A is treated as a nondefunct schema object to ensure that schema consistency is not violated.

This ability to make schema objects defunct can be very useful in different ways in production environments. Schema objects that are no longer needed can be cleaned up by making them defunct. Then existing instances of those classes or attributes can be deleted if desired. At the same time, if the same schema object is found to be of use later, it can be brought back quickly by modifying the object by removing the *isDefunct* attribute on it. This also protects against the accidental removal of a schema object by mistake (by making it defunct). The operation can be reversed easily with no side effects. Note that because Active Directory does not do any cleanup after a schema object is made defunct, all instances of the schema object that was made defunct by mistake remain and become valid, normal objects when the defunct schema object is made active again.

Issues Related to Modifying the Schema

When you modify the schema, you must be aware of the implications and of the potential problems that can arise. There are three main issues involved with modifying the schema: replication, concurrency control, and handling invalid object instances.

Replication

Because the schema is replicated across all domain controllers in the forest, a schema update that is performed at one domain controller is guaranteed to be propagated throughout the forest. This guarantees a schema that is consistent forest-wide. However, because of replication latencies, there can be temporary inconsistencies.

For example, consider that a new class A is created at server X, and then an instance of this class (B) is created at the same server (X). However, when the changes are replicated to another server Y, the object B is replicated out before the *classSchema* object A. When the change arrives at server Y, the replication of B fails because Y's copy of the schema still does not contain the *classSchema* object A. Hence, Y does not know about the existence of A.

Active Directory solves this problem in such scenarios by explicitly replicating the Schema container from the originating server when such failures occur. Additionally, the replication of the Schema container triggers an immediate schema cache update on the target server. Active Directory then re-replicates the object that failed. In the example, re-replication brings in *classdefinition* object A and also puts it into the schema cache of Y. Retrying the replication of B now succeeds.

Concurrency Control

Active Directory must ensure that different program threads do not perform simultaneous, conflicting schema updates (such as when one thread is deleting an attribute and another is adding it to the *mayContain* list of a class).

To ensure this, any thread that attempts to perform a schema update also automatically writes a special attribute on the Schema container as part of the transaction. (Active Directory automatically causes the thread to write the attribute you do not have to do so in your program code.) Only one thread can write this attribute at any one time. This method guarantees schema consistency, but it does not guarantee which of the updates is successful. You must be aware of this when schema updates are made in a batch (such as in the case of the installation of directory-enabled applications).

For example, consider a scenario in which two Active Directory–aware programs, A and B, are being installed simultaneously, each of which creates several new schema objects. Because Active Directory creates one thread per object update, it is possible that some of the objects in program A and some of the objects in program B get created (if the internal threads do not overlap), and then one of the installations fails (because a thread for a schema object creation for program A overlaps with a thread for a schema object creation for program B).

Assume that program A fails. Now running A from scratch again does not work because some of the objects that program A created are already in the schema; trying to re-create them in the second run (existing objects) returns an error. Therefore, it is recommended that programs that modify the schema not be run concurrently, unless provisions are made in the program to first check if the schema update that is about to be made has already been made and then proceed accordingly.

Handling Invalid Object Instances

Schema update can make an existing instance of an object invalid. For example, suppose object X is an instance of class Y. Class Y has an attribute, Z, in its *mayContain* list. Therefore, because object X is an instance of class Y, object X can have this attribute defined on it. Assume that X does indeed have this attribute currently defined in it. Now a schema update is performed that modifies class Y by deactivating attribute Z from its *mayContain* list. Note that this change makes the instance of object X invalid because X now has an attribute, Z, that it is not allowed to have according to the class definition of Y (of which object X is an instance). Active Directory allows the now-invalid objects to remain in the directory and ensures that they do not cause any problems in the rest of the schema. Active Directory does not automatically clean up invalid objects, but invalid objects and attributes appear in searches and can be deactivated manually.

