PART III

ADVANCED SKILLS

Chapter 8

Active Directory Schema

This chapter describes the Active Directory data model and the way in which the rules of the schema enforce it. You can approach this chapter with the following three purposes:

- To understand better how Active Directory works behind the scenes
- To enhance administration by learning, for example, which attributes are indexed to make searches faster
- To learn the technical background to extend the schema of your forest and to prepare you for the next chapter about schema extensions

This chapter has the following sections:

- Overview of the Active Directory data model
- Schema in general
- Classes
- Attributes and syntaxes

OVERVIEW OF THE ACTIVE DIRECTORY DATA MODEL

Active Directory uses the LDAP data model, which is derived from the X.500 data model and, consequently, Active Directory implements many of X.500 features. However, Active Directory is not a full X.500 directory service.

In this first section we introduce the Active Directory data model, including most notably classes and attributes, as well as their relationship to the normal user and other objects that you see in your domains.

Classes, Objects, and Attributes

As you know, information in Active Directory is represented as objects, and there is an object for each user, computer, printer, and so on. Objects of the same type belong to the same *class,* so all user objects belong to the class `user`, all computer objects to the class `computer`, and all printer objects to the class `printQueue`.

You also know that information in an object is stored as values of various *properties,* which the corresponding class supports. For example, a user may have a phone number and home folder, whereas a printer might have knowledge about supported forms or print speed. Because this chapter is about the inner architecture of Active Directory, we want to use the slightly fancier term *attribute* instead of *property.* You can use these two words interchangeably.

The *schema* dictates which object classes and attributes a given forest supports. The *base schema* that ships with Windows 2000 supports 142 object classes and 863 attributes for those classes. The `user` class uses 207 attributes of the pool of 863. Active Directory implements most of the X.500 standard classes, but not quite all. The classes that it does not implement are `alias, strongAuthenticationUser`, and `groupOfUniqueNames`.

NOTE The attribute pool is common to all classes. The `description` attribute, for example, may be used by several classes (actually, all classes use it).

Bringing in another fancy term, you could say that each object is an *instantiation* of the corresponding class. For example, the Jack Brown object is an instantiation of the `user` class. Just as the class contains a number of attributes, the instantiated object contains the values for those attributes. However, only some of the attributes for a class are *mandatory,* which means that they must contain a value (a *single-valued* attribute such as `homePhone`) or values (a *multivalued* attribute such as `otherHomePhone`). Most of the attributes for a given class are *optional;* that is, they may contain a value, but more often they don't. Figure 8.1 shows these relationships.

NOTE You can see only a small subset of object types in the Users and Computers snap-in and only a subset of attributes for each object.

NOTE To create user objects with the Users and Computers snap-in, you must enter the user principal name (i.e., user logon name), even though it is just an optional attribute behind the scenes.

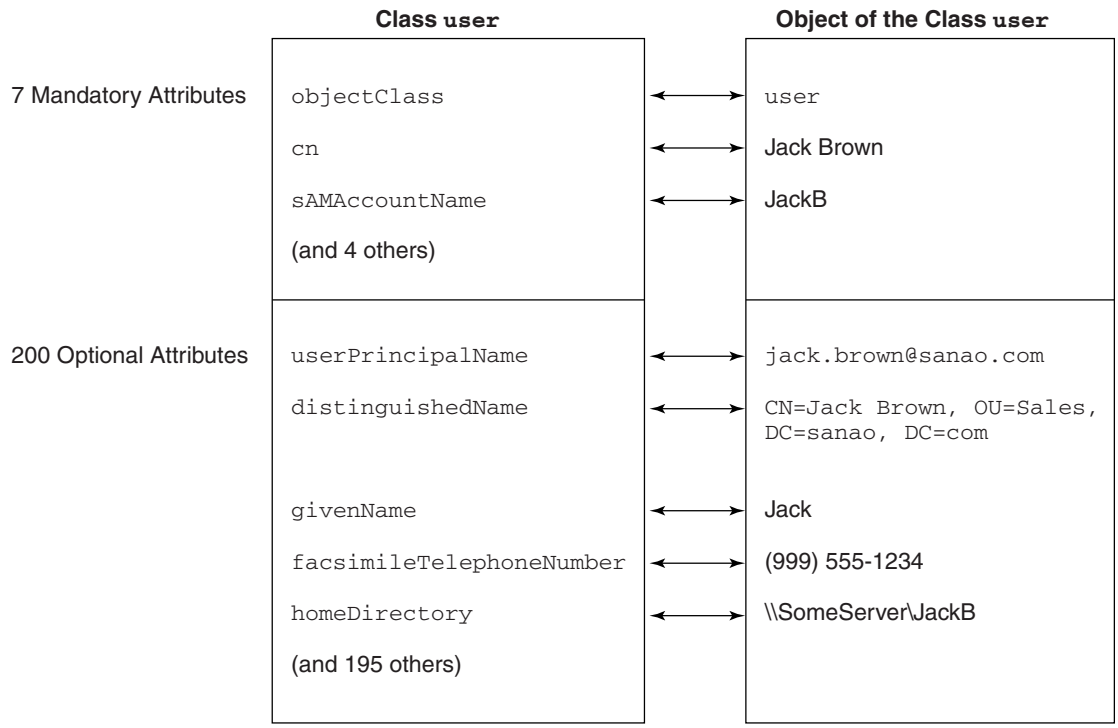


FIGURE 8.1 The class `user` defines 7 mandatory and 200 optional attributes for which any user object must *or* may contain values.

Each attribute has one of 23 *syntaxes,* such as `Integer` 14, `Unicode string` "abc", or `Generalized time` 04/26/2000 2:59:01 PM. We discuss syntaxes later in this chapter.

As you see in Figure 8.2, some of the classes (actually, most of them) are something other than the familiar users, groups, and computers.

Container and Leaf Objects

Active Directory objects of some classes are *container objects,* and the objects of the remaining classes are *leaf objects.* If you compare this to a file system, container objects correspond to folders and leaf objects correspond to files. We call these two types of classes *container classes* and *leaf classes.*

One obvious example of a container class is `organizationalUnit`. There are also many others, however, because a total of 56 classes of the base schema are container classes and 86 are leaf classes. Interestingly, `user` is also a container class; `user` objects may contain `nTFRSSubscriptions` and `classStore` objects.

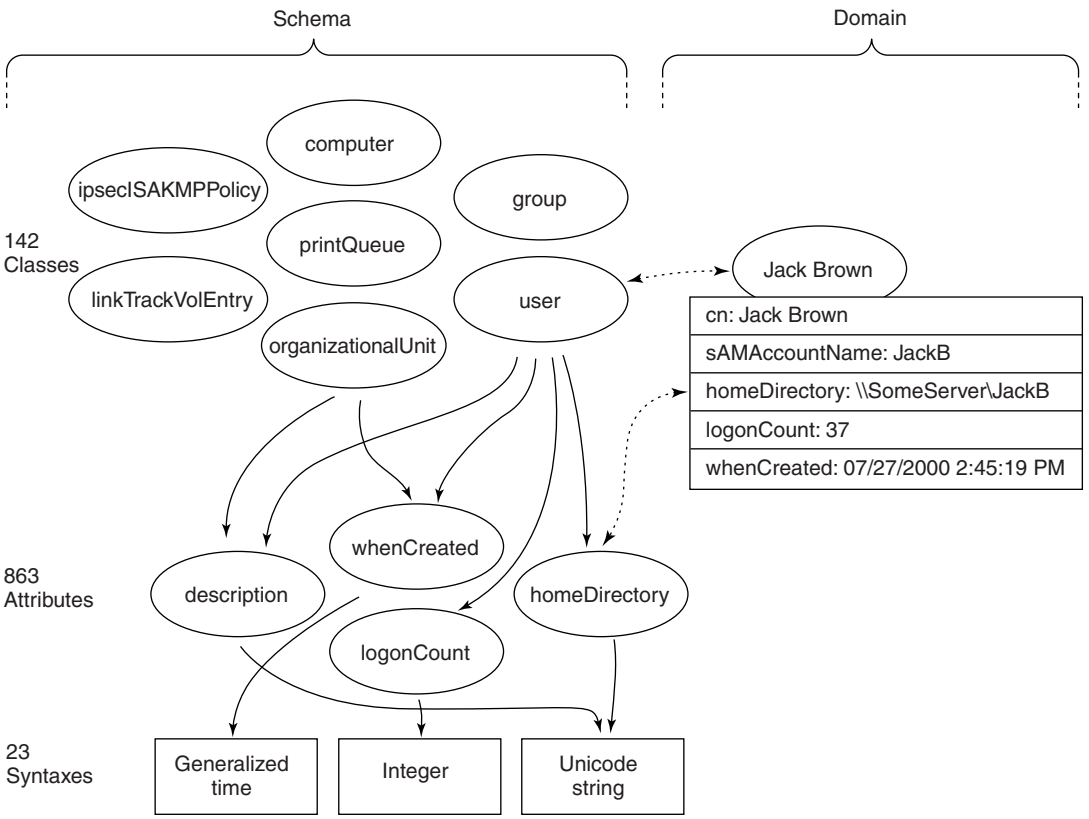


FIGURE 8.2 Each of the 142 classes uses a number of attributes, which in turn use a certain syntax.

NOTE "Container" can also mean a specific class—one of the 56 container classes is called `container`. Two examples of this class are `CN=Users` and `CN=Computers`, which you see with the Users and Computers snap-in under any domain.

Even though the base schema contains 86 leaf classes, those classes are not doomed to "eternal leafhood." You can modify the schema to make a leaf class a possible superior (i.e., a parent) to some other class. When you do so, a former leaf class (except for schema objects) becomes a container class.

One difference in our file system analogy is that in Active Directory both containers and leafs contain data, whereas in a file system files contain data, but folders don't.

NOTE Just as there is a class called `container`, one of the 86 leaf classes is called `leaf`. You can make also that class a container, so one of your container classes could be called `leaf`.

Indexing and the Global Catalog

An Active Directory database can contain thousands or even millions of objects. Therefore, we obviously need *indexing* to locate the right object or objects fast. Of the base schema, 64 attributes are indexed, examples being `givenName`, `sn` (Surname) and `birthLocation`. Searching and retrieving objects via indexed attributes is naturally much faster and efficient than using nonindexed attributes.

The *global catalog* helps in making local searches in a multidomain forest. One hundred thirty-eight attributes of the base schema are part of the global catalog, including `cn`, `userPrincipalName`, `givenName`, `sn` (Surname), and `printStaplingSupported`.

SCHEMA

In this section we explain the role of the schema, its location, and its inspection. We also explain briefly the topics of subschema subentry and schema cache.

Role of the Schema

The schema contains rules for object instantiation—that is, it dictates which objects a directory can contain, their relationships, and their possible content. (This is true for other directory services as well as Active Directory.) In other words, the schema governs the structure and content of Active Directory with structure and content rules. Table 8.1 explains these uses of the schema.

Table 8.1 describes why the schema is necessary to maintain a directory. In addition, the schema contains information that helps to maintain the schema itself. This is described in Table 8.2. We explore the topics introduced in Table 8.1 and Table 8.2 in the later sections "Classes" and "Attributes and Syntaxes."

Location of the Schema

Because everything in Active Directory is stored in objects, the schema is implemented as a number of objects. There is one object for each class in the schema (`classSchema` objects) and one for each attribute (`attributeSchema` objects).

TABLE 8.1 Uses of the Schema

Category	Description
Structure rules	• Possible parent classes for each class (i.e., under what classes of objects each object can exist). For example, a user object may exist under the container types `organizationalUnit`, `domainDNS`, `builtinDomain`, `container`, and `lostAndFound`.
Content rules	• The mandatory and optional attributes for each object class. • For each attribute there is a certain syntax and value range, as well as the choice between single-value and multivalue.
Miscellaneous	• Which attributes are indexed and which are stored in the global catalog. • The default security descriptor, category, and hiding value for each new object. • The name type (i.e., the naming attribute) of a class: CN, OU, or DC.

TABLE 8.2 Inside Uses of the Schema

Category	Description
Naming and identification	• Various names and ID numbers for schema classes, attributes, and syntaxes
Schema class hierarchy	• Whether a class is *structural, abstract,* or *auxiliary* • The parent of a class in the inheritance chain • The auxiliary classes that give a class additional attributes
Protection	• Whether a class or attribute is *system only* • The security descriptor for a class or attribute

However, there are no objects for the syntaxes; they are hard-coded into Active Directory. This means that classes and attributes can be created and modified, but syntaxes cannot.

NOTE A directory service vendor could implement the schema as a long text file. Microsoft chose to implement the Active Directory schema as Active Directory objects. This enables administrators and applications to query the schema contents and add or modify classes and attributes using the same object manipulation techniques as would be used with any Active Directory objects.

Consequently, the location of the schema is the location of the schema objects.

THE PHYSICAL LOCATION OF THE SCHEMA

As you learned in Chapter 5, every domain controller stores a full replica (or copy) of three partitions (or replication units): the schema, configuration, and domain partitions. Obviously, the schema (i.e., the schema objects) is physically located in the schema partition.

You also learned that the schema partition is replicated to every domain controller in a forest, so that all domain controllers contain identical information. Any changes to the schema would have to be initiated on the domain controller that holds the schema master role (as explained in Chapter 5).

Even though there are three separate partitions, their replicas in a given domain controller are stored in the same database table. That table resides in the Active Directory database file, which is called ntds.dit. The default location for the file is the folder C:\Winnt\NTDS.

There is another ntds.dit file located in C:\Winnt\System32. That file serves as the initial database file and it is copied to C:\Winnt\NTDS (or whatever location you choose) during the DCPromo process.

NOTE You may also come up with a file schema.ini (in C:\Winnt\System32). Despite its name, it doesn't initialize the schema in any way. Instead, it contains the information for the initial objects in your tree—mostly for the domain and configuration partitions.

THE LOGICAL LOCATION OF THE SCHEMA

All the 1,005 schema objects (142 classes and 863 attributes) are located in the *Schema container*. That container is of class `dMD` (the letters stand for "directory management domain") and its distinguished name is `CN=Schema`, `CN=Configuration`, `DC=`*`forest_root_domain`*. Figure 8.3 shows the location of the schema container in the directory tree.

NOTE Although there seems to be a `CN=Configuration` object under your root domain, you won't see it in the Users and Computers snap-in, even with Advanced Features turned on. That snap-in is meant to show contents of only domain partitions.

