Designing OU Structures that Work

Ken St. Cyr

AT A GLANCE:

- Key principles of good OU design
- Common OU design models
- Document your OU design

Don't underestimate the importance—and complexity—of designing a good Organizational Unit (OU) structure. I've found that IT departments often go in either direction—they

put too much emphasis on the OU structure or they don't put enough thought into it. Either way, this can lead to problems with your Active Directory® model.

Overemphasizing the OU structure takes focus away from other areas of Active Directory design, such as planning the site topology or thinking about domain controller sizing. On the other hand, when OU planning is put on the back burner, Group Policy and delegation suffer.

One of the excuses I've heard on occasion is that the OU structure is flexible and can be changed at a later date if it doesn't fit. It's true that the OU structure is flexible; however, administrators often discover that changing the OU structure down the road is harder than they had originally anticipated. Sure, new OUs can be added, but the old ones are not easy to clean up.

A poorly planned OU structure tends to take on a life of its own. If a new object is created in the directory and the administrator doesn't know where in the OU structure to place the object, he will either create a new OU or put the object somewhere that it doesn't belong. There are dangers to both of these scenarios. Creating a new OU is easy to do, but hard to track over the long run. Rampant OU creation contributes to a chaotic OU structure, and it's easy to let things creep into the directory undocumented. On the other hand, if you add an object to an existing OU in which it doesn't really belong, the new object might receive policies that it shouldn't get or permissions to the object might be delegated to unintended users.

When designing OU structures, you should keep a basic equation in mind: simplicity + adaptability = sustainability. If your design is too simple, it may not be adaptable and therefore will have to be changed too often. If your design is too adaptable, then everything will be compartmentalized, and that brings about too much complexity.

There are three key principles regarding Group Policy, delegation, and object administration that can help guide your design decision. These principles can be summed up with three questions you should ask yourself that will help to ensure the OU structure you are creating will stand the tests of time and organizational change:

1. Does this OU need to be created so a unique Group Policy Object (GPO) can be applied to it?
2. Does a particular group of administrators need to have permissions to the objects in this OU?
3. Will this new OU make it easier to administer the objects within it?

If the answer to any of these questions is "yes," then you should probably create the OU. If the answer to all three questions is "no," then you should rethink the layout and determine whether a different design might create a better fit. But before I dive into this any deeper and show you how to apply these principles, I should first explain why these principles are important.

Principle 1: Group Policy

The first principle of OU design is to take into account the Group Policy Objects that will be applied to an OU. A GPO allows you to configure settings for users and computers in an enforceable manner. You can define multiple GPOs in Active Directory and apply them to the entire domain, various OUs, or even the sites in the domain. GPOs are split into two categories—one for users and one for computers.

Both computer policies and user policies can be defined in the same GPO. The User Configuration section of the GPO mostly defines the experience the user will have when logged in. These types of settings also exist in the Computer Configuration section, but this section also contains more settings related to the security of the computer, such as who can log on to the computer locally or how data is encrypted.

Here are some fundamentals to keep in mind when defining OUs to support GPOs. First off, just because user and computer policies can both be defined in the same GPO, that doesn't mean it's a good idea to put user and computer objects in the same OU. Combining them in the same GPO makes GPO application harder to troubleshoot. This becomes very evident when you have a loopback policy enabled.

Secondly, many people forget that you can apply GPOs at the site level and therefore design their OU structures to model their site structure for the purposes of GPO application. This is a common model of OU design, known as the Geographic Model. I'll talk more about this model in a bit. I would be remiss not to acknowledge that the Geographic Model has its place in OU design, but as you'll see later, I don't believe that GPO application should be the primary reason for implementing this model.

Also, when you think about your OU structure in terms of GPOs, the goal should be to eliminate complexity. Make sure that the OU adds to the flow of GPO inheritance. If your OU contains servers that require the same policy as other servers, consider putting these computer objects under a broader Servers OU and creating multiple OUs for different server types below the Servers OU (see **Figure 1**). This can simplify policy application, as each computer object in the lower OUs will get the policy from the Servers OU as well as any other policies that are specific to that specific type of server.