Methods for Extending the Schema

Windows 2000 gives you some choices regarding the way you accomplish schema extension. You can import and export objects in a batch mode by using each of these administrative tools: LDIF Directory Exchange (LDIFDE), CSV Directory Exchange (CSVDE), and ADSI scripts. These tools enable you to administer large numbers of objects (such as users, contacts, groups, servers, and printers) in one operation. By using these tools, it is possible to export Active Directory data to other applications and services and to import information from other sources into Active Directory. These tools are installed automatically on all Windows 2000 servers. Or you can perform schema extension programmatically by using ADSI Edit. You can also extend the schema from the user interface with the Active Directory Schema console.

LDAP Data Interchange Format

The LDAP Data Interchange Format (LDIF) (file) format has a command-line utility called "LDIFDE" that allows you to create, modify, and delete directory objects. It can be run on a Windows 2000–based server or copied to a Windows 2000–based workstation. For example, LDIFDE can be used to extend the schema, export Active Directory user and group information to other applications or services, and populate Active Directory with data from other directory services.

LDIF is an Internet standard for a file format to perform batch import and batch export operations for directories that conform to LDAP standards. An LDIF file consists of a series of records that are divided by line separators. A *record* describes either a single directory entry or a set of modifications to a single directory entry and consists of one or more lines in the file.

Using the LDIFDE Tool

The LDIFDE tool is executed from the command prompt. At the prompt, type the command **LDIFDE** and the appropriate parameters in the following form:

LDIFDE [**-i**] [**-f**] [**-s**] [**-c**] [**-v**] [**-t**] [**-d**] [**-r**] [**-p**] [**-l**] [**-o**] [**-m**] [**-n**] [**-j**] [**-g**] [**-k**] [**-a**] [**-b**][**-?**][-**u**][-**y**]

Note A hyphen (-) is required before each parameter.

Table 4.13, Table 4.14, Table 4.15, and Table 4.16 contain descriptions of all of the parameters.

Table 4.13 LDIFDE Tool Basic Parameters

Basic Parameters	**Value(s) to Specify**	**Description**
-i	mode	Specifies import mode. If this parameter is not specified, the default mode for LDIFDE (and CSVDE) is export.
-f	filename	Identifies the import or export file name.
-s	server name	Specifies the domain controller to perform the import or export operation. If this parameter is not specified, the operation communicates with the domain controller of the domain to which the user is currently logged on.
-c	from distinguished name (string1) to distinguished name (string2)	Replaces all occurrences of string1 with string2. This usually is used when you are importing data from one domain to another and the distinguished name of the export domain has to be replaced with that of the import domain. This parameter is designed to support the import of data when the receiving domain name is different than the exporting domain name.
-t	port number	Specifies a port number. The default LDAP port is 389. (The Global Catalog port is 3268.)
-v	verbose mode	Sets verbose mode, which provides more detailed status description of the import/export operation. If this parameter is not specified, the default is nonverbose mode.
-?	Help	Use to display Help.