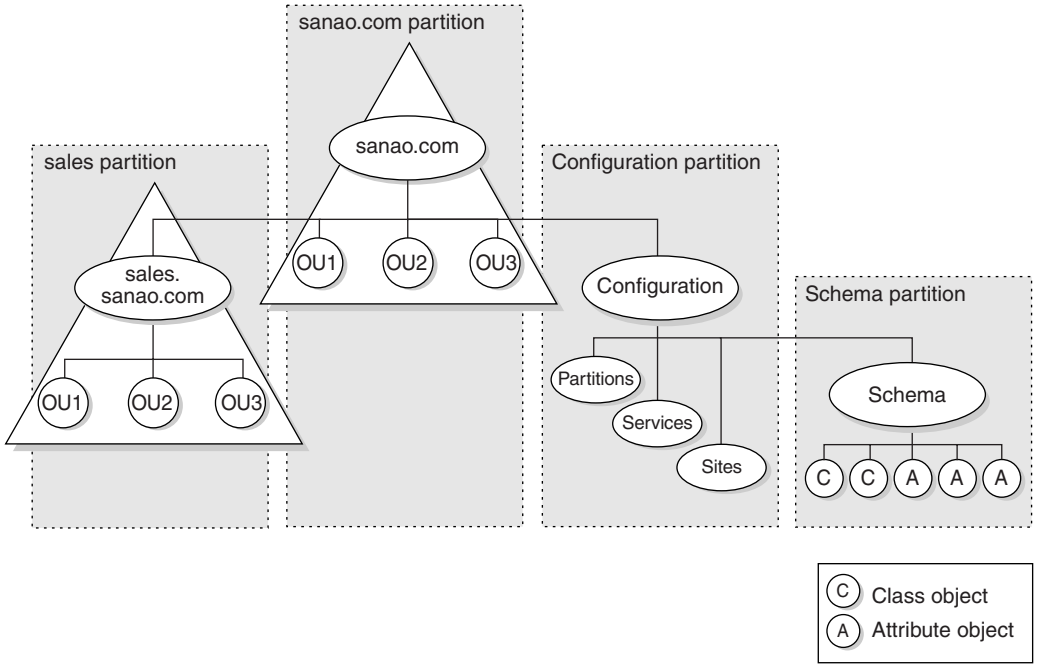


FIGURE 8.3 The schema container is logically under the configuration container, which in turn is under the forest root domain. Physically, however, the three are different partitions.

Inspecting the Schema with ADSI Edit

Now that you know that the schema is implemented as a number of objects and you know where to find them, we can start studying them in the user interface. The two main tools for the job are as follows:

- ADSI Edit, a general tool for viewing all the objects in Active Directory
- The Schema Manager snap-in, a specialized tool for viewing and managing the schema

We begin our discussion of these two tools with ADSI Edit. Because it is a general tool, it nicely shows you the general picture and the location of the schema objects, not just the schema contents.

NOTE There are other tools that you can use to view the schema. For example, Support Tools contains LDP, which enables you to do various LDAP operations—among others, to view the schema.

We mentioned previously that with the Users and Computers snap-in you see only part of the objects and only part of the attributes of each object. With ADSI Edit you see all the objects and all of their attributes.

ADSI Edit is part of Support Tools, which you have to install separately. Locate the folder Support\Tools on your Windows 2000 CD and run Setup.Exe. After the installation, you will find the tool by clicking the Start button and then selecting Programs, Windows 2000 Support Tools, Tools.

Figure 8.4 shows the screen that opens when you launch ADSI Edit. ADSI Edit shows container objects with yellow folder icons and leaf objects with white icons similar to the icons used for text documents in Windows Explorer. The left pane shows only container objects, and the right pane shows both container and leaf objects.

To familiarize yourself with ADSI Edit, it's a good idea to first check the properties (or attributes) of one of your users before you start to explore the schema objects.

INSPECTING ATTRIBUTES OF CLASSES AND ATTRIBUTES

When you are ready to move to the schema objects, you can start inspecting their attributes. For example, you can check the attributes of the `user` class schema object (see Figure 8.5).

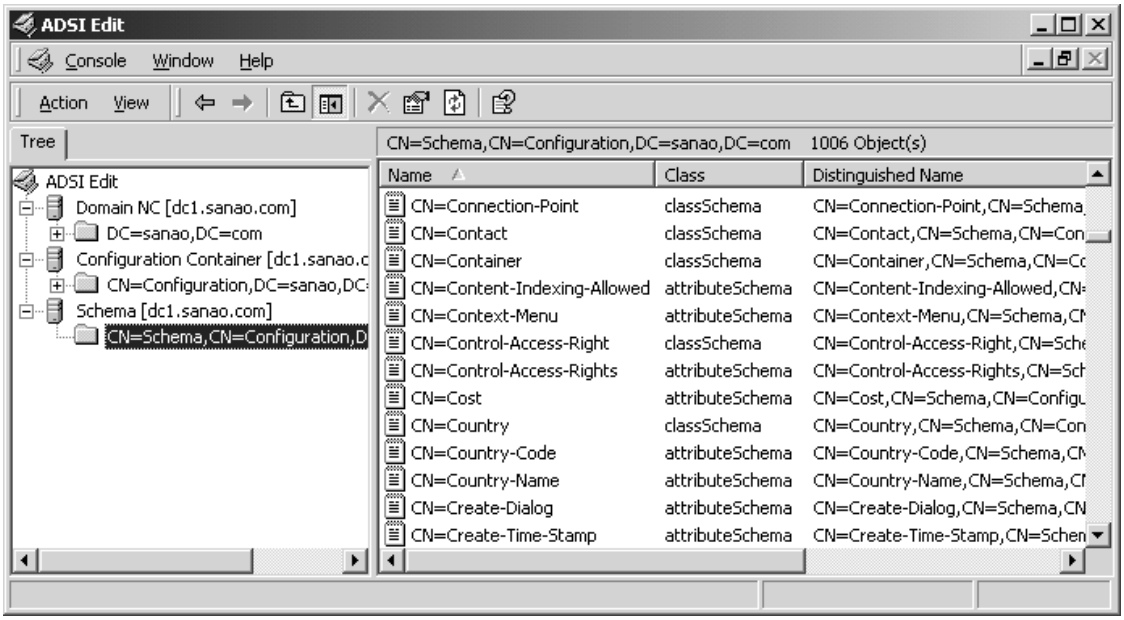


FIGURE 8.4 When you first start ADSI Edit, it shows three partitions (aka naming contexts). Under the Schema container you can see all the class and attribute objects.

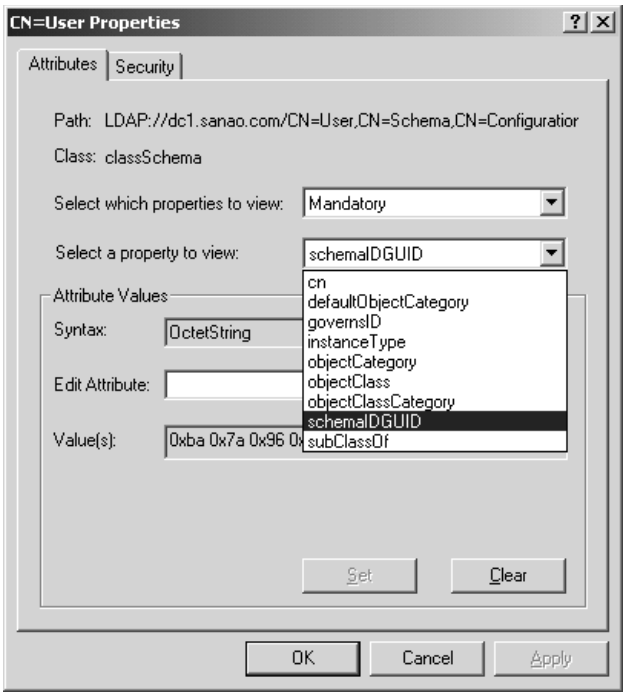


FIGURE 8.5 When you select Properties in the context menu of any object, ADSI Edit opens a dialog box that enables you to browse all the attributes of that object. From the first drop-down list, you can choose Mandatory, or Optional, or Both.

WARNING ADSI Edit enables you to change the attribute values. When you study schema objects, be sure to click Cancel at the end. Do not click OK or Apply.

Note that because all objects contain attributes, `classSchema` and `attributeSchema` objects also contain attributes. For example, the object `CN=Jack Brown` has an attribute `givenName`, with the value "Jack." The schema contains the object `CN=Given-Name`, which in turn contains such attributes as `schemaIDGUID` and `attributeSyntax`.

We discuss the contents of `classSchema` objects a little later in the section "Classes" and `attributeSchema` objects in the section "Attributes and Syntaxes."

NOTE ADSI Edit displays all the attributes of a class or attribute object, but only some of them are meaningful to the schema. Most of the attributes are the same ones that all objects in Active Directory have, such as `uSNChanged`, which helps to track replication.

TIP `SchemaIDGUID` and some other attributes use `OctetString` syntax, which is about the same as binary. If you want to see just the bytes without "0x"s, you can copy and paste the string to Notepad and then replace all "0x"s with nothing.

VARIOUS ATTRIBUTE NAMES

You may have noticed that each attribute has two slightly different names. Table 8.3 summarizes the two naming conventions together with the administrative tool name.

NOTE Unlike the example in Table 8.3, the three attribute names may be quite different, such as "Surname"—"`sn`"—"Last name" or "WWW-Page-Other"—"`url`"—"Web Page Address (Others)." This is the case especially with attributes that have long-established X.500 names.

Just as attributes do, classes have both common names and LDAP names.

Sometimes the common name and LDAP name are the same, but they refer to different attributes. Table 8.4 gives two examples of this (using four attributes).

The attribute names that you see in administrative snap-ins, such as Users and Computers, are not usually stored in the schema. Instead, you find them under the DisplaySpecifiers container, which in turn is under the Configuration container. In this container, first locate the object with your locale identifier (e.g., 409

TABLE 8.3 Various Attribute Names

Name	Example	Naming Convention	Where to Find
Common name	Facsimile-Telephone-Number	Each word starts with an uppercase letter; a dash occurs between words	The common name of the `attributeSchema` object.
LDAP name	`facsimileTelephoneNumber`	Name starts with a lowercase letter; each subsequent word starts with an uppercase letter; no dash between words	The `lDAPDisplayName` attribute of the `attributeSchema` object.
Name in admin tools	Fax Number	Any name that is descriptive and consistent	See the explanation in the text.

TABLE 8.4 Some Confusing Name Pairs

LDAP Name	Common Name
`info`	Comment
`comment`	User-Comment
`street`	Street-Address
`streetAddress`	Address

for "English (United States)"). Then open the properties for `CN=user-Display` and select the attribute `attributeDisplayNames`. It is a multivalued attribute that contains pairs of LDAP names and display names.

NOTE The list of name pairs is not likely to contain display names for all attributes. For names that are not on the list, each attribute schema object has an attribute `adminDisplayName`, which the administrative tools can use instead.

NOTE In reality, the Users and Computers snap-in doesn't use all the display specifier attributes (even though it is suggested here and in Microsoft's documentation). For example, most of the field names in user properties (such as Description or Office) are hard-coded in the tool instead of readable from the `attributeDisplayNames` attribute.

We use mainly LDAP names in this book for three reasons:

- If you use ADSI scripts, you need LDAP names.
- If you use LDAP filters, you need LDAP names.
- The Schema Manager snap-in, Replication Monitor, and many other administration tools use LDAP names.

Inspecting the Schema with the Schema Manager Snap-In

The Schema Manager snap-in is made specifically for viewing and managing the schema. Because Microsoft doesn't make it available to casual users, you have to take some extra steps to access it.

WARNING Just as with ADSI Edit, do not select OK or Apply in the Schema Manager snap-in dialog boxes. Always exit by clicking Cancel.

TIP If you want to run the Schema Manager snap-in in a workstation, you must first install the Admin Pak. Locate the file AdminPak.MSI in the I386 folder of a Windows 2000 Server CD. When you double-click that file, it will install the Admin Pak.

1. In the Run window (click the Start button and select Run) or at the command prompt, enter the command `regsvr32 schmmgmt.dll` and press Enter. You should get a message indicating that the registration was successful.
2. Start MMC (Microsoft Management Console) by typing `mmc` and press Enter (again, either in the Run window or at the command prompt).
3. In the Console menu, select Add/Remove Snap-in. In the dialog box that opens, click Add.
4. From the list of snap-ins, select Active Directory Schema, click Add, and click Close. Click OK to close the original dialog box.
5. To save what you just did, in Console menu select Save As, select the name and location of the .msc file that you want, and click Save. Next time you want to start the snap-in, double-click the .msc file you just saved.

Now you are ready to take a look at your schema from a slightly different view than what you saw with ADSI Edit. Figure 8.6 and Figure 8.7 illustrate this view.

Using the Schema Manager snap-in, you can also see the mandatory and optional attributes for any class, which would be quite cumbersome with ADSI Edit. Note that this time we don't mean the attributes of a `classSchema` object, such as `schemaIDGUID`, but the attributes that the instances of the class may use, such as `homeDirectory` or `givenName`.

In addition to the lists in the screen shots in Figure 8.6 and Figure 8.7, you can open a properties dialog box for any class or attribute. We show those dialog boxes and discuss them in the later sections "Classes" and "Attributes and Syntaxes."

NOTE You must select an attribute in the Attributes container to be able to open the properties dialog box for it. If you select the attribute from the list shown in Figure 8.7, you will see just one item, Help, when you right-click the attribute.

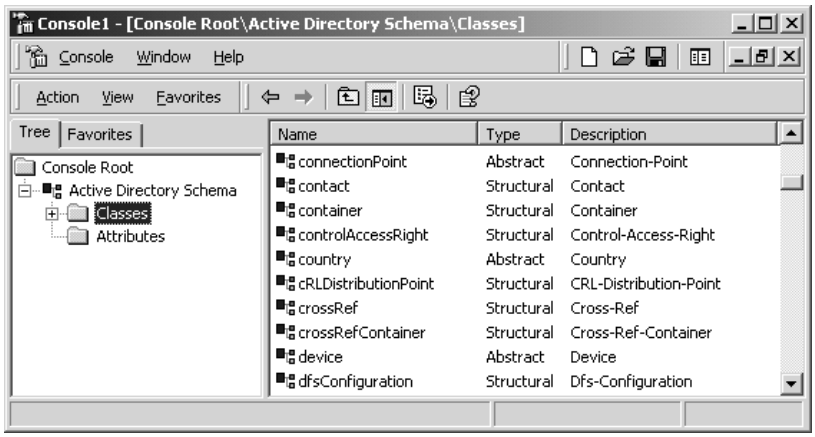


FIGURE 8.6 When you start the Schema Manager snap-in, it shows two containers: one for all the classes and one for attributes. Unlike with ADSI Edit, the objects are listed by their LDAP names.

Dumping the Schema to a Spreadsheet

Although the two graphical tools we just described are nice to use to explore the schema, they both have one problem: You can see the attributes for only one `classSchema` or `attributeSchema` object at a time.