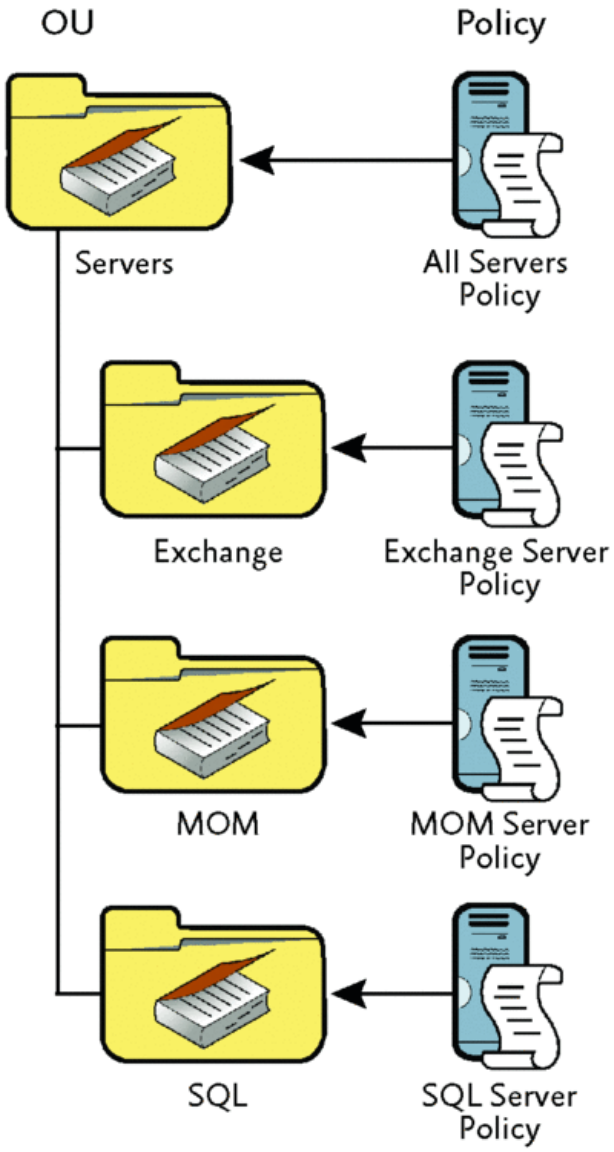

Figure 1 **Creating multiple OUs for different server types** (Click the image for a larger view)

Another fundamental is to ensure that you don't unnecessarily create or link multiple GPOs. With a GPO, you can create one policy and apply it to multiple OUs. This is sometimes helpful, but it can also be harmful. If you have to change a GPO setting and you have an overly complex system of linked GPOs, you could accidently apply a change to the wrong objects. The more links you create, the more difficult it is to grasp the scope of a policy. Likewise, you should avoid creating additional policies with the same settings as other policies. If you find that you often do this, consider modifying your OU structure to apply a new GPO inheritance model.

And finally, you should almost always create a new OU for user objects and computer objects. By default, user and computer objects are placed in containers, which don't allow you to link GPOs directly to them. GPOs can be applied to the Users and Computers containers from the domain, but unless you block inheritance elsewhere, those policies will be applied to every user and computer in the domain. In Windows Server® 2003, you can use the rediruser.exe and redircomp.exe tools to change the default location for user objects and computer objects to the OU that you created for them.

Principle 2: Delegation

It is important that you create the OU structure in a manner that is consistent with how permissions are delegated in the domain. Keep in mind that when permissions are delegated in Active Directory, the permission changes are made only to the object. So if you were to give a user Full Control for a particular computer object, that person would be able to modify the attributes of the object, but they would not have Administrator privileges on the computer itself.

Here are some recommended practices regarding delegation for when you design an OU structure:

Design with Permission Inheritance in Mind For instance, say you want your Tier 1 administrators to be able to change the password for most accounts. There is a special group of users for which the administrators should not have the ability to reset passwords, but the admins do need to be able to change the display names on those accounts.

You really have two options here. First, you could create two separate parallel OUs and separate the special users from the regular users. This, however, means that if you want to change the delegation options for all users then you have to change these permissions in two separate places. This also contradicts the practice of not unnecessarily multiple linking policy (see **Figure 2**).