Table 4.14 LDIFDE Tool Export-specific Parameters

Export-specific Parameters	**Value(s) to Specify**	**Description**
-d	base distinguished name	Sets the distinguished name of the search base for data export. If this parameter is not specified, it defaults to the root of the domain.
-r	LDAP filter	Creates an LDAP search filter for data export. For example, to export all users with your surname, the following filter could be used: -r "(&(objectClass=user)(sn=yoursurname))". Note that the default is (objectClass=*). For more information about LDAP search filters, see "Name Resolution in Active Directory" in this book.
-p	scope	Sets the search scope. Values are: Base, OneLevel, or SubTree. If not specified, the default is SubTree. For more information about the search scope, see "Name Resolution in Active Directory" in this book.
-l	LDAP attribute list	Sets the list of attributes to return in the results of an export query. If this parameter is omitted, all attributes are returned. For example, to retrieve only the distinguished name, common name, first name, surname, and telephone number of the returned objects, the following attribute list would be specified: -l "distinguishedName, cn, givenName, sn, telephone". (Note: Quotation marks around the list of attributes is optional.)
-o	attributes in results	Omits a list of attributes from the results of an export query. This is used when exporting objects from Active Directory and then importing them into another LDAP-compliant directory. Some attributes might not be supported in the directory receiving the objects. For example, to omit *whenCreated* and *whenChanged*, the following would be specified: -o "whenChanged, whenCreated". This parameter omits the attributes from the results. If not specified, all attributes are included (Note: Quotation marks around the list of attributes is optional.)
-m	Active Directory attributes	Omits attributes that apply to only Active Directory objects such as *objectGUID* (globally unique identifier), *objectSID* (security identifier), *pwdLastSet* (password last set), and *samAccountType*. By default, this parameter is disabled. Its primary purpose is to export entries in preparation of re-importing them into Active Directory. This also activates the linked attribute option, which appends to the end of the file the values for attributes that are linked to the current object. For example, a parent object has linked attributes to a child object, and those entries are placed at the end of the file. (Note: Some attributes are read-only for Active Directory and, thus, cannot be re-imported. The -m option strips these attributes out at the time of export to prepare the file for re-importing.)
-n	binary values	Specifies not to export binary values. By default, this parameter is disabled.
-j	directory path	Sets the log file location. The default is the current directory.
-g	paged searches	Specifies not to perform paged searches. If not specified, it performs paged searches. Note that some servers might not support the paged search control.

-u	Unicode	Enables Unicode support. When this parameter is used during an export operation, a Unicode file is generated. When the parameter is used during an import operation, the tool would expect a Unicode file as input.
-y		Enables lazy commit to the directory.

Table 4.15 LDIFDE Tool Import-specific Parameters

Import-specific Parameters	Value(s) to Specify	Description
-k	action if errors are encountered	Skips errors during the import operation and continues processing. If not specified, the import operation stops if it encounters the following errors: LDAP_ALREADY_EXISTS LDAP_CONSTRAINT_VIOLATION LDAP_ATTRIBUTE_OR_VALUE_EXISTS LDAP_NO_SUCH_OBJECT ERROR_MEMBER_IN_ALIAS It will also skip objects with no attributes. For more information about these errors, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

Table 4.16 LDIFDE Tool Credentials Parameters

Credentials Parameters	Value(s) to Specify	Description
-a	user distinguished name password OR *	Sets the command to run by using the supplied user distinguished name and password. The default is to run by using the credentials of the currently logged on user. For example, -a "cn=yourname,dc=yourcompany,dc=com password". * = option to hide password
-b	username domain password OR *	Sets the command to run as the username domain password. The default is to run using the credentials of the currently logged on user. * = option to hide password

Note Make sure that all required attributes exist when you create or modify objects. For example, the required attributes for creating a user are *distinguishedName*, *objectClass*, and *samAccountName*.

Exporting and Re-Importing Objects

Linked attributes contain information about the links to a current object. During a normal export session, a parent object might be exported before its child object. On the re-import operation, if the parent object has been added before the child object, the operation fails because the child object is not yet in the directory.

However, when the -m parameter is used to export objects and re-import them into Active Directory, all entries that contain a linked attribute are appended to the end of the file. Moreover, the linked addition is separated from the main object creation call so that the failure in membership addition does not cause the object creation to fail. The linked attribute is appended to the end of the file.

Read-only Properties on Objects

Active Directory has Security Accounts Manager (SAM) properties that are read-only because they are set by the system at the time the object is created. When the -m parameter is used to export objects and re-import them into Active Directory, all of the SAM attributes are ignored during the export operation. In that way, when the entries are re-imported into Active Directory, they succeed because they do not contain any SAM information.

