



realtimepublishers.com[™]

The Definitive Guide™ To

Windows 2000
Administration



Sean Daily

Darren Mar-Elia

Chapter 5: Managing the Distributed File System	171
How Dfs Works on the Server	172
A Share of Other Shares.....	172
Redundant Shares	173
Dfs Walkthrough.....	175
Defining a Dfs Root	175
Making the Root Redundant	179
Defining Dfs Links.....	181
Making a Redundant Dfs Link	182
A Client’s View of Dfs.....	184
Recommended Uses for Dfs.....	185
Creating a Distributed Network of Backups for Users’ Home Directories.....	185
Organizing Disparate Shares and Naming Conventions into One Logical Structure	185
Publishing a Distributed Web site.....	186
Avoiding Service Outages that Result from Scheduled Shutdowns	186
Replicating Profile Directories.....	186
Technical Overview of Dfs	186
AD Integration.....	186
File Replication Service	187
What’s Replicated, What Isn’t	187
Tuning Replication.....	188
Troubleshooting Dfs.....	190
Dfs Event Log	191
FRS Logging	192
FRS Staging.....	192
Dfs and FRS Registry Tweaks	193

Chapter 5: Managing the Distributed File System

“Do they still come out with technologies that amaze you sometimes?” At first thought, that was a question that had a simple answer. “Nah, not really,” I replied to my girlfriend as we made our way down the interstate. “Well, every now and then something comes along that I think is pretty cool—but most of it’s just the same old stuff.”

It’s ironic that she’d ask that question on the same day I’d start writing this chapter. For anyone who’s been working in networking for any length of time, it’s easy to think that there’s not much new under the sun. After all, a lot of the core concepts of networking—authenticating users, sharing files, accepting dial-in connections, and so on—have been around for at least a decade in one form or another. However, what does amaze me are the high-availability features (features that increase a computer’s fault tolerance, thus keeping it running and available) that have been developed in recent years.

Here in the third millennium, we have the technology to build highly available systems using redundant power supplies, redundant NIC cards, redundant disk drives, and more—all of which can be replaced without taking a system offline. It costs more to implement those types of fault tolerance, but it’s worth it if you can provide more uptime, right? That’s great in theory, but that theory falls flat on its face when your CFO can’t open up one of his budgeting spreadsheets because you’ve taken the system down for routine maintenance.

For better or worse, adding redundancy (ways for one computer system to take on the processing or transmission load when others fail) to the most basic of networking services—file sharing—was costly and difficult before Win2K. Clustering was an option for some individuals, but it was costly at best, and in the worst case, it still relied on a “shared” set of disks for storing data. That shared set of disks could easily be rendered inoperative for hours if a catastrophic physical event occurred (such as a leak in the ceiling that took out the shared drive array). For the last decade, we’ve had plenty of ways to make hardware redundant but no really easy way to make files “redundant.” Win2K is changing all that.

Win2K comes with a (somewhat) new capability called the *Distributed file system*, or *Dfs* for short. I say “somewhat” because *Dfs* actually existed back in the days of Windows NT, but in a much more limited (and much less useful) form. *Dfs* was designed to answer one question for users: “Where are my files?” The question “Where are my files?” can mean one of two things. The first is, “Which server are my files stored on?” The other is, “Um, is the server offline? I can’t seem to get to my files.”



As a historical note, I should mention that the original *Dfs*, which was part of the DCE specification, has been around for a very long time. So although Microsoft is certainly the most visible champion of *Dfs* to date, it didn’t invent the concept.

Dfs answers both of these questions. First, it tackles the problem of finding files throughout a complex enterprise network by consolidating shares located on different servers into one logical structure. Second, *Dfs* solves the problem of finding files by maintaining replicas of shared files on other servers. By doing this, *Dfs* makes sure that your users’ files are always available.

In this chapter, we’ll be discussing the following aspects of *Dfs*:

How it works on the server



How it works on the client

What you need to implement it

How it works behind the scenes

A sample Dfs configuration

How Dfs Works on the Server

If you've been working with Win2K for a while, the concept of file shares is nothing new. Quite simply, you create a directory on a system, put files in it, then share that directory on your network so that users can get to the files. It's all rather straightforward and easy to use.

However, basic file sharing is plagued with two problems. First, it's not redundant. If you take a server offline that is sharing files, those files are unavailable. Second, once you begin working with dozens of servers and a handful of shares on each server, your users will find it difficult to remember where things are stored. Just imagine a medium-sized organization with 25 servers and an average of 10 shares per server—that's 250 possible areas that users might need to connect to. In both of these cases, Dfs can come to the rescue.

A Share of Other Shares

Dfs solves the organizational nightmare by allowing you to create a special type of file share that is filled with other file shares. For example, let's say that you have five servers in your company and each server has a share that your users need to access, as shown in Figure 5.1.