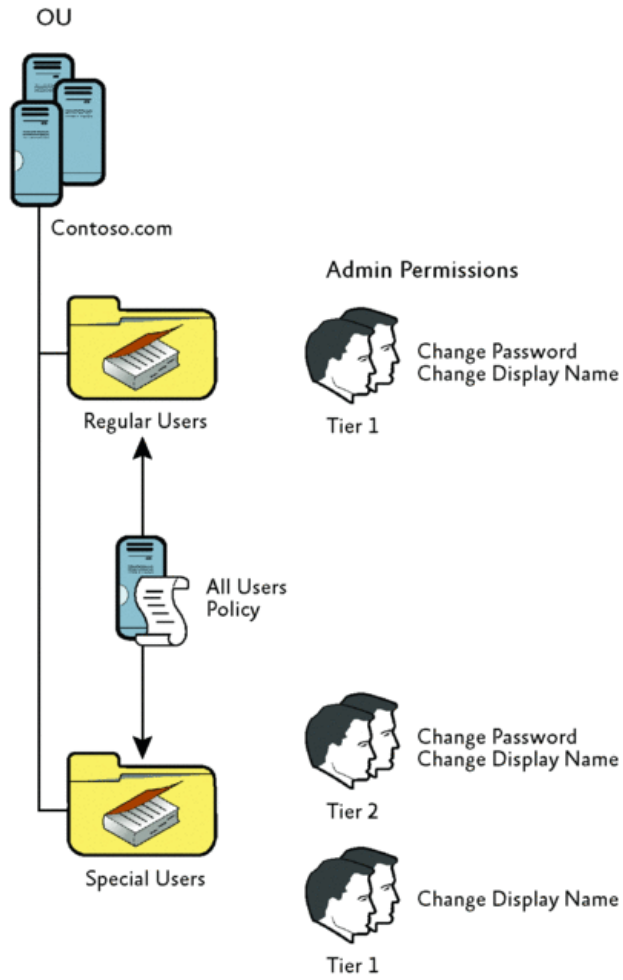

Figure 2 **Maintaining two separate parallel OUs** (Click the image for a larger view)

The other option is to create a nested OU and place an explicit deny permission on the OU that has special users in it. Any delegation expert will tell you that an explicit deny is not preferable—but in this case, you need to select the lesser of the two evils (see **Figure 3**). You can either duplicate and maintain the settings on two separate OUs or you can place an explicit deny on one of the OUs. The explicit deny is actually the better long-term decision.

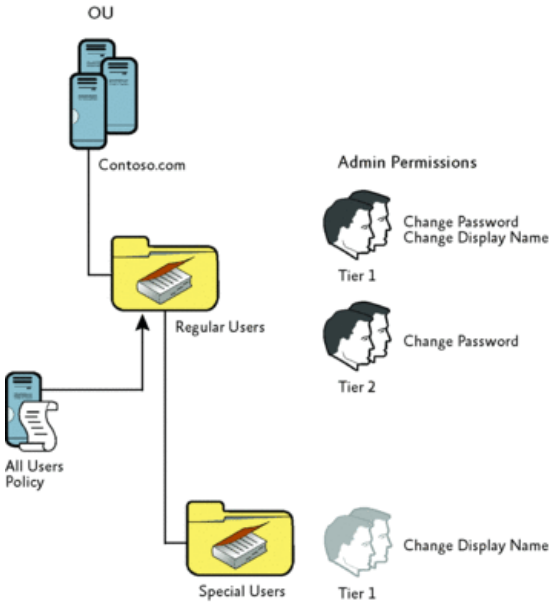

Figure 3 **Using an explicit deny permission** (Click the image for a larger view)

Beware of AdminSDHolder This example works well unless the special users all belong to one of the Admin groups (Domain Admins, Schema Admins, Enterprise Admins, or Administrators) since permissions for accounts in these groups are handled differently. The idea is that you don't want to accidently give someone permissions to an admin account.

If you create a separate OU for administrators, you'll find that the permissions that you delegate to them keep disappearing. This is due to AdminSDHolder, which is a special container that applies its Access Control List to every administrator account on a specified interval. As a result, any delegation changes you make to an admin account will be reversed if the changes aren't also made to the AdminSDHolder container. So you shouldn't separate administrator accounts from other accounts for the purpose of delegation. It is preferable, though, to separate out the administrator accounts for the purpose of Group Policy—this is especially true in Windows Server 2008, where you can have multiple password policies.

Principle 3: Object Administration

The OU should facilitate administration of the objects. Grouping objects into the same OU can often make it easier to perform bulk changes. The Active Directory Users and Computers snap-in lets you edit certain attributes on a selection of multiple objects. So if you regularly have to change an attribute on a group of objects, it is easier to do if they are all in the same OU.