Example of an LDIF Import File

In the following example of an LDIF import file format, you also can see how to add a *user* object to the myDomain.microsoft.com domain:

```
dn: CN=sampleUser,CN=Users,DC=myDomain,DC=microsoft,DC=com
changetype: add
cn: sampleUser
description: Example of an Imported User using LDIFDE
objectClass: user
sAMAccountName: sampleUser
```

The following is an example of the command that is used to import the file in the preceding example:

```
ldifde -i -f import.ldf -v
```

Manipulating Data in an LDIF Export File

The preferred method of manipulating the distinguished name (*distinguishedName*) during an LDIFDE export operation is to use the –c parameter. For example, by using this parameter in conjunction with the –m parameter, you can import a large group of users from one domain into another domain.

Note You must use a text editor to make substantial changes to attribute values in your export file prior to import.

Comma-Separated Value File Format

The bulk import and export of data to and from Active Directory can be performed by using files that store data in the Microsoft comma-separated value (CSV) file format, also known as a .csv file. The CSV file format is supported by many other applications, such as Microsoft® Excel, that can read and save data in the CSV file format. Also, Microsoft® Exchange Server administration tools can import and export data by using the CSV format. The CSV format has a command-line utility called "CSVDE" that allows you *only* to add new objects. It can be run on a Windows 2000–based server or copied to a Windows 2000–based workstation.

The CSV format consists of a simple text file with one or more lines of data where each value is separated by a comma. The text file contains entries where the initial entry is a comma-separated list of attribute names. Each subsequent entry in the text file represents a single object in the directory. Attribute values are delimited by commas.

Using the CSVDE Tool

The CSVDE tool is executed from the command prompt. At the command prompt, type the command **CSVDE**. The parameters that are used for the CSVDE tool are the same as those that are used for the LDIFDE tool. However, unlike the LDIFDE tool, CSVDE creates files that can be read from applications other than LDAP servers. For example, if you want to view all Active Directory users in a Excel report, CSVDE is used to export the directory data into the .csv file format, which could then be read by Excel.

The CSVDE tool is executed from the command prompt. At the prompt, type the command **CSVDE** and the appropriate parameters in the following form:

CSVDE [**-i**] [**-f**] [**-s**] [**-c**] [**-v**] [**-t**] [**-d**] [**-r**] [**-p**] [**-l**] [**-o**] [**-m**] [**-n**] [**-e**] [**-j**] [**-g**] [**-k**] [**-a**] [**-b**] [**-?**][**-u**]

The descriptions of these parameters are contained in Tables 4.13, 4.14, 4.15, and 4.16. As for the LDIFDE tool, the default mode for CSVDE is Export, unless otherwise specified by using the -i parameter for the import mode.

Note CSVDE cannot be used to modify or delete objects. It can be used only to *add* directory objects. A hyphen (-) is required before each parameter.

Viewing Data in the .csv File

When you view data in the .csv file, the values for multivalue attributes are expressed as a single value that is internally delimited by a second user-definable delimiting character (by default, $). Attribute values are listed left to right in the order in which the attribute names are listed in the initial entry. Values are positional, and every entry must account for each attribute listing in the initial entry. The attribute names must be in the same order as the data in any line that follows the first line, as shown in the following example:

```
dn,cn,firstName,surname,description,objectClass,sAMAccountname
"cn=John Smith,cn=Users,dc=myDomain,dc=microsoft,dc=com",John
Smith,John,Smith,Manager,user,jsmith
"cn=Jane Smith,cn=Users,dc=myDomain,dc=microsoft,dc=com",Jane
Smith,Jane,Smith,President,user,janes
```

Each object stands alone and does not need the context of another object to be listed in the .csv file, which simplifies the reading and writing of files and allows objects from different classes to be contained in a single file.