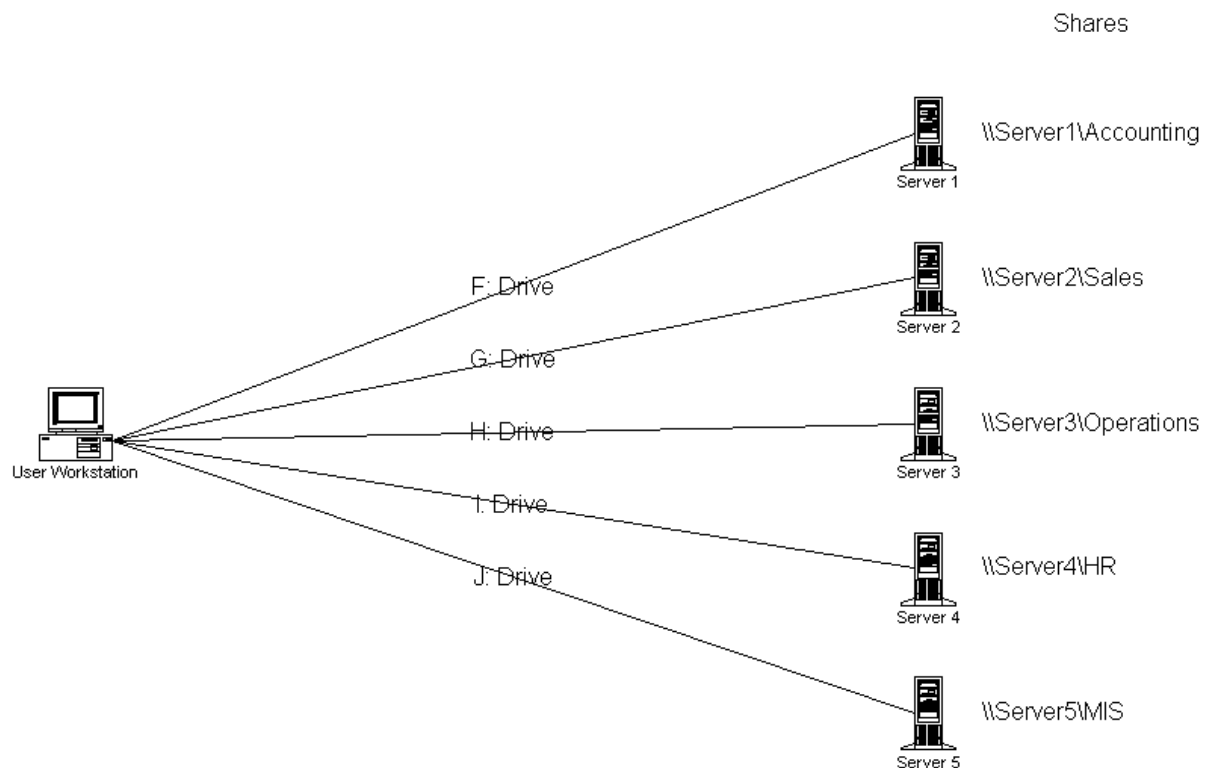


Figure 5.1: Typical non-Dfs client/server file-sharing configuration.

Navigating across all of these shares can be tricky. At the very least, your users will need to remember five different drive letters. In the worst case, they'll need to remember the server names and share names for all five shares. Wouldn't it be nice if you could organize all of those file shares into one "megashare"?

Using Dfs, you can define a company-wide share, then roll all of your individual shares into the master Dfs share. This makes the individual shares easier to navigate and much easier to remember. When users navigate through the objects in a Dfs structure, they move from one server to another without needing to know about it. Figure 5.2 shows the same set of servers and shares shown in Figure 5.1, but this time with a Dfs structure organizing everything into one logical namespace.

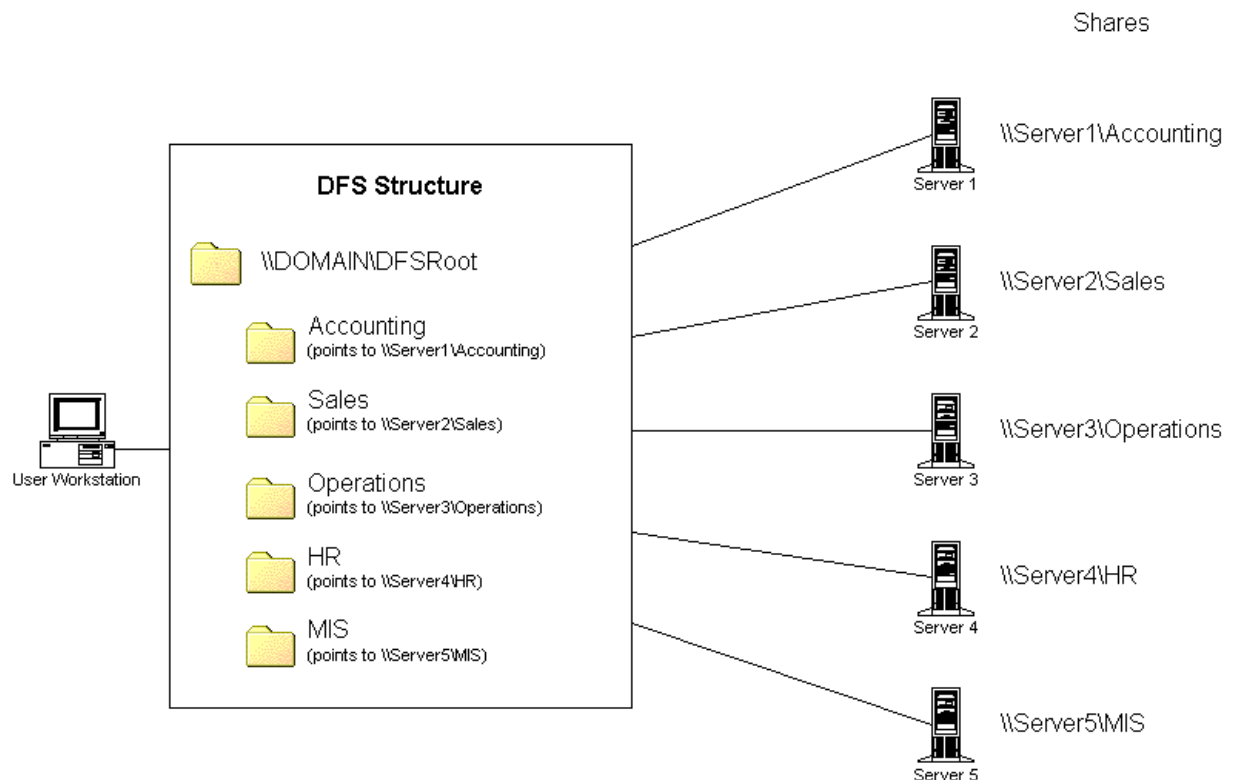



Figure 5.2: Five servers, each with a share, organized into a Dfs structure.