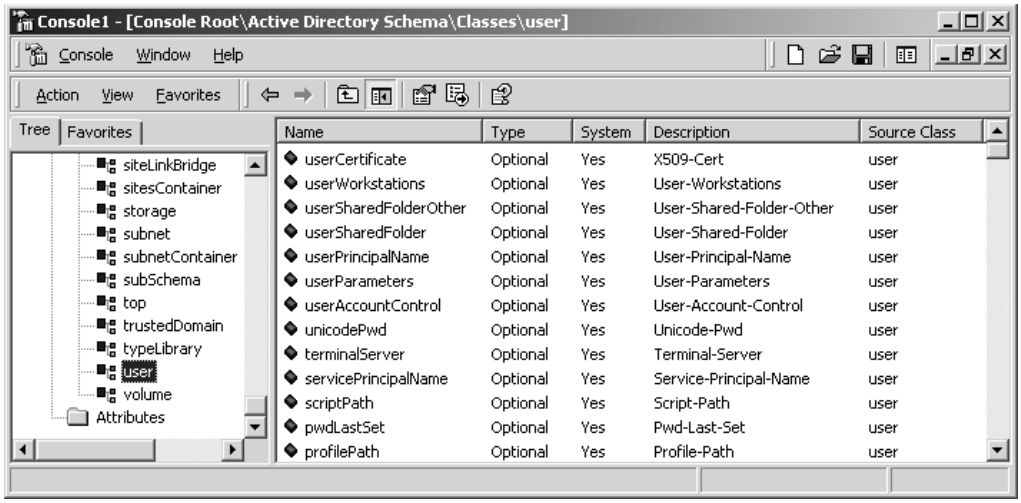


FIGURE 8.7 When you select a class in the left pane, the right pane shows the attributes for that class. Again, they are listed by their LDAP names. The text in the description column is usually the same as the common name you can see with ADSI Edit.

In this section we explain how you can dump all the information in your schema to a spreadsheet. It allows a broader view to the schema contents. Also, you can sort and filter the data, which gives you a better idea of how the various attributes are used. As you read on in this chapter, you can use the tables created here as a reference.

We use Excel in our explanation of how to dump the schema into a spreadsheet, but you can use other spreadsheet and database applications. First, you dump all your `classSchema` and `attributeSchema` objects into two text files, and then import those text files to Excel.

At a domain controller, you must type the following two commands at the command prompt (click the Start button and select Programs, Accessories, Command Prompt) and press Enter after each command:

```
C:\>csvde -f classes.txt -d cn=schema,cn=configuration,dc=sanao,dc=com
-r (objectCategory=classSchema)
C:\>csvde -f attributes.txt -d cn=schema,cn=configuration,dc=sanao,
dc=com -r (objectCategory=attributeSchema)
```

You should see output like the following for each command:

```
Connecting to "(null)"
Logging in as current user using SSPI
Exporting directory to file classes.txt
Searching for entries...
Writing out entries.............................
.............................................
Export Completed. Post-processing in progress...
142 entries exported
The command has completed successfully
```

Each of the two commands specifies an output file, a base distinguished name from which to dump, and an LDAP filter.

NOTE You can also use the CSVDE tool on a workstation. See its online help for more information.

Next you must launch Excel and perform the following steps for both of your text files:

1. Select File, Open.
2. Select one of your newly created text files and click Open. A text import wizard should start up.
3. Specify that your data is delimited (instead of fixed width).
4. Specify that the delimiter is a comma.

5. Click Finish to complete the wizard. You should now have about 30 columns of data, one column for each attribute.
6. Click cell A2 to activate it. Choose Freeze Panes in the Window menu. Now your first row with the column labels (i.e., attribute names) stays visible, even if you scroll down the sheet.
7. Adjust the width of each column as you like. You can also double-click the right border of each column header (F, G, H, and so on) to autosize the columns. Note that some of the data values may be longer than the width of your screen. If you cannot resize a wide column by dragging its right border with a mouse, you can use the Format menu option Column, Width.
8. If they are present, remove the columns `uSNChanged`, `uSNCreated`, `whenChanged`, `whenCreated`, `dITContentRules`, `extendedClassInfo`, `modifyTimeStamp`, and `extendedAttributeInfo`. You are not interested in them because they don't define schema characteristics.
9. Open the Data menu and select Filter, AutoFilter. This turns each column header into a drop-down list filter. When you open a list, you see all the distinct values for that column (i.e., attributes). If you select a value, your sheet will be filtered to show only the lines with that value.
10. Save the sheet in XLS format.

Figure 8.8 shows an Excel sheet that results from the preceding steps. We haven't sorted the lines—you might want to sort them by the `lDAPDisplayName` column (using the Sort feature in the Data menu of Excel).

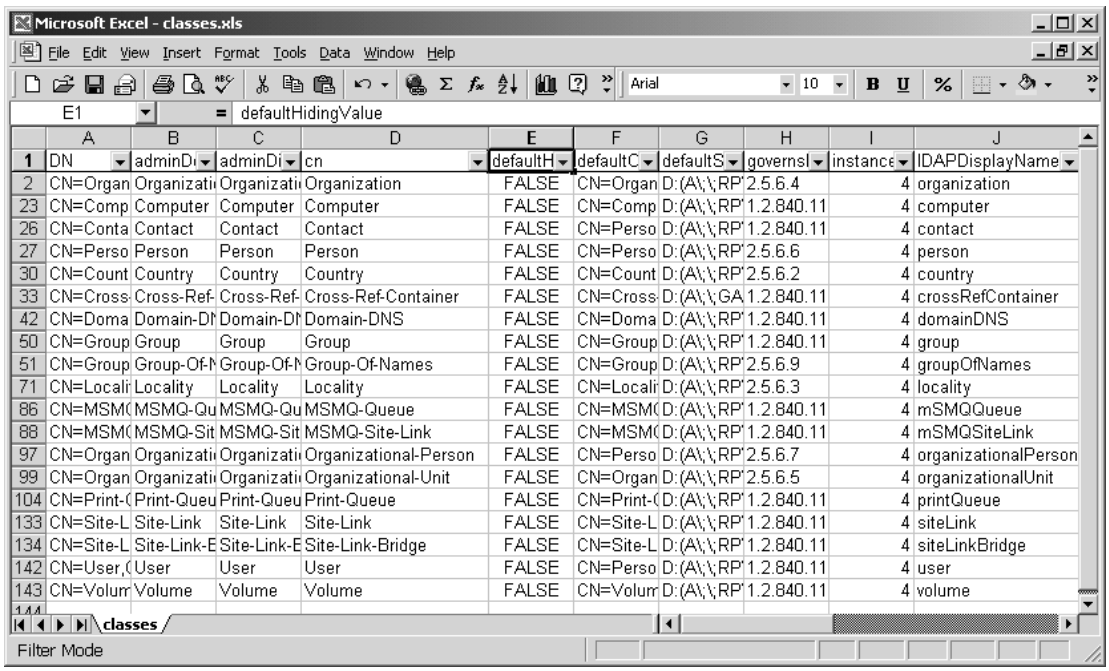
Subschema Subentry

Active Directory supports LDAPv3, which requires a directory service to expose its schema in a single `subSchema` object. Active Directory stores this object in the Schema container with the name `CN=Aggregate`.

The Aggregate object contains some multivalued attributes, which list the classes and attributes available in the schema. If you want to take a look, those attributes are as follows:

- `objectClasses`
- `attributeTypes`
- `extendedClassInfo`
- `extendedAttributeInfo`
- `dITContentRules`

In Chapter 10 and Chapter 11, we use the ADSI interface for directory access. When you specify a path such as `LDAP://sanao.com/schema` for ADSI, or just



	A	B	C	D	E	F	G	H	I	J
1	DN	adminDi	adminDi	cn	defaultH	defaultC	defaultS	governsI	instance	lDAPDisplayName
2	CN=Organ	Organizati	Organizati	Organization	FALSE	CN=Organ	D:(A\;\;RP	2.5.6.4	4	organization
23	CN=Comp	Computer	Computer	Computer	FALSE	CN=Comp	D:(A\;\;RP	1.2.840.11	4	computer
26	CN=Conta	Contact	Contact	Contact	FALSE	CN=Perso	D:(A\;\;RP	1.2.840.11	4	contact
27	CN=Perso	Person	Person	Person	FALSE	CN=Perso	D:(A\;\;RP	2.5.6.6	4	person
30	CN=Count	Country	Country	Country	FALSE	CN=Count	D:(A\;\;RP	2.5.6.2	4	country
33	CN=Cross	Cross-Ref	Cross-Ref	Cross-Ref-Container	FALSE	CN=Cross	D:(A\;\;GA	1.2.840.11	4	crossRefContainer
42	CN=Doma	Domain-DI	Domain-DI	Domain-DNS	FALSE	CN=Doma	D:(A\;\;RP	1.2.840.11	4	domainDNS
50	CN=Group	Group	Group	Group	FALSE	CN=Group	D:(A\;\;RP	1.2.840.11	4	group
51	CN=Group	Group-Of-N	Group-Of-N	Group-Of-Names	FALSE	CN=Group	D:(A\;\;RP	2.5.6.9	4	groupOfNames
71	CN=Locali	Locality	Locality	Locality	FALSE	CN=Locali	D:(A\;\;RP	2.5.6.3	4	locality
86	CN=MSMC	MSMQ-Qu	MSMQ-Qu	MSMQ-Queue	FALSE	CN=MSMC	D:(A\;\;RP	1.2.840.11	4	mSMQQueue
88	CN=MSMC	MSMQ-Sit	MSMQ-Sit	MSMQ-Site-Link	FALSE	CN=MSMC	D:(A\;\;RP	1.2.840.11	4	mSMQSiteLink
97	CN=Organ	Organizati	Organizati	Organizational-Person	FALSE	CN=Perso	D:(A\;\;RP	2.5.6.7	4	organizationalPerson
99	CN=Organ	Organizati	Organizati	Organizational-Unit	FALSE	CN=Organ	D:(A\;\;RP	2.5.6.5	4	organizationalUnit
104	CN=Print-(	Print-Queu	Print-Queu	Print-Queue	FALSE	CN=Print-(	D:(A\;\;RP	1.2.840.11	4	printQueue
133	CN=Site-L	Site-Link	Site-Link	Site-Link	FALSE	CN=Site-L	D:(A\;\;RP	1.2.840.11	4	siteLink
134	CN=Site-L	Site-Link-B	Site-Link-B	Site-Link-Bridge	FALSE	CN=Site-L	D:(A\;\;RP	1.2.840.11	4	siteLinkBridge
142	CN=User,(	User	User	User	FALSE	CN=Perso	D:(A\;\;RP	1.2.840.11	4	user
143	CN=Volum	Volume	Volume	Volume	FALSE	CN=Volum	D:(A\;\;RP	1.2.840.11	4	volume
144										

FIGURE 8.8 After you have all the class definitions in Excel, you can filter in just the classes you specify—for example, classes that have `defaultHidingValue = False`.

`LDAP://schema`, ADSI will expose the `subSchema` object as one container with 142 classes, 863 attributes, and 33 syntaxes under it. The properties for these "virtual objects" are more limited than those with the real Schema container, but they have some advantages for scripting.

NOTE Active Directory supports 23 syntaxes, as discussed later in this chapter. The ADSI interface is using 33 syntaxes because it must support other environments also—most notably, NetWare. The ADSI syntaxes are listed in Chapter 10.

Schema Cache

Because the schema guards the structure and content of Active Directory objects, it is needed every time any object (such as the user Jack Brown) is added or modified. Accessing the schema from the ntds.dit file would be too slow; therefore, every domain controller holds a copy of the schema in RAM. This copy is called the *schema cache.*

Internally the schema cache is not an identical copy of the bytes on disk; it is structured a little differently for easy and fast access.

Naturally the schema cache is built based on the information in the disk version each time the domain controller starts. If a change is made to the schema on the schema master, the change starts to replicate to other domain controllers, just as any change to Active Directory does. In each domain controller, the change goes first to the schema on disk and then, after a 5-minute delay, the schema cache is updated.

You cannot use most of the schema changes until they are in the schema cache of the domain controller you are using. Consequently, there is an additional 5-minute wait after the possible replication latency from the schema master to your domain controller. During this waiting period, applications continue to use the old schema cache.

The 5-minute delay here is for the same reason as with replication. After one change, usually more changes soon follow. Because each schema cache update consumes quite a few bytes of memory, it is more efficient to wait a little and make them all at once.

NOTE The 5-minute period is counted from the first change, and it doesn't reset if there is another change 1 minute later. Therefore, that latter change has to wait only 4 minutes to get into the schema cache.

The extra memory consumption in schema reload is due to the fact that when a new schema version is reloaded in cache, the old schema cache stays in memory for all the threads that were already running. Once all the old threads have exited, the older schema cache copy is released from memory. This also means that if you test schema changes, you probably need to restart your admin tools, so that they start to use the new schema cache and therefore can see the changes.

TRIGGERING THE SCHEMA CACHE UPDATE

If 5 minutes is too long for you to wait, you can trigger the cache update immediately with the Schema Manager snap-in. In the left pane, right-click the node Active Directory Schema and select Reload the Schema. There is also a programmatic way to trigger the schema cache update. You must write the value 1 to an attribute `schemaUpdateNow` residing in a special `rootDSE` virtual object. We cover this in later chapters:

- Chapter 9 contains an example of adding a new attribute and class to the schema and triggering the update using the LDIFDE tool.
- Chapter 10 describes the `rootDSE` object in more detail.
- Chapter 11 contains an example of adding a new attribute and class to the schema and triggering the update using the ADSI interface in a script.

Before we conclude our discussion about the schema in general, we'll mention the constructed attributes.

Constructed Attributes

Not all attributes for an object are stored in the schema on disk. Instead, they are built from other attributes. Twenty-two of the 863 attributes in the base schema are constructed.

The 22 constructed attributes are as follows:

```
primaryGroupToken
allowedAttributes
allowedAttributesEffective
allowedChildClasses
allowedChildClassesEffective
aNR
attributeTypes
canonicalName
createTimeStamp
dITContentRules
extendedAttributeInfo
extendedClassInfo
fromEntry
modifyTimeStamp
objectClasses
parentGUID
possibleInferiors
sDRightsEffective
subSchemaSubEntry
tokenGroups
tokenGroupsGlobalAndUniversal
tokenGroupsNoGCAcceptable
```

CLASSES

In the overview section of this chapter we introduced classes, and in the "Schema" section we mentioned that there is a class schema object for each class. In this section we examine classes and their schema objects in more detail.