Another example shows the .csv file format and lists the header, which contains the LDAP display names of the properties ("attributes") — distinguished name, object class, common name, given name, surname, telephone number, street address, locale, country/region, and sAMAccountName.

```
dn,objectClass,cn,givenName,sn,telephoneNumber,
street,l,c,sAMAccountName
"cn=James Smith,cn=Users,dc=myDomain,dc=microsoft,dc=com",user,James
Smith,James,Smith, ,203-223-2233, 15 Woodbine St.,Fenwick,US,jsmith
```

All data values are represented as strings. Numeric values are represented by numeric strings; binary values are represented by hexadecimal strings. Hexadecimal strings start with the character "x," followed by a single quotation mark ('), then the hexadecimal string, and, finally, another single quotation mark ('). The following is an example of a hexadecimal string:

```
X'0105000000000005150000007 9e3fc535729024c235f636bf5010000'
```

Syntax information is stored in the schema of the destination directory. Programs that accept imported .csv files determine how to process the values by using the schema in the target directory.

A missing or unsupported attribute value has an empty position in the string. For example, if the third attribute value for an entry is missing, it would be expressed as follows:

```
firstvalue,secondvalue,,fourthvalue
```

Multivalue attributes are separated by semicolons (;). For example if there are three attributes and the second one is a multivalue attribute, it would be expressed as follows:

```
1stvalue,2ndvalue1;2ndvalue2;2ndvalue3,3rd value
```

Reserved characters that appear in string properties are represented through an escape mechanism. The following are reserved characters:

- Backslash (\)
- Semicolon (;)
- Special character for hexadecimal representation (x')

The escape mechanism uses a backslash (\) before a reserved character as an escape character. If a value contains a backslash, the backslash in the value also must be preceded by the escape character — that is, by another backslash (for example, \\). The semicolon (;) character is used to delimit multivalues. If the value itself contains a semicolon, the semicolon in the value must be preceded by the escape character (for example, \;). The hexadecimal prefix (x') character, if used in a value, must also be preceded by an escape character (for example, \x').

There are two other characters that must be handled in a special way. They are the comma (,) and the double quotation marks (") characters. The comma (,) is treated as a special character in the CSV format because it is used to separate values. If the value contains a comma (,), the format specifies that the comma has to be enclosed by double quotation marks (for example, value1,value2 are represented as "value1,value2"). The double quotation marks (") character is used to contain values if the values contain commas (,). When a value contains a pair of double quotation marks as well as a comma, the quotation marks in the value have to be enclosed with another set of double quotation marks, as follows:

```
"value1","value2" is represented as ""value1"",""value2""
value1"value2 is represented as "value1""value2"
```

The following CSV file shows an example of adding an organizational unit, followed by a user, and a computer:

```
dn,cn,givenName,sn,description,objectClass,sAMAccountname
"ou=sampleOU,dc=myDomain,dc=microsoft,dc=com",,,,Sample
Organizational Unit,organizationalUnit,
"cn=John Smith,ou=sampleOU,dc=myDomain,dc=microsoft,dc=com",John
Smith,John,Smith,Sample User,user,jsmith
"cn=sampleComputer,ou=sampleOU,dc=myDomain,dc=microsoft,dc=com",samp
leComputer,,,Sample Computer,computer,computer1
```

Note Both ANSI text and UNICODE are supported.

Using LDIFDE and CSVDE to Modify the Schema

LDIFDE and CSVDE use files that contain directory data in the appropriate format. These files can be imported or exported to LDAP-based directory servers as a means of populating a directory or modifying objects in a directory. Because the Active Directory schema exists as a collection of directory objects, either of these tools can be used to extend the schema.

Note At present, CSVDE can be used only for additions to the directory, not for modifications to the directory.

LDIF File Format

The LDIF file format can be used to perform batch operations on directories that conform to LDAP standards. It is suitable for additions to the directory as well as modifications and deletions of directory objects. A *record* in an LDIF file consists of a sequence of lines that either describe a directory entry or a set of changes to a single directory. This format can be used for all LDAP operations.