Redundant Shares

Okay, so organizing all of your shares into one Dfs structure is cool, but it might not be a compelling enough reason to adopt Dfs in and of itself. Well, the benefit of organizing all of your shares into a Dfs structure is that you can then make all of them redundant by defining synchronized replicas on other servers. Now *that's* cool.

Redundant shares are, in my opinion, the single largest reason for implementing Dfs in any organization, even a small company with only two servers. Using a simple point-and-click interface, you can quickly and easily make your important file shares redundant so that if one server goes down, an alternate copy of the same data keeps servicing users' requests. This capability is a welcome addition to Win2K.

Let's take another look at the example we've been using. We have five servers, each with one share on it, and we've published those shares into a Dfs structure. We can easily add redundancy into this scenario by defining "backup" shares on each of our five servers, then synchronizing the data between the master share and the backup. Figure 5.3 shows our sample organization, with backup shares on each server and defined as alternates within Dfs.

 When you're implementing read-write replicas across multiple systems, it's extremely important to know how changes are replicated among servers. Be sure to read "File Replication Service" later in this chapter.

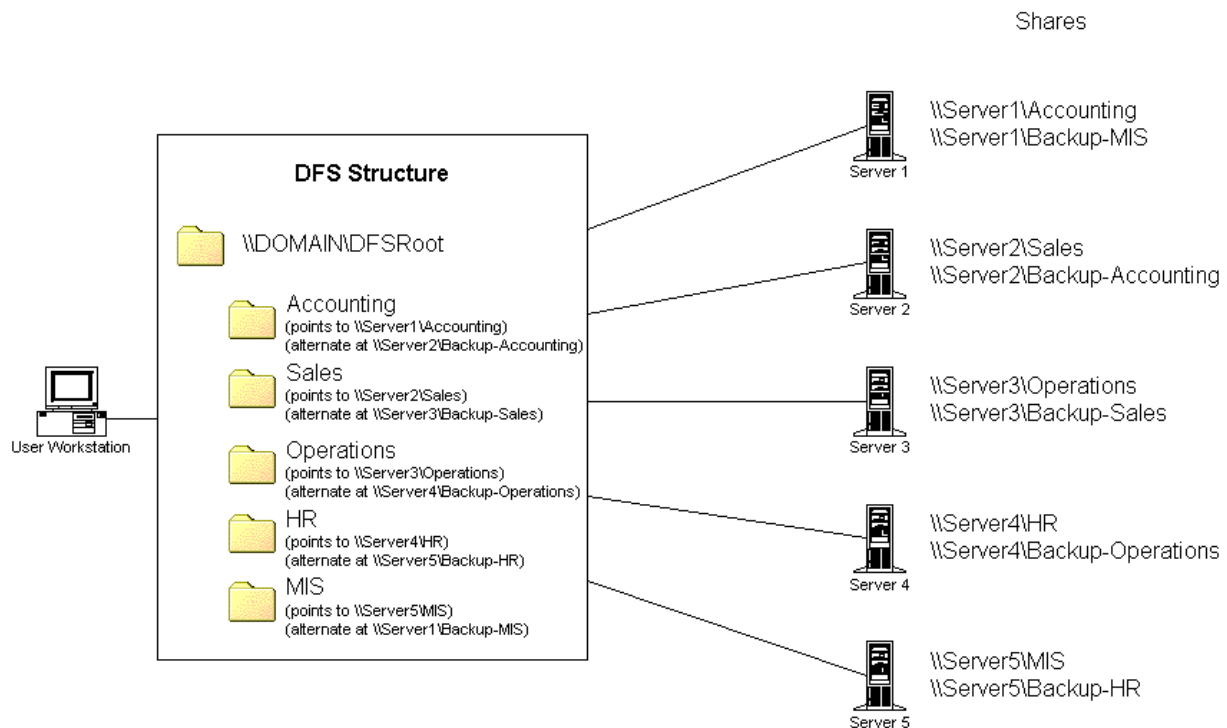



Figure 5.3: Servers with backup shares.

 Each server obviously requires sufficient disk space to support this setup. In other words, if the Accounting share from Server 1 consumes about 800 megabytes (MB), Server 2 needs at least 800 MB of free disk space to keep a redundant copy.

As you can see from Figure 5.3, if users connect to the "Accounting directory in the Dfs structure, and unless that share is offline, they'll (in theory) connect to \\SERVER1\ACCOUNTING. If that share is offline, they'll automatically be redirected to \\SERVER2\BACKUP-ACCOUNTING without even knowing it.



I say “in theory” because Dfs is smart—it knows where things are located based on the site information defined in AD. If sites are defined within your organization, Dfs will use the site information to determine which share to connect to first. Therefore, if Server2 was local and Server1 was thousands of miles away, the user would first connect to [\\SERVER2\BACKUP-ACCOUNTING](#). However, although this was in the original design spec for Dfs, it didn’t work properly until SP1 was released.

It’s easy to see that the benefits of this technology are huge. It’s free, it’s built into Win2K, and it can keep your users up and running even when you have to take systems down for routine maintenance. Now that you’ve got a good overview of what Dfs can do for your organization, let’s walk through the steps involved to set up a Dfs structure.

Dfs Walkthrough

Setting up Dfs is so easy that (assuming there’s more than one server in your organization) it doesn’t make sense to not use it. For the remainder of this chapter, I’ll be using a sample scenario of a company with two offices, one in New York and one in Los Angeles. Each office has one server, and each server has a share on it called HOMEDIRS. The HOMEDIRS share holds all of the home directories for the local users at each office. Using this example, I’ll publish both shares into a Dfs structure, then make them redundant by creating backup copies and defining the backups as alternate replicas.

Defining a Dfs Root

Just like a hard drive has a root directory, your Dfs structure needs to have some sort of starting point as well. Your Dfs root will point to a share on at least one of your servers, so start by defining a share for Dfs to use as the root. Personally, I don’t like to put too many files into the Dfs root, and I usually use the naming convention `\\<Server_name>\DFSROOT` so that it’s obvious to me what the share is being used for.