Each class is defined by the attributes of its schema object. This means that examining classes actually means examining those attributes. Each `classSchema` object may contain at most 95 attributes. However, remember that many of these attributes are not related to defining the schema, but instead are "normal" attributes for any object. Examples of these normal attributes are `wWWHomePage` and `uSNChanged`. Here we discuss 36 attributes and omit 59 normal "uninteresting" attributes.

NOTE You could claim that some of the 36 attributes we address are also uninteresting. For example, they include 5 attributes that are not used at all. We include them so that you don't need to wonder what they are if you run into them with ADSI Edit or some other tool.

NOTE We list a few numbers of attributes in the text and in Table 8.5. The exact numbers are not important; we mention them to give you an idea of how many different kinds of attributes there are.

Classes can inherit from other classes attributes their instances must and may contain. Of the 95 `classSchema` attributes, 22 are the class' own attributes and 73 are inherited from the `top` class. In the list of 36 attributes in Table 8.5, 22 are the class' own attributes and 14 are inherited attributes.

We have divided the 36 attributes into four categories. These categories resemble those listed in the "Role of the Schema" section earlier in this chapter (see Table 8.1 and Table 8.2).

- *Names and identifiers.* There are 14 of these attributes (plus 1 naming-type attribute). Fortunately, not all of them are used.
- *Structure and containment rules.* These 7 attributes define where in the tree each object may be created (or instantiated) and which attribute values it must and may contain.
- *Class inheritance.* A class may inherit containment from another class. The 4 attributes in this category define how this happens.
- *Miscellaneous.* The remaining 10 attributes provide some defaults, for example, to the objects that will be created to the corresponding class.

Table 8.5 lists the 36 attributes by their LDAP names. If the common name has no resemblance to the LDAP name, the common name appears in parentheses

TABLE 8.5 Attributes of a `classSchema` Object

Name	Type	Source Class	Base Schema Classes (142)	`user` Class Example
Names and Identifiers				
`lDAPDisplayName`	O	`classS.`	All values are unique	`user`
`cn` (Common-Name)	M	`classS.`	All values are unique	User
`adminDisplayName`	O	`top`	Same as `cn`	User
`name` (RDN)	O	`top`	Same as `cn`	User
`distinguishedName` (Obj-Dist-Name)	O	`top`	All values are unique	`CN=User, CN=Schema, CN=Configuration, DC=sanao, DC=com`

(*continued*)

TABLE 8.5 (*continued*)

Name	Type	Source Class	Base Schema Classes (142)	user Class Example
canonicalName	O	top	All values are unique	sanao.com/ Configuration/ Schema/User
displayName	O	top	Not used	—
displayNamePrintable	O	top	Not used	—
classDisplayName	O	classS.	Not used	—
adminDescription	O	top	Same as cn	User
description	O	top	Not used	—
governsID	M	classS.	125 are Microsoft IDs (1.2.840.113556), 17 are X.500 IDs (2.5)	1.2.840.113556.1.5.9
schemaIDGUID	M	classS.	All values are unique	ba7a96bf e60dd011 a28500aa 003049e2
objectGUID	O	top	All values are unique	4856468b 62963047 8c32fcfe 70f6ca9b
rDNAttID	O	classS.	134 x cn, 4 x dc, 1 x c, 1 x l, 1 x o, 1 x ou	cn
Structure and Containment Rules				
mustContain	O	classS.	All are <not set>	—
systemMustContain	O	classS.	51 are set; 91 are <not set>	—
mayContain	O	classS.	All are <not set>	—
systemMayContain	O	classS.	116 are set; 26 are <not set>	A long list of attributes
possSuperiors	O	classS.	All are <not set>	—
systemPossSuperiors	O	classS.	134 are set; 8 are <not set>	builtinDomain + organizationalUnit + domainDNS
possibleInferiors	O	top	56 are set (=container); 86 are not (=leaf)	nTFRSSubscriptions + classStore

(*continued*)

TABLE 8.5 Attributes of a `classSchema` Object (*continued*)

Name	Type	Source Class	Base Schema Classes (142)	`user` Class Example
Class Inheritance				
`objectClassCategory`	M	`classS.`	124 are structural; 14 are abstract; 4 are auxiliary; none are 88-classes	1 (=structural)
`subClassOf`	M	`classS.`	89 are `top`; 53 are others	`organizationalPerson`
`auxiliaryClass`	O	`classS.`	All are <not set>	—
`systemAuxiliaryClass`	O	`classS.`	7 are set, 135 are <not set>	`securityPrincipal + mailRecipient`
Miscellaneous				
`nTSecurityDescriptor`	M	`top`	All are set	A DACL, SACL, and so on
`isDefunct`	O	`classS.`	All are <not set>	—
`defaultObject-Category`	M	`classS.`	Almost all are same as the class distinguished name	`CN=Person, CN=Schema, CN=Configuration, DC=sanao, DC=com`
`defaultHidingValue`	O	`classS.`	123 are True; 19 are False	False
`defaultSecurity-Descriptor`	O	`classS.`	138 are set; 4 are <not set>	Almost a 1,000-character-long string, starting with `D:(A;;RPWPCRCCDC`
`systemOnly`	O	`classS.`	8 are True; 134 are False	False
`systemFlags`	O	`top`	140 are set; 2 are <not set>	16
`objectClass`	M	`top`	All are `top+ classSchema`	`top + classSchema`
`objectCategory`	M	`top`	All are `CN=Class-Schema, CN=Schema, CN=Configuration, DC=sanao, DC=com`	`CN=Class-Schema, CN=Schema, CN=Configuration, DC=sanao, DC=com`
`schemaFlagsEx`	O	`classS.`	Not used	—

(such as "distinguishedName (Obj-Dist-Name)"). The Type column indicates whether the attribute is mandatory or optional. The Source Class column indicates either classSchema or top, the latter meaning the attribute was inherited from the top class. The Base Schema Classes column gives a summary of the kind of values the 142 classes have for the attributes in the table. Finally, the user Class Example column lists the values for the user class.

In the remaining subsections, we examine each of the four categories.

Names and Identifiers

The various names and identifiers identify the classes by both people and Active Directory. This category of 15 attributes includes nine different names, two descriptions, and three identifiers. The fifteenth attribute (rDNAttID) is none of these, but we include it in this category because it is name-related.

Table 8.6 describes the 15 attributes. Fortunately, Active Directory doesn't use quite all of them, as indicated in the table. Consequently, we are left with "only" six different names and one description. Furthermore, canonicalName, distinguishedName, and RDN are redundant with cn, because you can derive them directly from cn. This leaves us with three "nonredundant" names (the first three in Table 8.6) and one description.

Object Identifiers

As you see in Table 8.6, one of the many identifiers for each class schema object is an *object identifier (OID)*. An OID consists of numbers that have dots in between them, such as 1.3.6.1.4.1.123123. The number is hierarchical, so the first number in an OID is the highest level of the tree.

Anyone who "owns" a certain OID can allocate new child OIDs to it. For example, if you have the *base OID* 1.2.3, you could allocate to it OIDs, such as 1.2.3.1, 1.2.3.77, and 1.2.3.77.4.3.2.1.

Just like IP addresses, OIDs are administered globally so that no two organizations in the world can have the same base OID. OIDs allow unique identification of all kinds of things—they are used to identify classes, attributes, and syntaxes in X.500 directories and variables in Simple Network Management Protocol (SNMP), among other things.

Microsoft uses the base OID 1.2.840.113556.1 in many Active Directory classes (and attributes). Table 8.7 describes the history of that OID.

NOTE Not all class schema objects and attribute schema objects of the Active Directory base schema use the OIDs in Table 8.7. Some use OIDs defined in X.500 or some other source.

TABLE 8.6 Name and Identifier Attributes of a `classSchema` Object

Name	Syntax*	Multi-valued	Description	`user` Class Example
`lDAPDisplayName`	`Unicode string`	No	Name to identify the class in ADSI scripts, LDAP filters, many low-level admin tools, and internally with all LDAP access.	`user`
`cn` (Common-Name)	`Unicode string`	No	Actual name of the class schema object.	User
`adminDisplayName`	`Unicode string`	No	Name that admin tools can use as their display name if there isn't a name in the `Display-Specifiers` container. If `adminDisplayName` is also not specified, the system will use `cn` instead.	User
`name` (RDN)	`Unicode string`	No	Same as `cn`.	User
`distinguishedName` (Obj-Dist-Name)	`Distin-guished name`	No	"`cn` with a path," identifies the location of the object (which is always the Schema container).	`CN=User, CN=...`
`canonicalName`	`Unicode string`	Yes	"`cn` with a path," but using a different format from `distinguishedName`.	`sanao.com/Configuration/Schema/User`
`displayName`	`Unicode string`	No	Not used.	—
`displayName-Printable`	`Printable string`	No	Not used.	—
`classDisplayName`	`Unicode string`	Yes	Not used.	—
`adminDescription`	`Unicode string`	No	Descriptive text for the class for admin tools.	User

(*continued*)

TABLE 8.6 (*continued*)

Name	Syntax*	Multi-valued	Description	`user` Class Example
`description`	`Unicode string`	Yes	Not used.	—
`governsID`	`OID string`	No	Object ID (OID) of the class (see the section, "Object Identifiers").	1.2.840.113556.1.5.9
`schemaIDGUID`	`Octet string`	No	128-bit unique GUID to identify the class.	ba7a96bf e60dd011 a28500aa 003049e2
`objectGUID`	`Octet string`	No	128-bit unique GUID to identify this class schema object (as well as any object in Active Directory).	4856468b 62963047 8c32fcfe 70f6ca9b
`rDNAttID`**	`OID string`	No	Whether the class prefix is `cn=`, `ou=`, `dc=`, `c=`, `l=`, or `o=`. This attribute also specifies whether the naming attribute (i.e., RDN) of the instances of this class is cn, ou, or some of the other choices just listed.	`cn`

*We discuss the syntaxes in the "Attributes and Syntaxes" section.
**The `rDNAttID` attribute choices `c`, `l`, and `o` are used by the `country`, `locality`, and `organization` classes, respectively. These classes exist in Active Directory, but are not used in any way. Consequently, you won't see them unless you install Exchange 2000 or some other software that would use them.

Another base OID that Microsoft owns is 1.3.6.1.4.1.311. Each numeric part has a name, so we can express this same base OID as `iso.org.dod.internet.private.enterprise.microsoft`. "Org" in the name indicates an "identified organization" that ISO acknowledges and "dod" indicates the U.S. Department of Defense. These "private enterprise numbers" are currently assigned by the Internet Corporation for Assigned Names and Numbers (ICANN, `http://www.icann.org/`). In addition to assigning enterprise OID numbers, ICANN coordinates the assignment of Internet domain names and IP address numbers. You can see the current list of assigned private enterprise numbers at `ftp://ftp.isi.edu/in-notes/iana/assignments/enterprise-numbers`.

TABLE 8.7 Microsoft Active Directory OIDs

Value	Owner
1	ISO, which issued 1.2 to ANSI
1.2	ANSI, which issued 1.2.840 to USA
1.2.840	USA, which issued 1.2.840.113556 to Microsoft
1.2.840.113556	Microsoft, which issued 1.2.840.113556.1 to Active Directory
1.2.840.113556.1	Active Directory, where class schema objects use either 1.2.840.113556.1.3 or 1.2.840.113556.1.5, and attribute schema objects use either 1.2.840.113556.1.2 or 1.2.840.113556.1.4.

Recall that we discussed the X.500 standards in Chapter 1. OIDs have similar roots, as they were first defined in ITU-T X.208, which corresponds to ISO/IEC 8824. However, X.208 has been superseded by the ITU-T recommendation X.680 (12/97): "Abstract Syntax Notation One (ASN.1): Specification of basic notation."

In addition to the OID notation, X.680 specifies the following base OIDs:

- 0: ITU-T assigned.
- 1: ISO assigned.
- 2: Joint ISO/ITU-T assigned. Active Directory contains some classes and attributes from X.500, and they use the base OIDs 2.5.6 and 2.5.4, respectively. There is also one "2.5.20" class ("X.500 schema object class") and six "2.5" attributes other than 2.5.4.

You can inspect various assigned OIDs at `http://www.alvestrand.no/domen/objectid/top.html`.

OBTAINING A BASE OID

If your organization needs to add classes or attributes to the schema, it must obtain a base OID. This need could rise from two situations.

- You need to modify the Active Directory schema of your organization.
- You sell applications to other organizations, and those applications need to modify the schema of the customer Active Directory.

There are three ways to get a base OID.

- Run the OIDGEN utility included in the Windows 2000 Resource Kit. It generates one base OID for classes and another for attributes. A resulting OID could

be 1.2.840.113556.1.4.7000.233.28688.28684.8.96821.760998.1196228.1142349. You can use these OIDs in your tests, but for a production network you should register a base OID, as explained in the next two choices.

- Apply for a free private enterprise number from ICANN using the form at `http://www.isi.edu/cgi-bin/iana/enterprise.pl`. You should receive a number (1.3.6.1.4.1.something) in a few days by e-mail.
- Apply for a base OID from another issuing authority, perhaps for a fee. Many countries have a country-specific organization to issue OIDs. For organizations in United States, you can try ANSI and the Web page `http://web.ansi.org/public/services/reg_org.html`. (However, ANSI seems not to be very responsive. We sent e-mail to three people whose e-mail addresses appeared on the ANSI Web page regarding OIDs, but we have not received an answer from them in 6 months.)

Once you get the base OID, you should establish the policy regarding administration of the numbers in your organization, just like you probably have had to do with IP addresses. For example, you can dedicate the branch ".1" to new classes and ".2" to new attributes. Because OIDs are a general standard, you may also need the base OID for uses other than Active Directory.

Structure and Containment Rules

The main job of the schema is to establish the structure and content rules for Active Directory. This is done with the first six attributes described in Table 8.8. They consist of three pairs of a normal attribute and a system attribute (`mustContain` + `systemMustContain`, `mayContain` + `systemMayContain`, and `possSuperiors` + `systemPossSuperiors`).

When creating classes, administrators can set all six attributes. For existing classes, they can change only the three normal attributes, not their system counterparts. The base schema uses only the three system attributes; it does not use their normal counterparts. This way, Active Directory can protect the base schema definitions. You cannot remove base schema attributes from any base schema class, nor can you remove possible parent classes.

NOTE Even if an attribute is not a system attribute, it doesn't mean that you can change it freely. There are other restrictions to modifying the schema, which we discuss in the next chapter.