The preferred method when modifying the schema is to use the Active Directory Schema console to edit the schema on a practice system that is isolated from your real enterprise. You can use the LDIFDE export to produce a script, which you can then run against your live system. The following example represents the contents of a sample LDIF file that can be used to add a new attribute to Active Directory.

```
dn: CN=New-Attribute-Name,CN=Schema,CN=Configuration,
DC=microsoft,DC=com
changetype: add
objectClass: attributeSchema
ldapDisplayName: newAttributeName
adminDisplayName: New-Attribute-Name
adminDescription: New-Attribute-Name
attributeId: 1.2.840.113556.1.4.8000.1 <- the id has to be unique
attributeSyntax: 2.5.5.12
omSyntax: 64
isSingleValued: TRUE
systemOnly: FALSE
searchFlags: 0
showInAdvancedViewOnly: TRUE
```

The following example shows an LDIF file that can be used to force a schema cache update.

```
dn:
changeType: modify
add: schemaUpdateNow
schemaUpdateNow: 1
```

For more information about the LDIF format and using LDIFDE, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

CSV File Format

The CSV file format is a simple format whose primary benefit is ease of use. In the CSV file format, each line represents a discrete object in the directory, with the object's attributes separated by commas. The first line of the file always contains all of the attribute names. Each subsequent line represents a different entry in the directory. Values for multivalue attributes can also be specified and are delimited by the semicolon (;) character.

Because this format is compatible with the Microsoft Excel CSV format, you can dump directory information to an Excel spreadsheet or import data from a spreadsheet into Active Directory. This format can be used only for additions to the directory. The following example represents the contents of a CSV file that can be used to add a user to Active Directory:

```
dn,objectClass,cn,sn,givenName,telephoneNumber,street,l,c,sAMAccountName
"CN=John Doe,DC=myDomain,DC=microsoft,DC=com",
User,John Doe,Doe,John, 555-456-7890,123 Magnolia Ave.,Redmond,US,jdoe
```

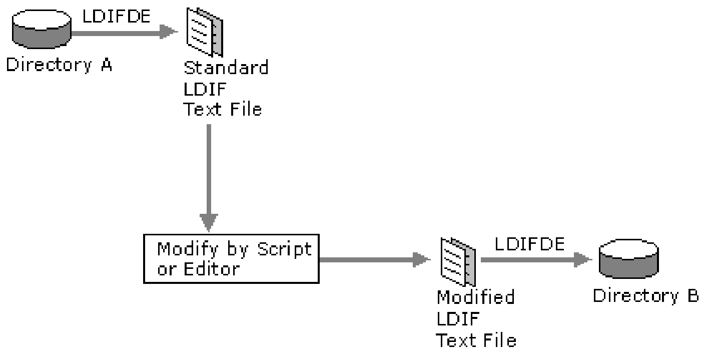
Both the CSVDE and LDIFDE tools have command line help that can be viewed by typing the command name at an MS-DOS® prompt. Because both of these tools allow data to be imported and exported, there are a number of different uses for them.

Migration to Active Directory By using either of these tools, users of other directory services can import data to Active Directory. This works for any directory that is LDAP-compatible as long as the attribute names match.

Publishing Information from the Directory You can use either of these tools to export directory data to another application that can read either the LDIF or CSV format. You can also export to other LDAP-compatible directory services, provided there are matching attribute names.

Adding Resources to the Directory In addition to the Active Directory Users and Computer console and ADSI Edit, administrators can choose to use one of these tools to add objects to a directory. These tools lie somewhere between the other options in terms of ease of use and flexibility. Because the schema is represented in Active Directory as directory objects, you can use LDIFDE or CSVDE to extend the schema with new or modified schema objects. In fact, if your application requires schema modifications, the best way to accomplish this is to distribute an LDIF or CSV file with the application that is to be imported to the schema.

Figure 4.5 illustrates one way in which LDIF can be used to extend Active Directory.



If your browser does not support inline frames, click here to view on a separate page.