A Dfs server can host only a single fault-tolerant Dfs root. Once you’ve defined a share for Dfs to use as its root, launch the Distributed File System MMC snap-in by selecting Start>Programs>Administrative Tools>Distributed File System. This will display the Dfs MMC, which probably won’t have anything in it. To start defining a Dfs structure, click Action, then select New Dfs Root. This will start the New Dfs Root wizard; the first step is shown in Figure 5.4.



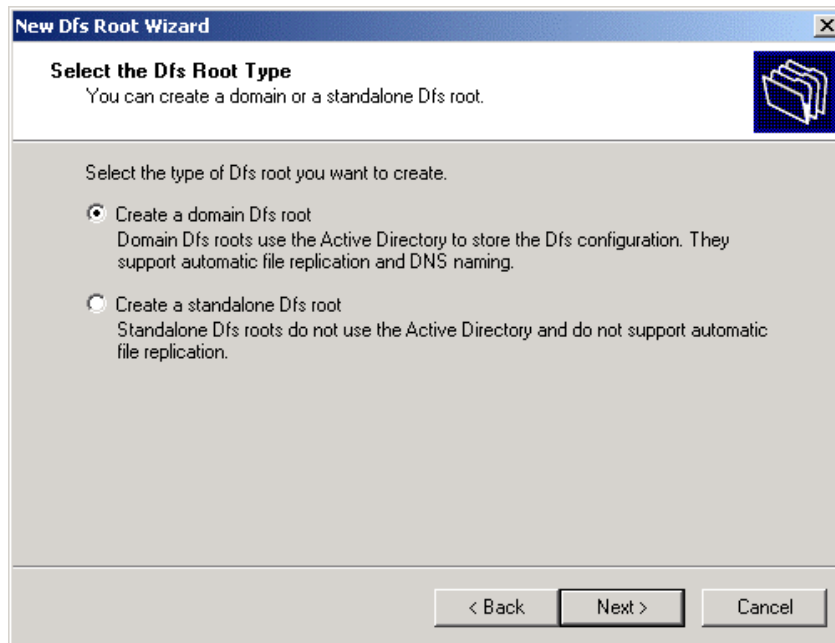


Figure 5.4: Configuring a new Dfs root.

For most purposes, you'll want to select Create a Domain Dfs Root because it enables file replication among shares across systems as long as the shares in the Dfs structure are on NTFS volumes. (Standalone Dfs roots don't support automatic replication, so they're not nearly as useful, and I won't be covering them in much detail.) Then press Next to proceed to the next step of the wizard, shown in Figure 5.5.

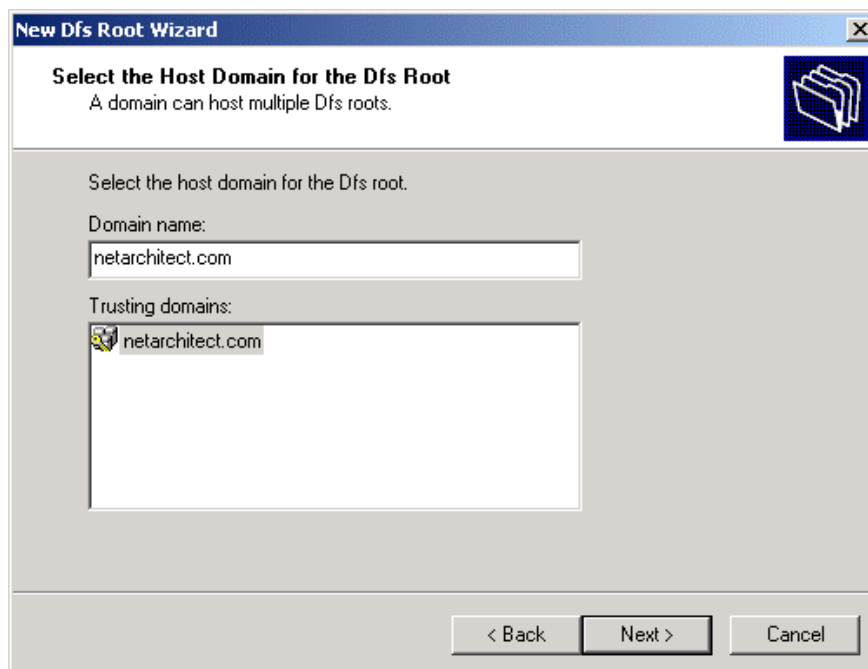


Figure 5.5: Defining a host domain for the Dfs root.

Because Dfs structures span multiple servers, the typical client-mapping mechanism of `\\Servername\Sharename` obviously won't work. Therefore, in Win2K, domain-based Dfs structures are accessed on client systems by mapping to the domain instead of to a specific server name. The root level of a domain-based Dfs structure of this kind is referred to as a *fault-tolerant root*. The UNC that your users will use to connect to the Dfs root can either reference the down-level NetBIOS domain name or the DNS domain name—for example, `\\DOMAIN\DFSROOT` or `\\DOMAIN.COM\DFSROOT`.

This first dialog box is where you'll define the domain name, which must be a valid domain within your AD. Select the appropriate domain, then click Next to go to the next step of the wizard, shown in Figure 5.6.