Because all seven attributes here are multivalued and use the `OID string` syntax, Table 8.8 does not include this information. Table 8.8 indicates if each

TABLE 8.8 Structure and Containment Attributes of a `classSchema` Object

Name	Constructed	System Only	Description	`user` Class Example
`mustContain`	No	No	Admin-changeable list of mandatory attributes for the instances of this class	—
`systemMust-Contain`	No	Yes	Same as previous, but only the Directory System Agent (DSA) can change the list after the class has been created	—
`mayContain`	No	No	Admin-changeable list of optional attributes for the instances of this class	—
`systemMay-Contain`	No	Yes	Same as previous, but only the DSA can change the list after the class has been created	A long list of attributes
`possSuperiors`	No	No	Admin-changeable list of possible parent containers for the instances of this class	—
`systemPoss-Superiors`	No	Yes	Same as previous, but only the DSA can change the list after the class has been created	`builtinDomain` + `organizational-Unit` + `domainDNS`
`possible-Inferiors`	Yes*	Yes	List of the classes that an instance of this class can contain instances of (i.e., possible child object types)	`nTFRS-Subscriptions` + `classStore`

*`PossibleInferiors` is a constructed attribute. It is calculated from the attributes `possSuperiors` and `systemPoss-Superiors` to kind of sum up the effect, but the value is not stored in the schema on disk.

attribute is constructed and if only the system, but not an administrator, can alter the value.

NOTE If you compare the constructed `possibleInferiors` attribute to the attributes on which it is based, they seem not to match. The reason is that `possibleInferiors` accounts for the effect of inheritance, as discussed in the next subsection.

Figure 8.9 illustrates the role of the structure and containment attributes. The `systemMayContain` attribute of a `classSchema` object lists some of the attributes (inheritance may list more) that the instances of that class may contain (i.e., optional attributes). `SystemPossSuperiors` specifies some of the possible parents that the instances may have, and `possibleInferiors` displays the list of possible children.

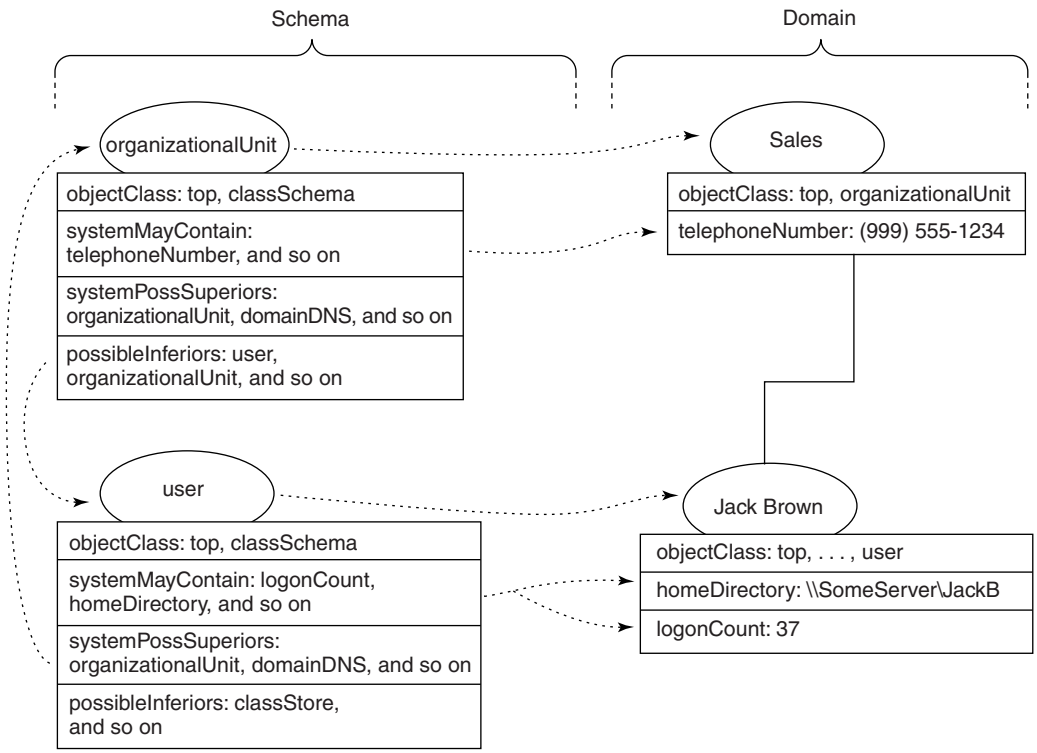


FIGURE 8.9 The main job of the schema is to establish the structure and content rules for Active Directory.

Four constructed attributes that are based on the attributes in Table 8.8 reside in the normal objects, such as users. Therefore, these attributes are not part of the schema, but still, you can study the schema by studying them.

- `AllowedChildClasses` and `allowedChildClassesEffective` both contain the same information as `possibleInferiors` (mentioned in Table 8.8), but they reside in normal objects, such as the user Jack Brown.
- `AllowedAttributes` and `allowedAttributesEffective` both contain a list of all mandatory and optional attributes of an object. For example, with Jack Brown, these two attributes both contain a list of 207 attributes, the number of mandatory and optional attributes of any `user` class object.

These four constructed attributes also take into account class inheritance.

Class Inheritance

A class gets most of its characteristics from the corresponding class schema object. However, a class inherits some features from several class schema objects, using *class inheritance.* We can also say that a class is *derived* from another class.

In this section we explain how, by means of inheritance, the list of possible superiors, mandatory attributes, and optional attributes are built from the information in several objects.

Class inheritance makes defining new classes easier, because they can build on existing classes. There is no need to list all possible superiors and attributes from the ground up. Table 8.9 lists the four attributes relevant to class inheritance.

USER CLASS EXAMPLE

Figure 8.10 illustrates the way the three lists (possible superiors, mandatory attributes, and optional attributes) are built up for the user class. We picked the user class as an example because it is by far the most intuitive.

The inheritance chain in Figure 8.10 (`top—person—organizational-Person—user`) has nothing to do with the hierarchy of the schema objects. They all exist as siblings in `CN=Schema`, `CN=Configuration`, `DC=sanao`, `DC=com`. Nor is there any connection with the normal directory hierarchy (domain—OU—user).

Figure 8.10 requires several explanatory comments.

- To save space, we abbreviated attribute names—`sMayContain` instead of `systemMayContain`, and so on.
- In addition to the system attributes shown in Figure 8.10, `mustContain`, `mayContain`, and `possSuperiors` also will inherit. This base schema example just doesn't happen to use them.

TABLE 8.9 Class Inheritance Attributes of a `classSchema` Object

Name	Syntax	Multi-valued	Description	`user` Class Example
`objectClass-Category`	`Enumeration`	No	1 = Structural 2 = Abstract 3 = Auxiliary 0 = 88-class	1 (=structural)
`subClassOf`	`OID string`	No	The class from which this class inherits containment and structure attributes. The parent is called a *superclass* and the child a *subclass.*	`organizational-Person`
`auxiliaryClass`	`OID string`	Yes	Admin-changeable list of auxiliary classes from which this class inherits containment attributes.	—
`system-AuxiliaryClass`	`OID string`	Yes	Same as previous, but only the Directory System Agent (DSA) can change the list.	`securityPrincipal + mailRecipient`

- `Top` has 69 optional attributes and `person` has 4 of its own. Still, the sum is 72 instead of 73 because there is 1 common attribute. For the same reason, the other sums may be a little less than you might expect.
- The two auxiliary classes `securityPrincipal` and `mailRecipient` are also subclasses of `top`. To keep the figure simple, and because this relationship has no effect, Figure 8.10 does not show this relationship.
- In Figure 8.10, only the `user` class happens to use auxiliary classes. Any superclass could also use auxiliary classes, in which case those attribute lists would also affect the subclasses.
- We show LDAP names (such as `mailRecipient`) as object names, even though to be precise the object names should be common names (such as Mail-Recipient). LDAP names appear because in this book we systematically call classes and attributes by their LDAP names.

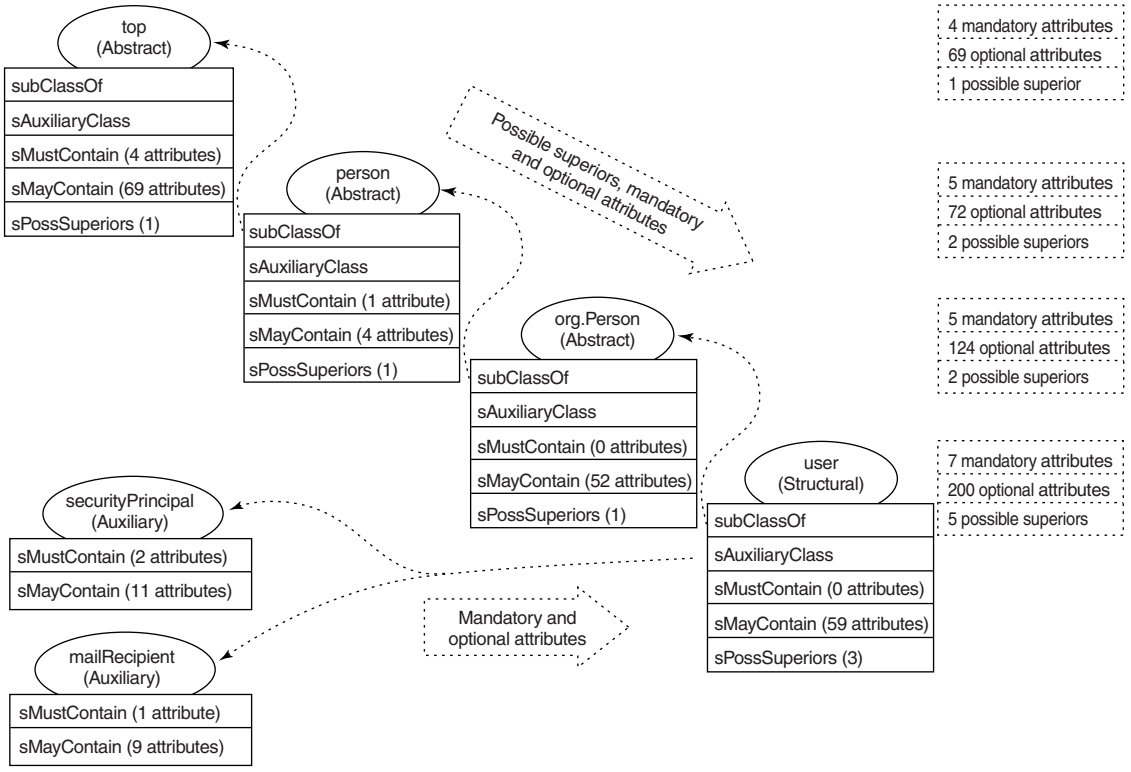


FIGURE 8.10 The list of possible superiors, mandatory attributes, and optional attributes for each class is the sum of its own list and the lists of all its superclasses. Also, auxiliary classes may add attributes to the list of mandatory and optional attributes.

CLASS CATEGORIES

Each class in Figure 8.10 was marked to be in one of three categories: *structural, abstract,* or *auxiliary.* Active Directory classes belong to four categories, as Table 8.10 describes. Figure 8.11 shows three categories of classes and their relationships as superclasses and subclasses.

Miscellaneous Characteristics of Classes

As the last set of class characteristics, we discuss ten miscellaneous attributes. Three of them provide defaults for objects to be created and four are quite general attributes that any Active Directory object will have. See Table 8.11 for details.

TABLE 8.10 Class Categories

Category	Number of Classes in the Base Schema	Can Be Instantiated*	Purpose and Comments
Structural	124	Yes	• Structural classes are the normal ones, because you can actually create objects for these classes. • Structural classes are derived from abstract or other structural classes and they can include auxiliary classes.
Abstract	14	No	• Abstract classes act as templates from which you can derive the actual structural classes or auxiliary and other abstract classes, if necessary.
Auxiliary	4	No	• Auxiliary classes just store lists of mandatory and optional attributes, which you can include in other classes. • Auxiliary class can be derived from another auxiliary class.
88-class	0	Yes	• The 1993 version of X.500 introduced the three previous categories. Any class created before that time belongs to a generic 88-class, which refers to the year 1988 when the previous version of the standard was approved. You shouldn't create any 88-class classes.

*Remember that *instantiation* means creating an object of some class. For example, when you create user Jack Brown, you just made an instance (or object) of the class `user`.

Table 8.11 introduces the following four concepts:

- `showInAdvancedViewOnly`
- Category 1 and 2 schema objects
- `objectCategory`
- Security Descriptor Definition Language (SDDL)

SHOWINADVANCEDVIEWONLY

All Active Directory objects have a `showInAdvancedViewOnly` attribute, which is derived from the `top` class. If the value of this attribute is set to True, an administrative snap-in can choose not to show the object in the user interface. If the attribute value is False or missing, an administrative snap-in should show the object.

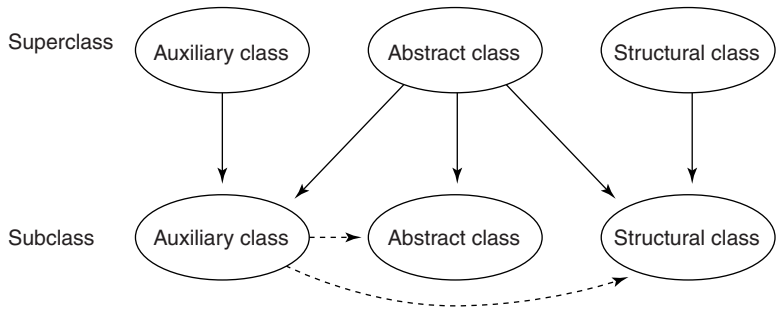


FIGURE 8.11 A class of any category can be inherited, or derived, from an abstract class. Only an auxiliary class can be inherited from another auxiliary class and only a structural class can be inherited from another structural class. In addition, the attribute lists in auxiliary classes can be used in abstract and structural classes (indicated by the dashed arrows).

Many objects are not interesting even to administrators, so setting this attribute helps to hide those objects. When you turn on Advanced Features in the Users and Computers snap-in, you make visible those objects that have the `showInAdvanced-ViewOnly` attribute set to True.