This is also particularly useful if you are making updates with a script. Scripting languages make it very easy to enumerate all of the objects in an OU and deal with them one by one. The other option is to search for and modify each object individually. Simply placing the objects in the same OU for administration can sometimes save you hours of unnecessary work each week.

Another way to help with the administration of objects is to separate the objects into different OUs based on their type. Creating a separate OU for printer objects or published shares ensures that these objects won't need to be weeded out when you perform administration on other objects in the OU. This practice also aligns with the practice of not grouping user and computer accounts together in the same OU.

Choosing a Model

Now that I've covered the principles of OU design, I can take a closer look at some common design models. Note that there are many more models than the ones I have space to cover here. And you aren't limited to working within the constraints of a single model. You can pick pieces from each and build your own hybrid model that addresses your particular needs.

Just about any model can be successful on a small scale, but when your enterprise grows, your ability to keep a handle on the environment shrinks. So make sure that you first thoroughly test your model in an adequate lab environment. And remember that while OU structures can be changed easily at first, they are harder to change the longer they are in place.

The Shallow Model

The Shallow Model gets its name from the fact that it is kept mostly flat. In this model, a few high-level OUs contain the majority of the objects (see **Figure 4**). This model is mostly exercised in smaller businesses where there is a small IT shop, there aren't a lot of different divisions, or people tend to play multiple roles. I typically recommend not going any deeper than 10 sub-OUs, though Microsoft suggests a 15 sub-OU limit before performance penalties are realized.

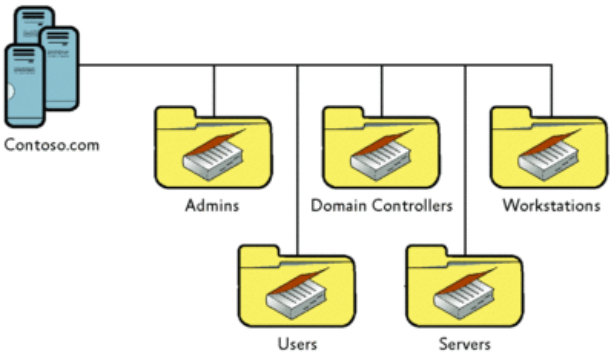

Figure 4 **The Shallow Model has a few high-level OUs that contain the majority of objects** (Click the image for a larger view)

If your Human Resources person is also your payroll person, then it doesn't make sense to create two separate OUs for Human Resources and Payroll. In the Shallow Model, all the user objects can be grouped into one big Accounts OU or they can be kept in the default Users container. At the very least, your user objects should be separated from your computer objects.

For this model, I also recommend that you go a step further and separate your workstations from servers. Then you can at least apply different Group Policies without having to define a GPO that uses a Windows® Management Instrumentation (WMI) Query to filter out workstations or servers.

One advantage of keeping your OU structure wide, rather than deep, is that Active Directory searches will execute faster. I typically recommend not going any deeper than 10 sub-OUs. Your control over objects isn't very fine-grained in this model, but if you're managing objects in a small business, you won't need that fine-grained control. Thus, this model would be difficult to use successfully in a large enterprise, but it works very well in smaller organizations.

The Geographic Model

In the Geographic Model, you create separate OUs for different geographic regions. This model is best suited for organizations that have decentralized IT departments but don't want to incur the costs associated with operating multiple domains (see **Figure 5**).