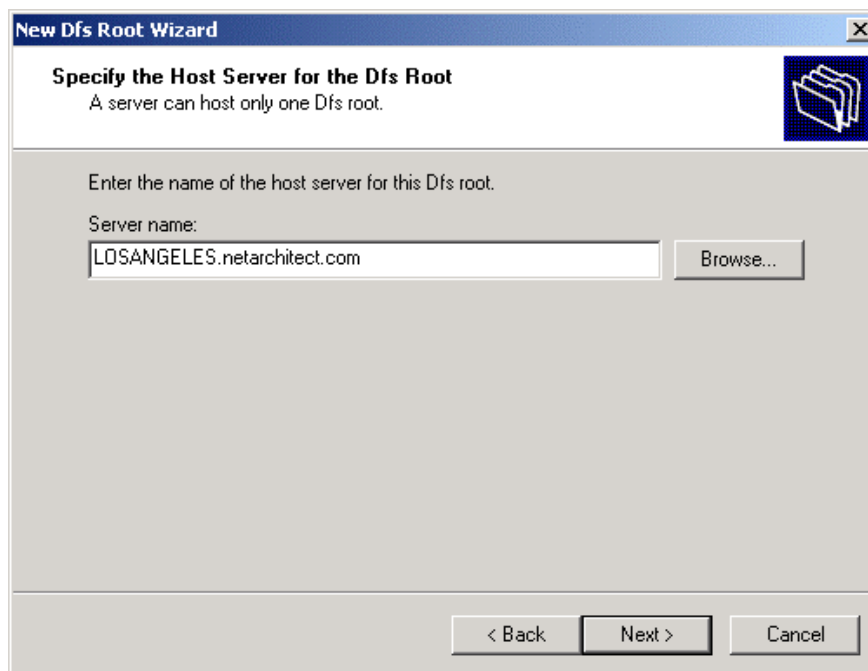


Figure 5.6: Choosing a server for the initial Dfs root.

As I mentioned earlier, the Dfs root must point to a share on a server somewhere, and this is where you'll define that initial share. Enter a server name by either typing in its full DNS name or by clicking Browse to find it. (As you can see in Figure 5.6, I've chosen the LOSANGELES server as the starting point for the Dfs root.) Then click Next to continue. The next step in the wizard is shown in Figure 5.7.

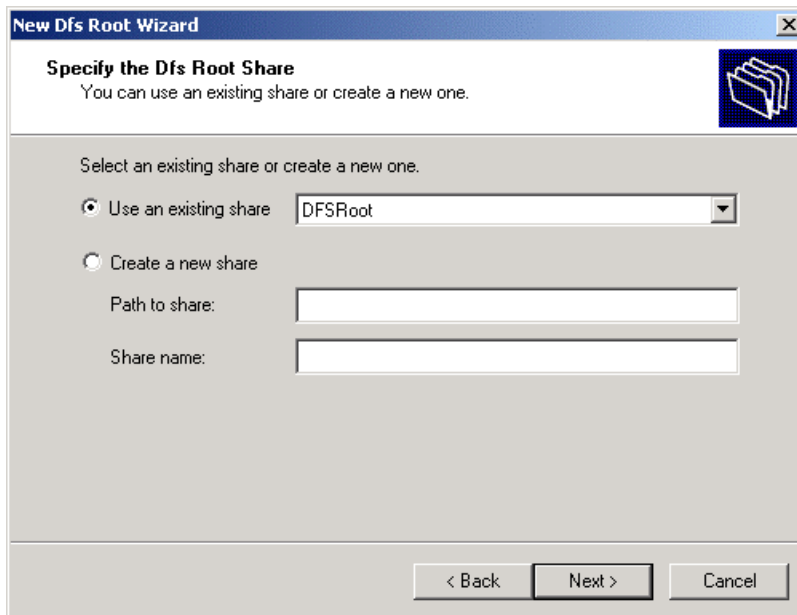


Figure 5.7: Choosing a share for the initial Dfs root.

If you've previously defined a share for the Dfs root on one of your servers, you'll select it here by choosing the correct share in the Use an Existing Share drop-down box. If you didn't create a share as your Dfs root, you can create one by selecting Create a New Share, then entering the appropriate path and share name. Once you've configured the share for the Dfs root, click Next to move on to the next step of the wizard, shown below in Figure 5.8.

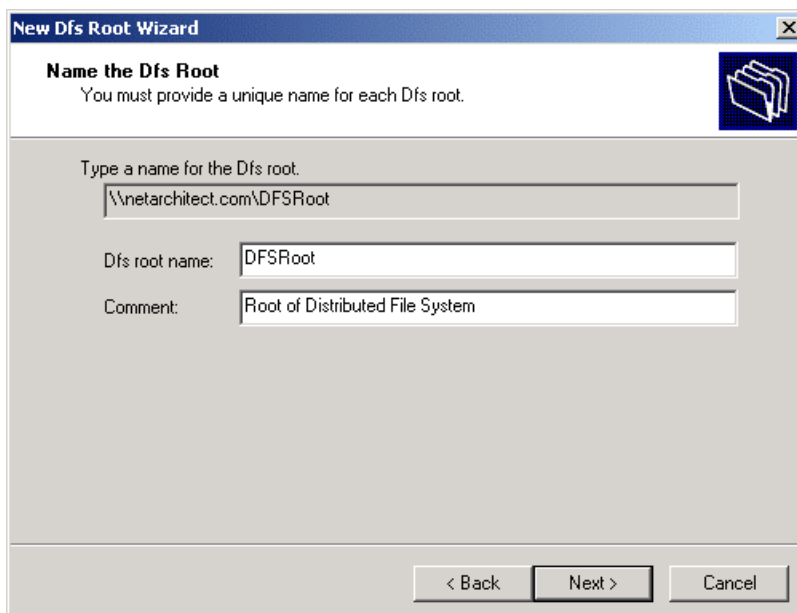


Figure 5.8: Choosing a name for the Dfs root.

As I mentioned earlier, typical server-based shares are accessed using the typical `\\Servername\Sharename` naming convention, and domain-based Dfs structures are accessed

using the `\\Domainname\Dfsname` naming convention. Earlier in the wizard, you defined the domain name that you'd use; here you'll choose the Dfs name.

Choose a functional name, one that will make it easy for your users to understand what is stored on that server. I recommend against choosing a name that will limit you in any way. For example, creating a Dfs structure named Documentation is suitable if the server is storing documentation files across all of your systems, but it won't be intuitive if you ever store any other files there, like Excel spreadsheets or PowerPoint presentations. Choose an appropriate name, then click Next, and Finish to complete the wizard.