Figure 4.5 Extending Active Directory with LDIF

Using Active Directory Service Interfaces and Visual Basic Scripts

Although one potential benefit of using an LDIF or CSV file is that the administrator can look at it to see what it does, consider the merits of extending the schema programmatically:

- A programmatic extension is invariant; it is a Windows executable file. The binary cannot be tampered with, unlike an LDIF or a CSV file, either of which can be modified inadvertently or maliciously.
- Programs can detect and recover from errors and provide intelligent feedback.
- Programs handle Unicode without resorting to Base64 encoding. (Unicode is a 16-bit character set that contains all of the characters commonly used in information processing.)
- Programs can use the Windows Installer (.msi) setup APIs.

- Programs can be signed to prove their authenticity.

Active Directory provides a set of interfaces that enable you to gain access to directory objects, including schema objects, programmatically. ADSI defines a directory service model and a set of COM interfaces that you can easily use with a variety of programming languages. ADSI conforms to the Component Object Model and supports standard COM features.

By using Microsoft® Visual Basic® Script and ADSI, you can write scripts easily to accomplish various directory modifications, including extending the schema.

These are the specific ADSI interfaces to use when you extend the schema:

IADsContainer Use **IADsContainer::Create** to create new *classSchema* and *attributeSchema* objects.

IADs Use **IADs::Get (or GetEx)** to read the attributes of *classSchema* and *attributeSchema* objects. Use **IADs::Put** (or **PutEx**) to set the attributes of *classSchema* and *attributeSchema* objects. PutEx is particularly useful in manipulating the lists of *mustContain* and *mightContain* attributes because it is designed specifically for handling multivalue attributes.

The code in the example that follows represents a script that you can use to add a user to Active Directory.

```
Dim oDomain
Dim oUser

Set oDomain=GetObject("LDAP://OU=Marketing,DC=antipodes,DC=com")
Set oUser = oDomain.Create("user","cn=John Smith")
oUser.Put "samAccountName","JSmith"
oUser.Put "givenName","John"
oUser.Put "sn","Smith"
oUser.Put "userPrincipalName","jsmith@antipodes.com"
oUser.SetInfo
MsgBox "User created " & oUser.Name
Set oDomain = Nothing
MsgBox "Finished"
WScript.Quit
```

Note For more information on ADSI and ADSI interfaces, see the Microsoft Platform SDK link on the Web Resources page at http://windows.microsoft.com/windows2000/reskit/webresources .

Using the Active Directory Schema Console

The Active Directory Schema console allows members of the Schema Administrators group to manage the schema through a graphical interface. With it, you can create and modify classes and attributes and also specify what attributes are indexed and what attributes are replicated to the Global Catalog. After you start the Active Directory Schema console, the first thing that you must do is to make sure that the tool is focused on the schema master for your enterprise.

Note The Schema Management snap-in is not one of the default MMC snap-ins that is provided with Windows 2000 Server. To make it appear in the list of available snap-ins, you must install the admin tools package (Adminpak.msi). To register the Schema Management snap-in, open your %*systemroot*%\System32 folder and run **Regsvr32** Schmmgmt.dll from the command prompt or from the **Run** command on the **Start** menu.

To view or change the current schema master by using the Active Directory Schema console

1. Open MMC, and then install the Active Directory Schema snap-in.
2. Right-click **Active Directory Schema**, and then click **Operations Master**.
3. The **Current Operations Master** that is displayed is the schema master. Click **OK** to leave it as is.

   – Or –

   Click **Change** to change the server that is the current FSMO Role Owner for the schema. If the current domain controller (the one that is listed in **Current Focus**) is also the current operations master, you must use the Active Directory Tree console in MMC to focus on another domain controller before you can change the operations master.

After you have verified that the tool is focused on the current schema master, you can use it to add, modify, or deactivate attributes and classes. Remember that the account you are using must be a member of the Schema Administrators group and that the server must be set to allow schema modifications.

Note Schema objects that are part of the default base schema cannot be deactivated.