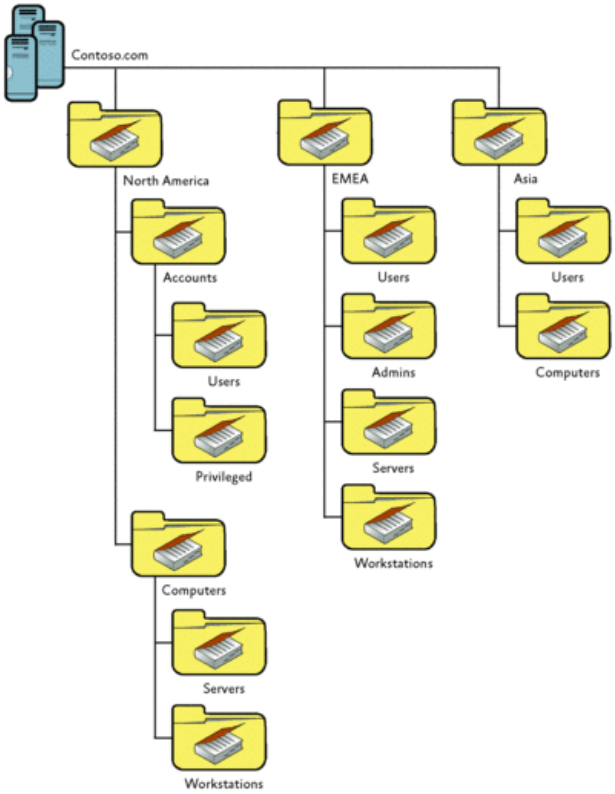

Figure 5 **The Geographic Model separates OUs by geographical region** (Click the image for a larger view)

Let's say you have offices in Atlanta, Baltimore, and Seattle. If each site manages its own users and computers, then this might be a good fit in terms of delegation. But what happens if a Seattle user flies out to Baltimore for training and then locks out his account. The IT folks in Baltimore may not be able to help this user if they have not been delegated permissions to this user's account. If it's 8 A.M. in Baltimore, it's 5 A.M. in Seattle, which means the user may have to wait hours before he can reach someone in the Seattle office for help.

Some global companies use a "follow the sun" model, where help desk calls are routed to a site located in a time zone where it's currently standard business hours. This means the company doesn't have to operate a 24-hour help desk at each site but can still provide late-night employees with help, such as unlocking their accounts, when necessary.

If this is your model, creating separate OUs based on geographical location probably isn't the best choice for your operational needs. You would have to delegate separate permissions to each regional help desk for every OU of users. However, if your sites do have their own IT departments, then the Geographic Model might, in fact, be a good model for your organization.

The Geographic Model is also difficult to pull off in a single domain due to the nature of how a domain operates. A domain tends to have a different security model from other domains. This is more evident when you look at an enterprise-wide application, such as Microsoft® Exchange.

The Exchange Server in Atlanta may have different message policies defined, but all of the Exchange Servers in the enterprise probably have the same GPOs applied. If this is the case, then putting the Exchange Servers in separate OUs based on region would cause you to unnecessarily link the same GPO to multiple OUs. In terms of delegation, you have to ask if the Exchange administrators really need unique permissions to the computer objects for the Exchange Servers. In most cases, computer objects that are split into geographic OUs are done so for Group Policy purposes, not delegation purposes.

The Type-Based Model

The Type-Based Model classifies objects by their purpose (see **Figure 6**). When you created your last user object, was it for a regular user account, an administrative account, or a service account? In a Type-Based Model, each of these objects is treated differently.

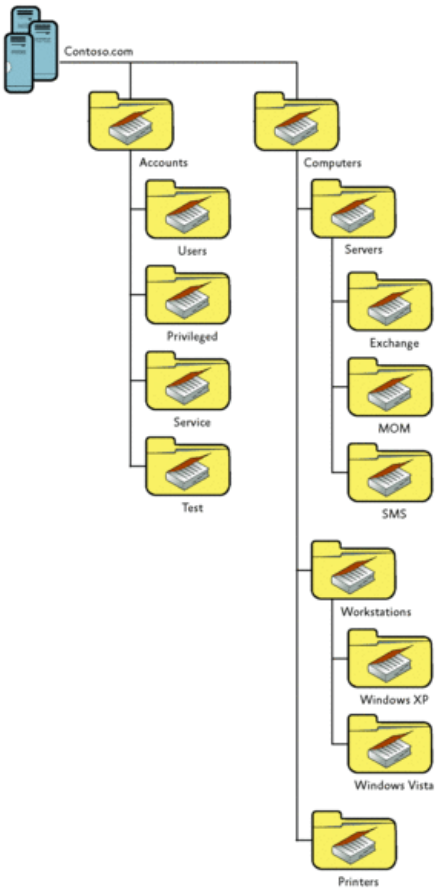

Figure 6 **The Type-Based Model groups objects according to their functions** (Click the image for a larger view)

In most cases, you should distinguish among different classifications of user objects for policies. Your policies will most likely differ based on user account type. For example, allowing people to log on to a computer with a service account is generally a bad business practice. Service account passwords are usually shared among many people, so if someone logs on with a service account, their identity remains anonymous. If something were to happen, you would have trouble tracking down the user who was logged in when the event occurred. In this example, you'd need to set a policy on service accounts that prevents interactive logon. This fits well in the hierarchical model shown in **Figure 3**.