This finishes the New Dfs Root wizard, at which point you'll have a functional Dfs root defined on your system. You can test it out by mapping a drive to it with the `\\Domainname\Dfsname` naming convention that you chose.

Making the Root Redundant

In my opinion, almost everything that you publish in a Dfs structure should be made redundant, and the root directory is no exception. As a matter of fact, the Dfs root is one of the most important pieces of the Dfs puzzle, so make it redundant across multiple systems if possible. If users can't map to the Dfs root, they won't be able to map to any of the other resources published in the Dfs structure (unless they do an old-fashioned mapping directly to the server).

To make the root redundant, you start by defining a Dfsroot share on the server, then define that share as a redundant root in your Dfs structure. (A redundant root is also known as a root replica.) Create the necessary file share, then right-click your Dfs root in the Dfs MMC. Select New Root Replica, which will start walking you through a wizard identical to the one you used to define the Dfs root.

In the first step of the wizard, you'll need to select the server that will house the replica share. In Figure 5.9, you can see that I've selected the NEWYORK server.

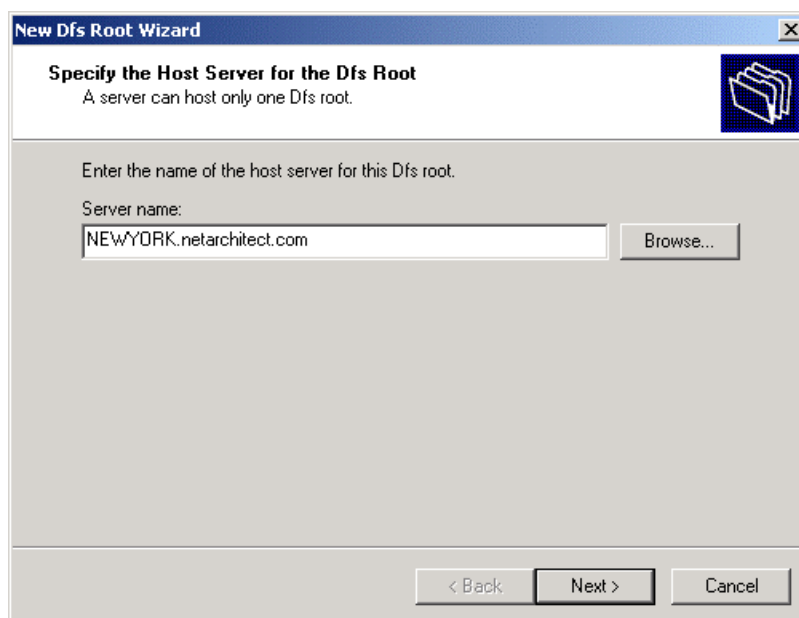


Figure 5.9: Choosing a server for a root replica.

After you've selected the appropriate server for your root replica, you'll need to choose a share as well, and you'll do it in the same way that you defined your original root share. Once you've defined your root replica share, you'll be asked to configure the replication parameters, as shown in Figure 5.10.

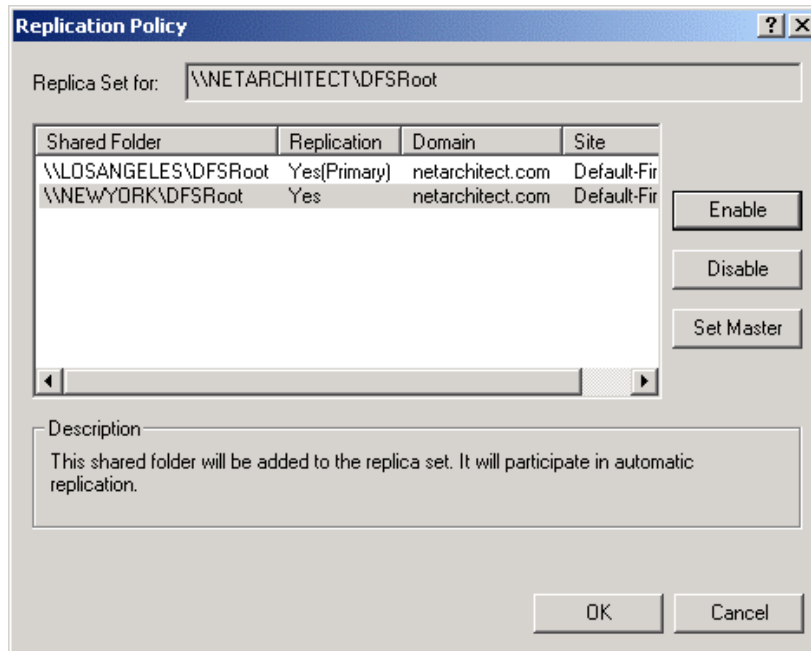




Figure 5.10: Configuring replication parameters.

As you can see in Figure 5.10, you define replication among shares by enabling replication on each one. Enabling replication on each share is as simple as highlighting each share, then clicking Enable. This toggles the Replication value for each share from *No* to *Yes*.

You'll also notice the *Primary* designation next to one of the shares. Highlighting a share, then clicking Set Master, selects this option. By defining a primary (or master) share, you tell Windows 2000 Server which share it should use as the initial source for the data. During the first synchronization between these two shares, all of the data from the master share will be copied to the replica share. Once all the data has been copied, the File Replication Service (FRS) will keep an eye on both locations and replicate any changes from one server to the other.


 When referring to the shared folders used in Dfs replica sets, Microsoft uses the terms *primary* and *master* interchangeably.

Once the initial synchronization is complete, the *Primary* designation will disappear from the synchronization parameters. When the copies are in sync with each other, there is no need for the designation anymore because any changes to either share will be replicated to the other.

 Make sure that you select the correct share as the initial master. Selecting the wrong share could result in lost data!