TABLE 8.11 Miscellaneous Attributes of a `classSchema` Object

Name	Syntax	Multi-valued	Description	`user` Class Example
`nTSecurity-Descriptor`	`NT security descriptor`	No	Security descriptor of the class schema object; allows Schema Admins to modify and other users to read schema objects.	A DACL, SACL, and so on
`isDefunct`	`Boolean`	No	If set to True, the class is disabled.	—
`defaultObject-Category`	`Distinguished name`	No	Default value for `objectCategory` attribute for each new instance of this class.	`CN=Person, CN=...`
`defaultHiding-Value`	`Boolean`	No	Default value for `showInAdvanced-ViewOnly` attribute for each new instance of this class.	False

(*continued*)

TABLE 8.11 (*continued*)

Name	Syntax	Multi-valued	Description	`user` Class Example
`defaultSecurity-Descriptor`	`Unicode string`	No	Default value for the security descriptor for each new instance of this class, using Security Descriptor Definition Language (SDDL). The program that will create the instance can also specify a security descriptor to replace this default.	Long string, starting with `D:(A;;RPWPCR`
`systemOnly`	`Boolean`	No	If set to True, only the Directory System Agent (DSA) can create and modify instances of this class.	False
`systemFlags`	`Integer`	No	Fifth bit from right (0x10) tells whether this class belongs to *category 1* (part of the base schema) or *category 2* (an extension).	16 (=0x10)
`objectClass`	`OID string`	Yes	Class of this class schema object itself, the value is obviously `classSchema` for all class schema objects (plus the superclass `top`—see discussion about object categories later in this chapter).	`top + classSchema`
`objectCategory`	`Distinguished name`	No	Object category of this class schema object itself, the value is the distinguished name of the object `CN=Class-Schema` for all class schema objects.	`CN=Class-Schema`, `CN=Schema`, `CN=...`
`schemaFlagsEx`	`Integer`	No	Not used	—

NOTE `CN=Dfs-Configuration` has the value False for this attribute. However, it doesn't show in the normal view of the Users and Computers snap-in because the parent container, `CN=System`, has a value True. Once you turn on Advanced Features, they will both show.

CATEGORY 1 AND 2 SCHEMA OBJECTS

One bit in the `systemFlags` attribute tells whether the class belongs to category 1 or 2. *Category 1* classes and attributes are part of the base schema, and *category 2* schema objects are something that you or your application have added. In other words, category 2 schema objects are part of the schema extensions.

The reason for the two categories is that there can be (and there are) stricter rules for modifications of category 1 schema objects.

OBJECT CATEGORY

You have seen the word "category" used in more than one context in this chapter. Here is one more: As you know, each object belongs to some class—to be exact, it also belongs to all superclasses of the main class. For example, the object Jack Brown belongs to the classes `user`, `person`, `organizationalPerson`, and `top`. To be able to contain this list, the `objectClass` attribute is multivalued.

Each object belongs also to some *object category,* as expressed with a single-valued attribute `objectCategory`. The object category is usually the same as the class, but it may be different, most likely one of the superclasses. For example, the `objectCategory` of most `user` class objects is `person`, which is two steps up in the class hierarchy.

As you can see in Table 8.11, the schema defines only the `defaultObjectCategory` of each class, but the actual `objectCategory` attribute is per object. Consequently, two objects of the same class could belong in theory to two categories. However, this would only make things more confusing when you used the category as a search criterion. Also, Active Directory doesn't allow changing the `defaultObjectCategory` of any base schema object or the `objectCategory` of any existing object, which fortunately makes this confusing situation difficult to achieve.

Using object categories in LDAP filters and queries has the following advantages over using object classes.

- `ObjectCategory` is an indexed attribute, whereas `objectClass` is not. Therefore, LDAP filters and queries that use `objectCategory` are much faster than those that use just `objectClass`.

- Objects of several classes can use the same category. If you use a filter or query `objectCategory=person`, you will get a list of all people, regardless of whether they are users or contacts. Apart from this "`person = user + contact`," however, the base schema doesn't contain practical examples of several classes using the same object category.

SECURITY DESCRIPTOR DEFINITION LANGUAGE

Microsoft uses Security Descriptor Definition Language (SDDL) whenever a security descriptor should be described in a string format. Two examples of this are as follows:

- The `defaultSecurityDescriptor` attribute, mentioned in Table 8.11
- Security templates, which are INF files in C:\Winnt\Security\Templates

We explain SDDL briefly here by interpreting the `defaultSecurityDescriptor` for the `group` class (i.e., for each new group that you create). We don't use the `user` class as an example because the string would be ten times longer. For a detailed description of SDDL, refer to the Microsoft Platform SDK (software development kit) at `http://msdn.microsoft.com/library`.

The SDDL string for the `group` class is

```
D:
(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;DA)
(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;SY)
(A;;RPLCLORC;;;AU)
(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;AO)
(A;;RPLCLORC;;;PS)
(OA;;CR;ab721a55-1e2f-11d0-9819-00aa0040529b;;AU)
```

NOTE The string is divided here on seven lines to make it easier to read. In reality, it is one long string.

To interpret the string, you must be familiar with the contents of a security descriptor, as discussed in Chapter 4.

The `D:` in this string means that the subsequent data is a discretionary ACL. The owner, group, and system ACL are not included in this case.

The rest of the string consists of six pairs of parentheses, each of which is one ACE. The ACE in turn consists mostly of two-letter acronyms specifying who has what permissions.

The semicolons divide each ACE into six fields:

- ACE type
- ACE flags

- Permissions
- Object GUID
- Inherit object GUID
- Account SID

Table 8.12 shows the interpretation of the six ACEs. You can compare its elements to the SDDL string. Table 8.13 lists the SDDL acronyms, their spelled-out names, and their corresponding permission names in the user interface (i.e., in ACL Editor).

ClassSchema Object Property Pages

We conclude our discussion on classes by showing the Schema Manager snap-in property pages for a `classSchema` object. We picked (once again) the `user` class as an example.

TABLE 8.12 Default ACEs for a Group Object

ACE Type	Permissions*	Object GUID	Account SID
Access allowed	Read Prop, Write Prop, Control Access, Create Child, Delete Child, List Children, List Object, Read Control, Write Owner, Write DAC, Standard Delete, Delete Tree, Self Write		Domain Admins
Access allowed	Read Prop, Write Prop, Control Access, Create Child, Delete Child, List Children, List Object, Read Control, Write Owner, Write DAC, Standard Delete, Delete Tree, Self Write		System
Access allowed	Read Prop, List Children, List Object, Read Control		Authenticated Users
Access allowed	Read Prop, Write Prop, Control Access, Create Child, Delete Child, List Children, List Object, Read Control, Write Owner, Write DAC, Standard Delete, Delete Tree, Self Write		Account Operators
Access allowed	Read Prop, List Children, List Object, Read Control		(Personal) Self
Object access allowed	Control Access (i.e., extended rights)	Send to	Authenticated Users

*Domain Admins, System, and Account Operators have a list of 13 permissions. That list is the same as Full Control in the user interface. Authenticated Users and Self have a list of four permissions. The list is equal to the standard permission Read + special permission List Object.

TABLE 8.13 SDDL Permissions

AccessMask Bit Name	SDDL Acronym	SDDL Spelled-Out Name	Name in the User Interface
ADS_RIGHT_DS_CREATE_CHILD	CC	Create Child	Create All Child Objects
ADS_RIGHT_DS_DELETE_CHILD	DC	Delete Child	Delete All Child Objects
ADS_RIGHT_ACTRL_DS_LIST	LC	List Children	List Contents
ADS_RIGHT_DS_SELF	SW	Self Write	All Validated Writes
ADS_RIGHT_DS_READ_PROP	RP	Read Prop	Read All Properties
ADS_RIGHT_DS_WRITE_PROP	WP	Write Prop	Write All Properties
ADS_RIGHT_DS_DELETE_TREE	DT	Delete Tree	Delete Subtree
ADS_RIGHT_DS_LIST_OBJECT	LO	List Object	List Object
ADS_RIGHT_DS_CONTROL_ACCESS	CR	Control Access	All Extended Rights
ADS_RIGHT_DELETE	SD	Standard Delete	Delete
ADS_RIGHT_READ_CONTROL	RC	Read Control	Read Permissions
ADS_RIGHT_WRITE_DAC	WD	Write DAC	Modify Permissions
ADS_RIGHT_WRITE_OWNER	WO	Write Owner	Modify Owner

We show three screen shots (Figures 8.12 through 8.14)—one for each property page. We exclude the Security tab because it is a normal access control screen. For each screen, we list the attributes that correspond to the fields in the screen.

In the course of this section we have discussed 38 attributes of a class. The screens in Figures 8.12 through 8.14 show the values for only 12 of them (13 if you count the Security tab) corresponding to the `nTSecurityDescriptor` attribute. To see the remaining attributes, you need to use ADSI Edit.

Figure 8.12 shows the General tab. The corresponding attributes from top to bottom are as follows (the values are shown in parentheses):

- `lDAPDisplayName` ("user")
- `adminDescription` ("User")
- `cn` ("User")
- `governsID` ("1.2.840.113556.1.5.9")
- `objectClassCategory` ("Structural")

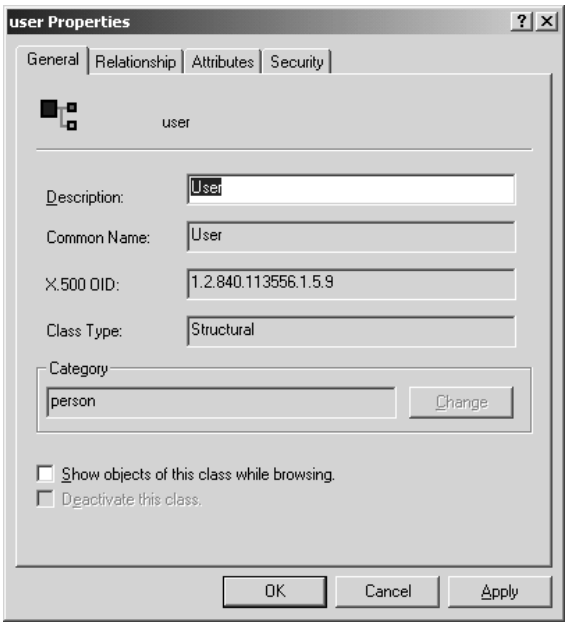


FIGURE 8.12 With the Schema Manager snap-in you can see the attributes of various classes, such as the `user` class.

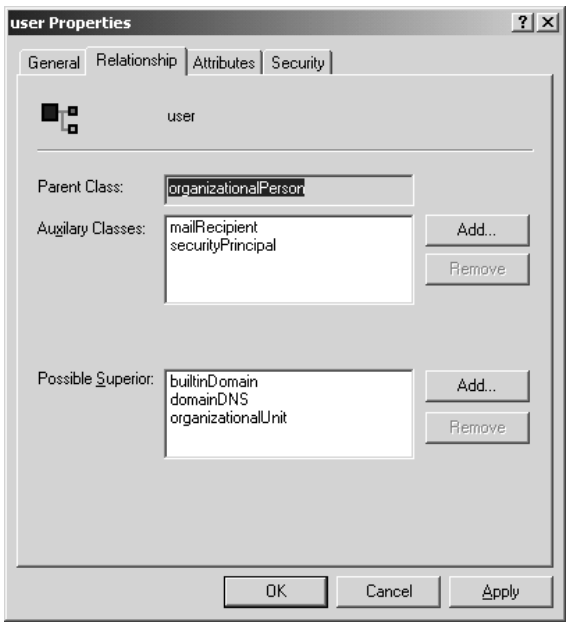


FIGURE 8.13 The Relationship tab shows both the inheritance hierarchy in the schema and the possible superiors in the normal directory tree.

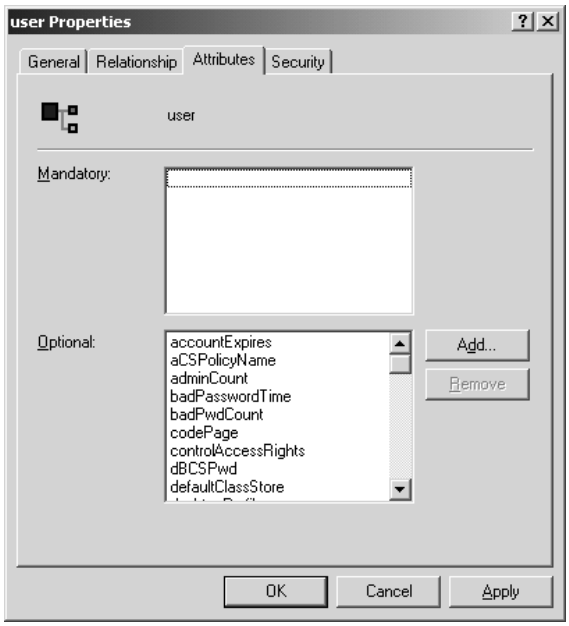


FIGURE 8.14 The Attributes tab lists the mandatory and optional attributes for the class, excluding inherited attributes.

- `defaultObjectCategory` ("person")
- `showInAdvancedViewOnly` (Note that the snap-in shipping with the original Windows 2000 has a bug so that setting this check box doesn't have the desired effect. The check box should control `defaultHidingValue`, which in turn controls the `showInAdvancedViewOnly` attribute of the objects to be created.)
- `isDefunct` ("not set")

Figure 8.13 shows the Relationship tab. The corresponding attributes from top to bottom are as follows (the values are shown in parentheses):

- `subClassOf` ("organizationalPerson")
- `auxiliaryClass`, `systemAuxiliaryClass` ("mailRecipient, securityPrincipal")
- `possSuperiors`, `systemPossSuperiors` ("builtinDomain, domainDNS, organizationalUnit")

Figure 8.14 shows the Attributes tab. The corresponding attributes from top to bottom are as follows (the values are shown in parentheses):

- `mustContain`, `systemMustContain` (empty list)
- `mayContain`, `systemMayContain` (a number of attributes)

Now that we have covered the different aspects of schema classes, we are ready to move on to the attributes and syntaxes. Attribute characteristics have something in common with class characteristics, but obviously there are also quite a few differences.

ATTRIBUTES AND SYNTAXES

In this section we discuss the attributes that define the attributes in the schema.

As with classes, there is a schema object for each attribute, which means a total of 863 `attributeSchema` objects in the base schema. Each of these 863 attributes is characterized by the attributes of its schema object and those `attributeSchema` objects themselves use 94 attributes out of the 863. Figure 8.15 shows the relationship between the attribute schema objects and their attributes.

NOTE The other figures that we have included in this chapter show LDAP names as schema object names (although this is imprecise). In Figure 8.15 we show the common names (which is more correct), because we want to show the actual object name and the LDAP name attribute separately.

Of the 94 attributes that the `attributeSchema` class uses, it inherits 73 attributes from `top` and defines 21 on its own level. As with classes, most of the inherited attributes are quite general (after all, they apply to any Active Directory object). We cover only 13 of the inherited attributes and all 21 "own" attributes for a total of 34 attributes.

NOTE When discussing classes, we listed 14 inherited attributes and now 13. The list is the same except that `possibleInferiors` doesn't apply here. For example, the `homePhone` of a user cannot have child objects.

We divide the 34 attributes into four categories.

- *Names and identifiers.* You probably remember that `classSchema` objects have 14 various names, descriptions, and identifiers. `attributeSchema` objects have even more: 17 names, descriptions, and identifiers.
- *Syntax and content rules.* These 7 attributes define the kind of data the attribute accepts and the values that are possible for it.
- *Searches.* The 2 attributes in this category control things such as indexing of the attribute and whether it is part of the global catalog.
- *Miscellaneous.* There are 8 miscellaneous attributes.