You could use GPO inheritance to your advantage here. For instance, you could have an All Users policy that refers to policies in place for all user objects. In addition, you could have a separate and distinct policy for service accounts, which builds on the All Users policy. This approach ensures your service accounts have the same base set of policies as every other account as well as the specific logon restrictions.

This approach also works well for delegation, where you are using permission inheritance instead of GPO inheritance. Suppose you want your Tier 2 administrators to have the ability to reset passwords for all accounts except for the Tier 3 administrator accounts. With a flat OU structure, you would have to delegate permissions to each OU that holds user accounts. However, in a Type-Based Model with a hierarchical structure, you can give the Tier 2 group "reset password" permissions on the Accounts OU, and then at the Tier 3 OU, you could simply uninherit the permissions or even set an explicit deny for Tier 2 to reset passwords.

This also works well for computer objects. Servers and workstations can be separated allowing different policies to be applied to them. Servers can then be further sub-divided into functions (see **Figure 1**). In this design, you can set a high-level policy on the Servers OU that affects all servers and still set individual policies on each lower-level OU.

For example, let's say you have a Microsoft Operations Manager (MOM) service account. With this tiered model, you can create a MOM GPO and apply it to the MOM Servers OU. And then inside that GPO, you can give the MOM service account rights to logon as a service. This would only apply to the MOM servers in that OU. The MOM servers will still get the Servers GPO from the higher-level Servers OU, but they will also get the specialized MOM GPO, which is linked at the MOM OU.

Document the Design

It can be very rewarding to design an OU structure that will stand up to the many changes an Active Directory environment encounters. But you need some way to track the dynamic characteristics of OUs. If you don't have this, you could quickly lose a handle on your environment. When things do need to change and an OU needs to be added or deleted, there must be clear guidance on what to do to ensure that your model continues to follow your design model, adhering to the three guiding principles. That's why you must have a well-documented design.

Microsoft provides documentation guides in the Windows Server resource kit. These guides are good if your structure is a solid framework that you don't expect to change much. But most organizations have a very dynamic structure that changes frequently. So here are some important tips to ensure that your OU structure is well-documented and able to support a dynamic environment.

Make Sure All the Info Is Relevant I'm a strong believer in documentation that has purpose. Too many operational documents have so much extraneous information that it's hard to find the stuff that matters. Don't get caught up in the process of documenting just for the sake of documenting. Do you really need to include the number of objects in each OU or each Access Control Entry (ACE) on the Access Control List for the OU? For OU documentation, the following information will usually be sufficient:

- OU Name
- Short description
- Who created it—or who to contact for more information or changes
- When it was created

Don't Make Updates Difficult If you have to tediously update some complex Microsoft Word document, you're more likely to put off entering updates. It's OK if you hold off on inputting small changes when you know you will soon have a bigger bulk of updates to input all at once. Unfortunately, people often forget these little changes or simply keep putting them off and then the job never gets done. Therefore, updating the document must be very easy so as not to discourage you. In most cases, a simple Microsoft Excel® spreadsheet with a few columns will work fine.

Make Comments on the Object Itself OU objects have a description attribute where you can enter comments. Rather than writing the comments in the design document, consider putting them in the description attribute so others can tell right away what the OU is for. If you need to include more details, then put a brief description in the description attribute and include more details in the OU document.

Automate the Documentation A script can be written to dump the contents of the OU structure out to a text file, an Excel spreadsheet, or even an HTML file. This script could be run on a scheduled task every night. This can be very useful if you incorporate comments in the description field of the OU. Then it's just a matter of dumping out the description attribute to the file and you automatically have a fully documented OU structure that is always current. If you create a new file every time the script runs, as opposed to overwriting the existing document, you can keep a historical record of how the OU structure has changed over time.

Unfortunately, most administrators don't realize the importance of good OU structure documentation until they really need it. And by then, at 3 A.M., it's nearly impossible to figure out what other OUs were accidently deleted without performing a restore.

Don't wait until this happens to you. I highly recommend that you become proactive in this area and immediately start an OU document and designate one person to be responsible for updating it. And if you follow the rule of making the documentation easy to update, this will be a very small task to keep rolling. □

Ken St. Cyr is a consultant for Microsoft with 10 years of IT industry experience. He has designed and implemented directory solutions based on Active Directory since its inception.