Following along with our example, you can see in Figure 5.10 that I've selected the share on the LOSANGELES server as the master share and the share on the NEWYORK server as the replica


share. Once I finish the root replica configuration, FRS will copy any files from the \\LOSANGELES\DFSROOT share to the \\NEWYORK\DFSROOT share, then keep them in sync with each other.

 FRS will sometimes take up to 15 minutes to begin the root replica synchronization.

Once you've completed all of these steps, you'll have a redundant Dfs root structure defined on your network. That's great, but a root directory by itself is about as useful as a C: drive without any subdirectories, so it's time to start putting some structure below the Dfs root.

Defining Dfs Links

After creating a Dfs root, the next item you'll want to define is a *Dfs link*. A Dfs link is simply a pointer to another set of shares. As far as your users are concerned, these links will look like subdirectories. Using our example, we're going to build two links, one called LA-HOMEDIRS and another called NY-HOMEDIRS. When users connect to the Dfs root on your network, they'll see two subdirectories there, LA-HOMEDIRS and NY-HOMEDIRS. When they navigate to one of those subdirectories, they'll be connected to the appropriate share on the appropriate server—without needing to know which server actually holds the files.

 We'll link the LA-HOMEDIRS link to the HOMEDIRS share on the LOSANGELES server, effectively renaming the share when we publish it in the Dfs structure. In this respect, Dfs is a handy tool for fixing any problems with naming standards in your organization: If naming standards vary from server to server, you can standardize them within Dfs while still preserving the actual share names.

To start defining Dfs links, right-click the Dfs root, then select New Dfs Link. This will display a dialog box, where you can create a new Dfs link for your system, as shown in Figure 5.11.

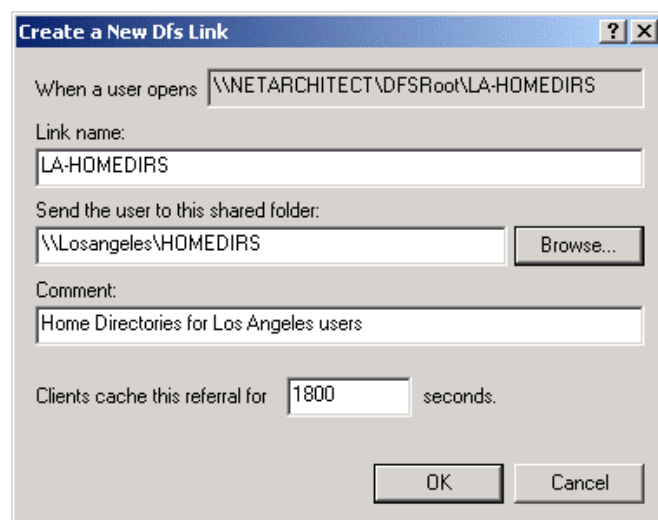



Figure 5.11: Defining a Dfs link.

In this dialog box, you'll tell Windows 2000 Server where to send your users when they navigate to the Dfs link. In our example, when users connect to the path \\DOMAIN\DFSROOT\LA-HOMEDIRS, they'll be transparently redirected to the file share named HOMEDIRS on the LOSANGELES server.

When clients navigate to a Dfs link, you can also specify how long they should cache this information—what server to connect to. If the user re-requests that information within the given time period, the client will automatically use the cached information instead of querying AD again. The default time for caching Dfs link information is 1800 seconds (30 minutes).

Once you've defined the Dfs link, it'll be immediately available in your Dfs structure. Following the same steps you used to create the LA-HOMEDIRS link, you can create one called NY-HOMEDIRS that will point to the home directory on the NEWYORK server.

By this point in our example, you should have a redundant Dfs root, along with links called LA-HOMEDIRS and NY-HOMEDIRS that will redirect users to the HOMEDIRS directory on the LOSANGELES and NEWYORK servers, respectively.

 Because of how Win2K handles Dfs replication, users sometimes won't receive the most up-to-date copy of a file. For example, if changes were written to the local server replica of a file and that server went down before replication occurred, a remote Dfs replica wouldn't contain those changes. If users then reconnected and were directed to a remote replica, they wouldn't see their changes. Although problems like these are less likely to occur since Microsoft fixed Dfs site affinity in SP1, you still need to be aware of them.

That's all great, but the fun is just about to start—making the home directory redundant.

Making a Redundant Dfs Link

Just like defining a Dfs root, defining a redundant Dfs link requires having another share available on another server for replicating the data. In the example we're using, we're going to create a share on the LOSANGELES server called NY-BACKUP and synchronize the data in order to back up the New York system. We'll do the opposite on the NEWYORK server, creating a share called LA-BACKUP in order to back up the Los Angeles system. Start in the Dfs MMC by selecting the Dfs link you want to make redundant. Right-click the link, then choose *New Replica*. A dialog box will appear, similar to the one shown in Figure 5.12.

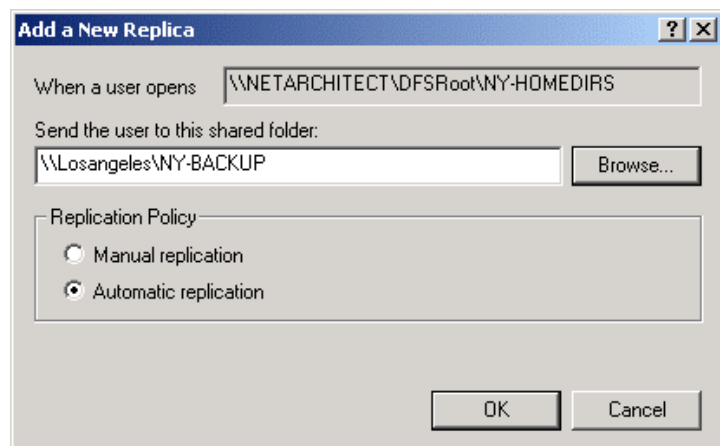


Figure 5.12: Adding a Dfs replica.