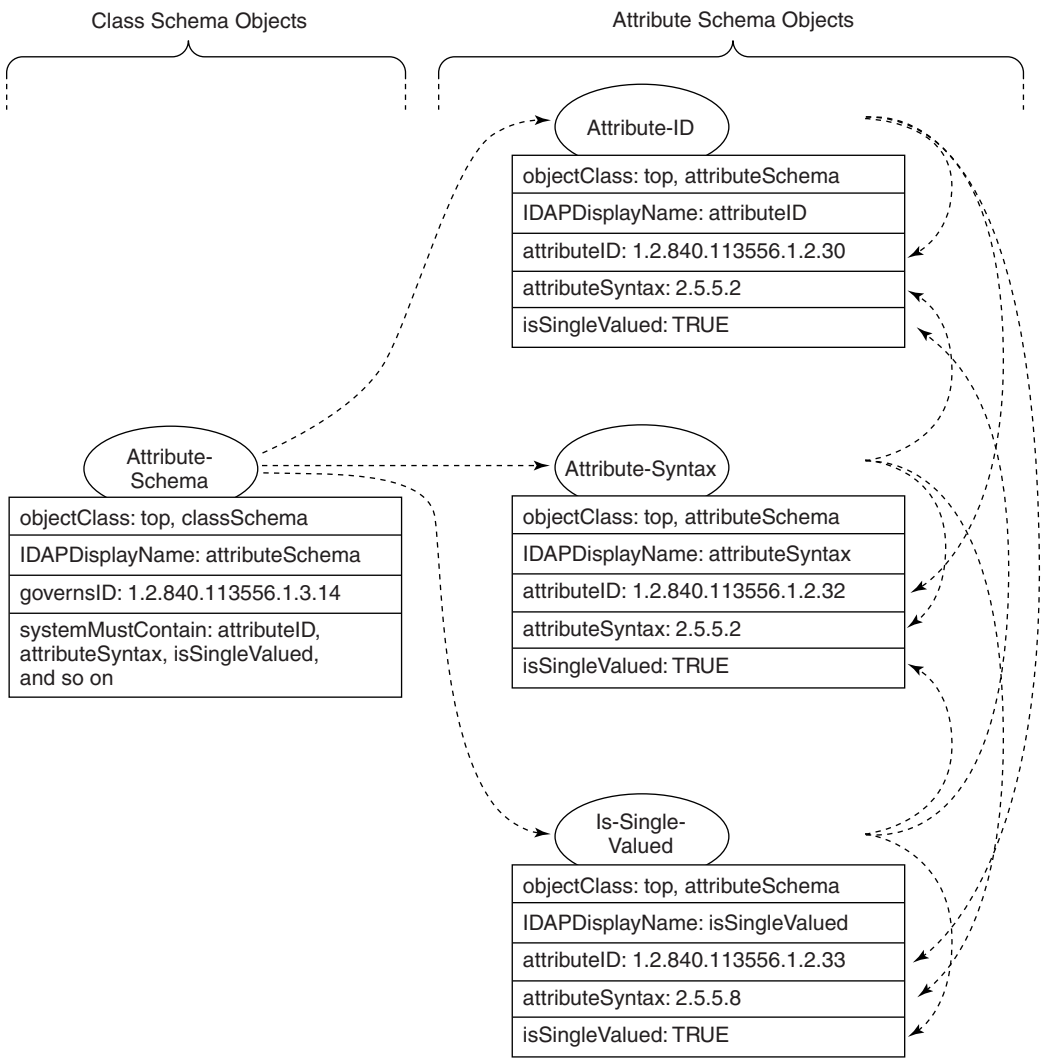


FIGURE 8.15 The `attributeSchema` class has 863 instances (i.e., `attributeSchema` objects) in the base schema. Each of these 863 objects defines the syntax and other aspects of one attribute. The definitions reside in the attributes of the objects, which means that the `attributeSchema` objects define attributes also for themselves. This figure shows three `attributeSchema` objects, which result in 3 × 3 arrows on the right side.

Table 8.14 shows all 34 attributes listed by their LDAP names. If the common name is quite different from the LDAP name, it appears in parentheses. The four sections following Table 8.14 describe the attributes in more detail. The Type column indicates whether the attribute is mandatory or optional. The Source Class column contains either `attributeSchema` or `top`—the latter meaning the

TABLE 8.14 Attributes of an `attributeSchema` Object

Name	Type	Source Class	Base Schema Attributes (863)	LDAP Display Name Attribute Example
Names and Identifiers				
`lDAPDisplayName`	M	`attr.S.`	All values are unique	`lDAPDisplayName`
`cn` (Common-Name)	M	`attr.S.`	All values are unique	LDAP-Display-Name
`adminDisplayName`	O	`top`	Same as `cn`	LDAP-Display-Name
`name` (RDN)	O	`top`	Same as `cn`	LDAP-Display-Name
`distinguishedName` (Obj-Dist-Name)	O	`top`	All values are unique	`CN=LDAP-Display-Name`, `CN=Schema`, `CN=Configuration`, `DC=sanao`, `DC=com`
`canonicalName`	O	`top`	All values are unique	`sanao.com/Configuration/Schema/LDAP-Display-Name`
`displayName`	O	`top`	Not used	—
`displayName-Printable`	O	`top`	Not used	—
`classDisplayName`	O	`attr.S.`	Not used	—
`adminDescription`	O	`top`	Same as `cn`	LDAP-Display-Name
`description`	O	`top`	Not used	—
`attributeID`	M	`attr.S.`	801 are Microsoft IDs (1.2.840.113556), 51 are X.500 IDs (2.5), 4 are Netscape IDs (2.16.840.1.113730), and 7 are "ITU-T data - PSS" IDs (0.9.2342)	1.2.840.113556.1.2.460
`schemaIDGUID`	M	`attr.S.`	All values are unique	9a7996bf e60dd011 a28500aa 003049e2
`objectGUID`	O	`top`	All values are unique	22a9624f 4b998f46 a658e3c2 7a357859
`attributeSecurityGUID`	O	`attr.S.`	128 are set; 735 are <not set>	—

(*continued*)

TABLE 8.14 (*continued*)

Name	Type	Source Class	Base Schema Attributes (863)	LDAP Display Name Attribute Example
mAPIID	O	attr.S.	70 are unique values; 793 are <not set>	33137
linkID	O	attr.S.	32 are unique values; 831 are <not set>	—
Syntax and Content Rules				
attributeSyntax	M	attr.S.	All are set to 1 of 15 values	2.5.5.12
oMSyntax	M	attr.S.	99 are set to "127"; 764 are set to 1 of 14 values	64
oMObjectClass	O	attr.S.	99 are set; 764 are <not set>	—
isSingleValued	M	attr.S.	620 are True; 243 are False	True
rangeLower	O	attr.S.	163 are set; 700 are <not set>	1
rangeUpper	O	attr.S.	162 are set; 701 are <not set>	256
extendedChars-Allowed	O	attr.S.	All are <not set>	—
Searches				
searchFlags	O	attr.S.	106 are nonzero; the rest 757 are zero	9
isMemberOf-Partial-AttributeSet	O	attr.S.	138 are True; 725 are <not set>	True
Miscellaneous				
nTSecurity-Descriptor	M	top	All are set	A DACL, SACL, and so on
isDefunct	O	attr.S.	All are <not set>	—
systemOnly	O	attr.S.	93 are True; 770 are False	False

(*continued*)

TABLE 8.14 Attributes of an `attributeSchema` Object (*continued*)

Name	Type	Source Class	Base Schema Attributes (863)	LDAP Display Name Attribute Example
Miscellaneous (continued)				
systemFlags	O	top	849 are set; 14 are <not set>	16
objectClass	M	top	All are top + attributeSchema	top + attributeSchema
objectCategory	M	top	All are CN=Attribute-Schema, CN=Schema, CN=Configuration, DC=sanao, DC=com	CN=Attribute-Schema, CN=Schema, CN=Configuration, DC=sanao, DC=com
schemaFlagsEx	O	attr.S.	Not used	—
isEphemeral	O	attr.S.	Not used	—

attribute was inherited from the `top` class. The Base Schema Attributes column indicates the kinds of values the 863 attribute objects have for the attributes in the table. Finally, the LDAP Display Name Attribute Example column lists the values for the `lDAPDisplayName` attribute.

NOTE As a detail, `lDAPDisplayName` is a mandatory attribute of an `attributeSchema` object, but it is only an optional attribute of a `classSchema` object.

Names and Identifiers

Most of the names, descriptions, and identifiers for attribute schema objects are exactly the same as names, descriptions, and identifiers for class schema objects. Consequently, there is no need to repeat the descriptions of the following nine attributes:

- `lDAPDisplayName`
- `cn` (Common-Name)
- `adminDisplayName`
- `name` (RDN)
- `distinguishedName` (Obj-Dist-Name)
- `canonicalName`

TABLE 8.15 Some Name and Identifier Attributes of an `attributeSchema` Object

Name	Syntax	Multi-valued	Description	LDAP Display Name Attribute Example
`attributeID`	`OID string`	No	Object ID (OID) of the attribute.	1.2.840.113556.1.2.460
`attributeSecurityGUID`	`Octet string`	No	ID that links the attribute to belong to some property set; permissions may then be given for this property set (see Chapter 6).	—
`mAPIID`	`Integer`	No	Messaging API (MAPI) applications identify attributes with this ID. Note that only 70 attributes have the `mAPIID` attribute set.	33137
`linkID`	`Integer`	No	Some attributes form forward-back link pairs, as discussed in the "Linked Attributes" section.	—

- `adminDescription`
- `schemaIDGUID`
- `objectGUID`

Also, the following four attributes are not used, as they were not used with class schema objects:

- `displayName`
- `displayNamePrintable`
- `classDisplayName`
- `description`

This leaves us with four not-yet-familiar attributes. They are described in Table 8.15.

LINKED ATTRIBUTES

When an attribute refers to another object in the directory, it is often beneficial that the target object has a reference to the first object. An example is a user's

membership in a group. The group has a member attribute, which includes the user (a forward link) and the user has a `memberOf` attribute, which includes the group (a back link). Figure 8.16 shows an example.

An even value in the `linkID` attribute denotes a forward link and a larger-by-1 odd value denotes a back link.

The base schema contains the following 13 linked attribute pairs:

- `member`—`memberOf` (Is-Member-Of-DL)
- `manager`—`directReports`
- `siteObject`—`siteObjectBL`
- `nonSecurityMember`—`nonSecurityMemberBL`
- `queryPolicyObject`—`queryPolicyBL`
- `privilegeHolder`—`isPrivilegeHolder`
- `managedBy`—`managedObjects`
- `hasMasterNCs`—`masteredBy`
- `serverReference`—`serverReferenceBL`
- `bridgeheadTransportList`—`bridgeheadServerListBL`
- `netbootServer`—`netbootSCPBL`
- `frsComputerReference`—`frsComputerReferenceBL`
- `fRSMemberReference`—`fRSMemberReferenceBL`

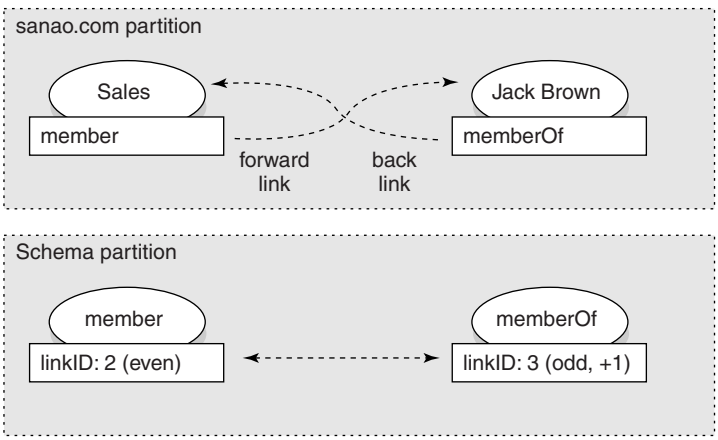


FIGURE 8.16 The `member` attribute is a forward link from a group object to a user object. The `memberOf` attribute is a back link from the user to the group. The relationship is defined in the corresponding `attributeSchema` objects.

NOTE If you count the attributes that have `linkID` defined, the number is not 2 × 13 = 26, but 32 instead. The reason is that the base schema includes 6 attributes that have `linkID` defined, but which are missing the corresponding back-link pair.

All back-link attributes, such as `memberOf`, are "system only"—that is, users or administrators cannot modify them. Active Directory is responsible for updating these attributes, maintaining *referential integrity* in the process. If either of the referenced objects is moved, Active Directory modifies the reference accordingly.

A forward-link attribute must use one of the following syntaxes: `DN`, `DN with Unicode string`, `DN with binary`, `access point DN`, and `OR name`. A back-link attribute must be of syntax `DN`.

Syntax and Content Rules

Seven `attributeSchema` attributes control the type of data and the values each attribute accepts. Table 8.16 describes these syntax and content attributes.

The data type is called syntax in this context and it is defined by three attributes: `attributeSyntax`, `oMSyntax`, and `oMObjectClass`. Naturally, the three attributes must be consistent, as you see in Tables 8.17 through 8.20 where we describe all the syntax choices.

Several attributes are necessary to express the syntax because more than one standard is involved. The first attribute (`attributeSyntax`) defines an X.500 syntax, and the two others a XOM syntax. "XOM" stands for "XAPIA X/Open Object Management," an interface for ASN.1 messaging defined by X.400 API Association (XAPIA) and X/Open (a former vendor standards organization). Currently, "X/Open" is a trademark that belongs to Open Group (`http://www.opengroup.org/`).

NOTE In "XAPIA," the *X* refers to X.400 and X.500 ITU-T standards, whereas in "X/Open," the *X* refers to UNIX.

NOTE The letters *XOM* can also stand for "X/Open OSI-Abstract-Data Manipulation."

SYNTAX CHOICES

Active Directory supports 23 syntaxes, but it uses only 19 of them in the base schema. Because the syntax choices are hard-coded and don't appear as objects in Active Directory, you cannot add syntaxes.

TABLE 8.16 Syntax and Content Attributes of an `attributeSchema` Object

Name	Syntax	Multi-valued	Description	LDAP Display Name Attribute Example
`attributeSyntax`	`OID string`	No	Identifies with an X.500 OID whether the value of this attribute is integer, string, or some other data format.	2.5.5.12
`oMSyntax`	`Integer`	No	About the same as `attributeSyntax` but expressed with a XOM code.	64
`oMObjectClass`	`Octet string`	No	If the XOM code is 127 (=object distinguished name), this attribute defines the "subsyntax."	—
`isSingleValued`	`Boolean`	No	Specifies whether the attribute is single- or multivalued.	True
`rangeLower`	`Integer`	No	The lowest possible value for an integer attribute or the shortest possible value for a string attribute.	1
`rangeUpper`	`Integer`	No	The largest possible value for an integer attribute or the longest possible value for a string attribute.	256
`extendedCharsAllowed`	`Boolean`	No	Specifies whether the attribute can contain extended characters; used only with the syntax `Teletex string`.	—

We can divide the syntaxes into the following categories:

- Simple data types (4 syntaxes)
- String data types (10 syntaxes)
- Time data types (2 syntaxes), which are actually also strings
- Reference data types (7 syntaxes), which are various object references

Tables 8.17 through 8.20 describe the syntaxes of each category. Some syntaxes have two names, in which case we mention the second name in parentheses.

TABLE 8.17 Syntaxes for Simple Data Types

Syntax	attributeSyntax	oMSyntax	Description	Count in Base Schema
Boolean	2.5.5.8	1	Values can be either TRUE or FALSE.	54
Integer	2.5.5.9	2	32-bit number.	149
Enumeration	2.5.5.9	10	32-bit number. Active Directory treats this as an integer.	6
Large integer (INTEGER8)	2.5.5.16	65	64-bit number.	66

TABLE 8.18 Syntaxes for String Data Types

Syntax	attributeSyntax	oMSyntax	Description	Count in Base Schema
OID string	2.5.5.2	6	An object ID string (e.g., 2.5.5.2) consisting of digits (0–9) and dots	20
Case-sensitive string (case-exact string)	2.5.5.3	27	Case-sensitive* string, each character of which belongs to the General-String character set**	0
Case-ignore string (teletex)	2.5.5.4	20	Case-insensitive string, each character of which belongs to the teletex character set**	8
Printable string	2.5.5.5	19	Case-sensitive string, each character of which belongs to the Printable character set**	13
IA5 string	2.5.5.5	22	Case-sensitive string, each character of which belongs to the *International Alphabet 5* (IA5) character set**	7
Numeric string	2.5.5.6	18	String, each character of which is a digit**	2
Octet string	2.5.5.10	4	Array of bytes (i.e., binary data)	120

(*continued*)

TABLE 8.18 Syntaxes for String Data Types (*continued*)

Syntax	attributeSyntax	oMSyntax	Description	Count in Base Schema
Unicode string (directory string)	2.5.5.12	64	Normal case-insensitive string using any Unicode characters	298
NT security descriptor	2.5.5.15	66	An octet string that contains a Windows NT/2000 security descriptor (SD)	3
SID string	2.5.5.17	4	An octet string that contains a Windows NT/2000 security identifier (SID)	8

*Whether a string is case-sensitive or case-insensitive matters when you start to use LDAP filters to specify which values match certain criteria.
**Active Directory doesn't currently enforce General-String, teletex, Printable, or IA5 character sets, or the digit restriction. However, this may change, so you should use only valid characters.

TABLE 8.19 Syntaxes for Time Data Types

Syntax	attributeSyntax	oMSyntax	Description	Count in Base Schema
UTC time string	2.5.5.11	23	Time-string format defined by ASN.1 standards. See standards ISO 8601 and X.680 for more information.* UTC, or Coordinated Universal Time, is roughly the same as GMT, or Greenwich Mean Time. This syntax uses only two characters to represent the year.	4
Generalized time string	2.5.5.11	24	Time-string format defined by ASN.1 standards. See standards ISO 8601 and X.680 for more information.* This syntax uses four characters to represent the year.	6

*These two time formats are further described in Chapter 6 in the "Specifying Values" section.

TABLE 8.20 Syntaxes for Reference Data Types

Syntax	attributeSyntax	oMSyntax	Description	Count in Base Schema
DN (distinguished name or DN String)	2.5.5.1	127	Distinguished name of an object in the directory. If the target object is moved or renamed, Active Directory updates the DN attribute accordingly.	92
DN with binary (DN with octet string)	2.5.5.7	127	This syntax stores a distinguished name along with some binary data. Active Directory keeps the DN up-to-date. The format is B:*hex digit count*:*bytes as hex*:*DN* (e.g., B:6:F12A4B:*someDN*).	4
OR name	2.5.5.7	127	An X.400 syntax (related to e-mail addresses).	0
Replica link	2.5.5.10	127	Syntax that repsFrom and repsTo attributes use to control replication. The corresponding attributes contain things such as the up-to-date vector of a replication partition.	2
Presentation address	2.5.5.13	127	OSI application entities use presentation addresses to address other application entities. See RFCs 1278 and 2252 and ISO DIS 7498-3 for more information.	1
DN with Unicode string	2.5.5.14	127	This syntax stores a distinguished name along with a string. Active Directory keeps the DN up-to-date. The format is S:*character count*:*string*:*DN* (e.g., S:5:hello:*someDN*).	0
Access point DN	2.5.5.14	127	An X.400 distinguished name.	0

All seven syntaxes in Table 8.20 have oMSyntax = 127, which means that they must have oMObjectClass also defined. The latter attribute distinguishes them, because some of them have identical attributeSyntax and oMSyntax. We list the oMObjectClass values in Table 8.21.

TABLE 8.21 `oMObjectClass` Values for "127" Syntaxes

Syntax	`attributeSyntax`	`oMSyntax`	`oMObjectClass` (hexadecimal)
DN	2.5.5.1	127	2B0C0287731C00854A
DN with binary (DN with octet string)	2.5.5.7	127	2A864886F7140101010B
OR name	2.5.5.7	127	56060102050B1D
Replica link	2.5.5.10	127	2A864886F71401010106
Presentation address	2.5.5.13	127	2B0C0287731C00855C
Access point DN	2.5.5.14	127	2B0C0287731C00853E
DN with Unicode string	2.5.5.14	127	2A864886F7140101010C

MULTIVALUED ATTRIBUTES

Each multivalued attribute can have up to 850 values, except for linked multivalued attributes, such as group members, which can have 5,000 values (i.e., group members).

When one value is added, deleted, or modified, the whole attribute is replicated to other domain controllers. Therefore, it is not a good idea to make multivalued attributes too large. Another reason to favor relatively small multivalued attributes is that when you read the attribute, all values are normally returned together.

The values are returned in random order; if you write a program or script to read the values, you cannot depend on the order.

Even though returning all values of a multivalued attribute is the normal behavior, LDAP allows specifying the range of values. This is explained in Chapter 6.

Searches

Some attributes are indexed, which allows fast searches based on those attributes, and some attributes are part of the global catalog. The indexed and global catalog attributes are defined with two `attributeSchema` attributes, which are described in Table 8.22.

NOTE Indexing and global catalog membership are per-attribute settings, not per-class settings. In other words, if the `givenName` attribute is indexed, this will apply to any class that happens to use `givenName`.

TABLE 8.22 Search Attributes of an attributeSchema Object

Name	Syntax	Multi-valued	Description	LDAP Display Name Attribute Example
`SearchFlags`	`Enumeration`	No	The bits in this number define whether the corresponding attribute is indexed and how it is treated in searches. (Table 8.23 describes the bits.)	9
`isMemberOf-Partial-AttributeSet`	`Boolean`	No	If True, the corresponding attribute is part of the global catalog and consequently is replicated to all global catalog servers.	True

NOTE You might wonder how it is possible that the member attribute is part of the global catalog (`isMemberOfPartialAttributeSet` is `True`), but only universal group members appear in the global catalog, whereas global and domain local group members don't. The answer is that Microsoft hard-coded this difference to Active Directory; that is, group membership doesn't care about `isMemberOfPartial-AttributeSet attribute`.

`SearchFlags` is a *bit-field* attribute. It contains 32 bits, 5 of which have a meaning. Those 5 bits are the least significant ones, which means that they are the rightmost bits if you use Windows Calculator to convert a decimal number to binary. The bits are described in Table 8.23. The bit value "1" means True—that is, the setting is on.

AMBIGUOUS NAME RESOLUTION

Ambiguous Name Resolution (ANR), which is mentioned in Table 8.23, needs a few words of explanation. ANR is an LDAP feature that allows using a simple LDAP filter instead of a complex filter in certain LDAP searches. You can use ANR manually with LDAP filters (discussed in Chapter 6). Also, when you perform a search with Windows 2000 Address Book, it will use ANR for you.

The following eight attributes are part of the ANR set in the base schema:

- `displayName`
- `givenName`

TABLE 8.23 `SearchFlags` Bits

Bit from Right	Hex Value	Description	Count in Base Schema
1	1	If set, the attribute is indexed.	64
2	2	If set, the attribute is indexed over container and attribute.	0
3	4	The attribute is part of the Ambiguous Name Resolution (ANR) set, which is explained in the "Ambiguous Name Resolution" section. This bit should be used in conjunction with the first bit.	8
4	8	The attribute is preserved when an object is changed to a tombstone (i.e., "deleted"). All attributes without this bit are deleted.	24
5	10	If set, the attribute is copied when duplicating a user with the Users and Computers snap-in.	33

- `legacyExchangeDN`
- `name` (RDN)
- `physicalDeliveryOfficeName` (user properties, General tab, Office)
- `proxyAddresses`
- `sAMAccountName` (user properties, Account tab, Pre-Windows 2000 name)
- `sn` (Surname)

ANR gives flexibility in two ways:

- Only a partial match is required from the beginning of the text. If you search for "Brown," you will get both "Brown" and "Brownfield."
- The search is performed on several attributes (those eight just listed). If you search for "Brown," it can be the starting part of not only the surname (last name), but also `displayName`, `sAMAccountName`, or any of the other five attributes.

If you search for two words, ANR works as just described, with one addition. It will also try if the words are "first name—last name" or "last name—first name." If

you want to find Jack Brown, you can type either "jac bro" or "brow jack." The corresponding LDAP filters would be

```
(anr=jac bro)
(anr=brow jack)
```

It is possible to suppress the first name/last name functionality and/or the last name/first name functionality. You do this by modifying the same `dSHeuristics` attribute that is used to enable the List Object permission. We explained the procedure in Chapter 4 in the "Enabling and Using the List Object Permission" section.

Miscellaneous Characteristics for Attributes

Most of the miscellaneous `attributeSchema` attributes are also `classSchema` attributes. Because we described the classSchema attributes when discussing classes, there is no need to repeat the discussion of the following attributes:

- `nTSecurityDescriptor`
- `isDefunct`
- `objectClass`
- `objectCategory`
- `schemaFlagsEx`

One new attribute, `isEphemeral`, is not used, so we are left with only two attributes to address here: `systemOnly` and `systemFlags`. Although those two attributes also appeared in the discussion of classes, they have some new aspects. Table 8.24 describes the attributes `systemOnly` and `systemFlags`, and Table 8.25 continues the description of the `systemFlags`.

NOTE For some peculiar reason, 14 of the base schema attributes and two of the classes are not marked to belong to category 1. Therefore, you could deactivate them, for example, even though you are not supposed to be able to do that for category 1 attributes or classes.

AttributeSchema Object Property Pages

There is only one property page for `attributeSchema` objects in the Schema Manager snap-in, and there is no Security tab. Figure 8.17 shows the property page for the object `lDAPDisplayName`.

TABLE 8.24 Miscellaneous Attributes of an `attributeSchema` Object

Name	Syntax	Multi-valued	Description	LDAP Display Name Attribute Example
`systemOnly`	`Boolean`	No	If True, you can set the value for the attribute only when creating an object. Afterward, only the DSA can change the value.	False
`systemFlags`	`Integer`	No	The bits in this number define things such as whether the corresponding attribute is replicated or constructed. (Table 8.25 describes the bits.)	16

TABLE 8.25 `SystemFlags` Bits

Bit from Right	Hex Value	Description	Count in Base Schema
1	1	If set, the attribute is not replicated from one domain controller to another.	39
2	2	This is an undocumented bit used by the system.	43
3	4	If set, the attribute is constructed. It is built from other attributes and not stored in the schema on disk.	22
5	10	If set, the attribute belongs to category 1 (part of the base schema); otherwise, it belongs to category 2 (an extension).	849
28	800 0000	This is an undocumented bit used by the system.	21

Of the 34 `attributeSchema` attributes that we have discussed in this section, Figure 8.17 shows the following 11, with the values in parentheses (we didn't count `showInAdvancedViewOnly`). To see all of the attributes, you need ADSI Edit.

- `lDAPDisplayName` ("lDAPDisplayName")
- `adminDescription` ("LDAP-Display-Name")

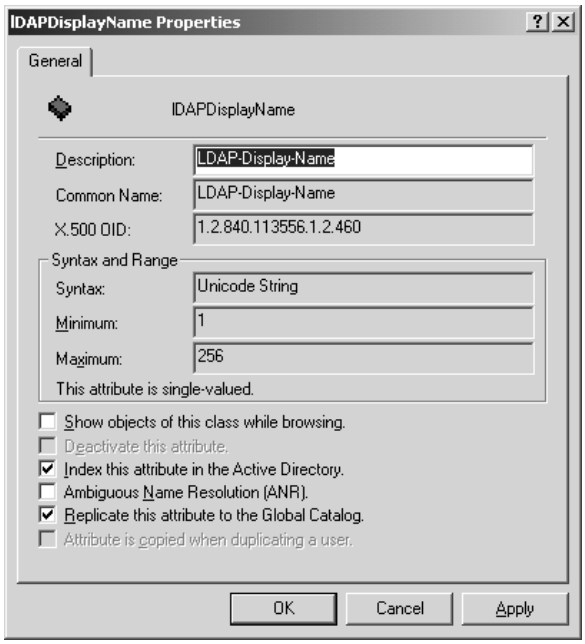


FIGURE 8.17 With the Schema Manager snap-in you can see the attributes of various attributes, such as `lDAPDisplayName`.

- `cn` ("LDAP-Display-Name")
- `attributeID` ("1.2.840.113556.1.2.460")
- `attributeSyntax`, `oMSyntax`, `oMObjectClass` ("Unicode String")
- `rangeLower` ("1")
- `rangeUpper` ("256")
- `isSingleValued` ("This attribute is single-valued")
- `showInAdvancedViewOnly` (Note that even though checking the "Show objects of this class while browsing" box sets this attribute to False, it doesn't have any practical meaning.)
- `isDefunct` ("Deactivate this attribute" not checked)
- `searchFlags` ("Index this attribute in the Active Directory" checked, "ANR" not checked, "Attribute is copied when duplicating a user" not checked)
- `isMemberOfPartialAttributeSet` ("Replicate this attribute to the Global Catalog" not checked)

CONCLUSION

In this chapter we discussed how the schema is built up from classes, attributes, and syntaxes. We also described the attributes of `classSchema` and `attributeSchema` objects.

We have examined the schema in detail thus far, but we have not made changes to it. In the next chapter we evaluate if we need to extend the schema, and if so, how to extend it.