The items in this dialog box are rather straightforward; you simply need to define the file server and share for the replica of your data and whether to automatically synchronize the data among the shares. As you can see in Figure 5.12, I've told Dfs that the same content in the HOMEDIRS

share on the NEWYORK server can also be found in the LA-HOMEDIRS share on the LOSANGELES server. Clicking OK will take you to the next step of defining a Dfs replica, shown below in Figure 5.13.

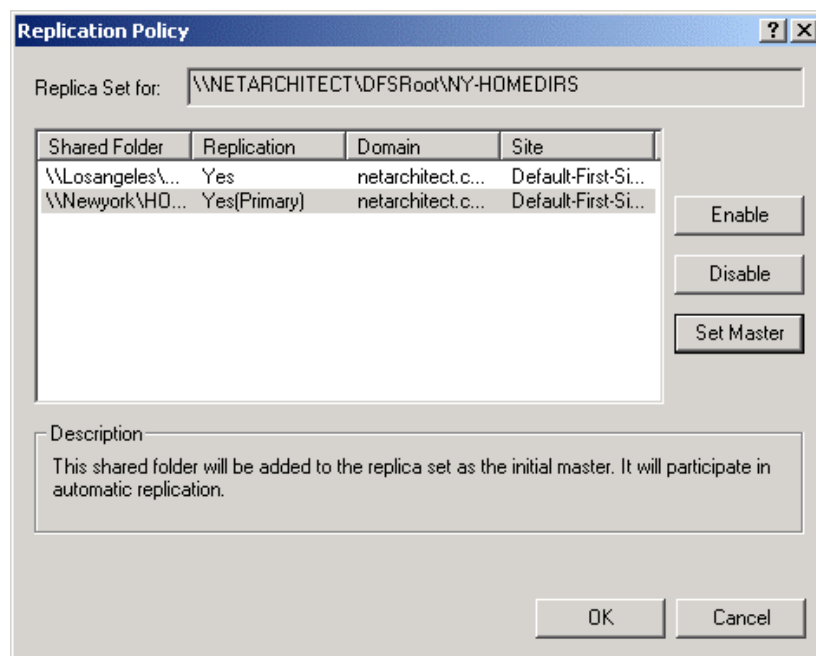


Figure 5.13: Configuring Dfs replication.

Just as you defined a replication policy for your Dfs root, so you also need to define a policy for the shares in your Dfs links. The dialog box at this step functions in the same way as the dialog box for Dfs replication: You need to enable the shares that will participate in the replication process, then set which share will be used as the master share for the first replication. This is critically important when you're using directories that actually have content in them, so make absolutely sure that you've set your master server correctly!

As you can see in the dialog box, I've defined that the share on the NEWYORK server will be the master (or primary) for this Dfs link. Once this dialog box is completed, FRS will dutifully start replicating the data from the New York server to the Los Angeles server within 15 minutes. Once the initial replication is complete, FRS will keep both shares in sync with each other. Repeat this process for the HOMEDIRS share on the LOSANGELES server: Configure the appropriate replica on the NEWYORK server, then enable the replication.

Congratulations—you've just built a redundant Dfs structure! Once you set up this type of configuration, your users should always be able to access their home directory, even if their primary server is offline. In our sample scenario, if the NEWYORK server suddenly crashes, all of the New York users will immediately start using the replica of their data stored in Los Angeles. This process is completely transparent to the users; they won't know that their primary server has crashed and will simply keep working away on their data. Once the NEWYORK server comes back online, FRS will synchronize the files between the systems, and the New York users will be able to use their local server's copy of their data once again.

A Client's View of Dfs

As we discussed “Dfs Walkthrough” earlier in this chapter, your client workstations connect to a Dfs structure by mapping a drive letter to your domain name instead of to a server name, then the appropriate Dfs name. As you can see in Figure 5.14, I’ve mapped to the Dfs structure called Dfsroot in the NetArchitect.com domain (`\\NETARCHITECT.COM\DFSROOT`).

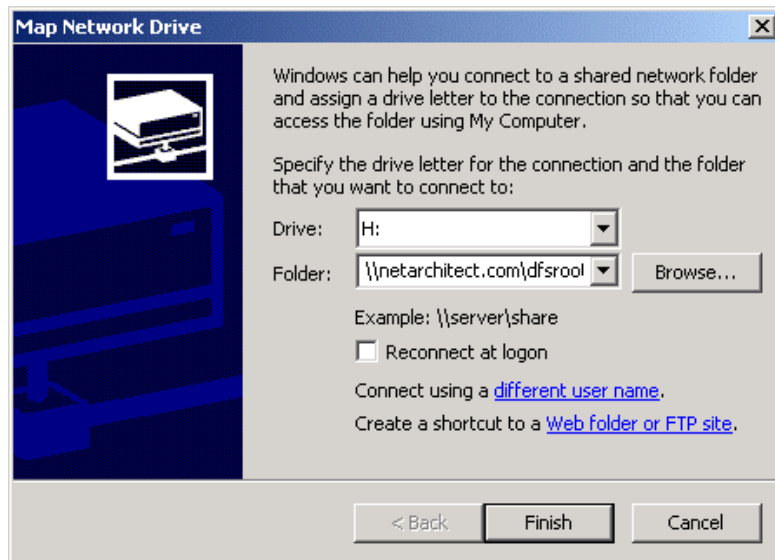


Figure 5.14: Mapping a drive letter to a Dfs structure.

When a Dfs-capable client computer sees this type of drive mapping—to a domain instead of to an individual server—it recognizes that this is a Dfs mapping and queries AD for the share (or list of shares) that it should connect to, to access the Dfs root.

Based on the responses that the Dfs client receives, it’ll then use a bit of logic to try to determine which share is the closest. It’ll compare its own site information in AD (if any has been defined) to the site information for the share(s) in the Dfs root. If one of the shares is obviously in the same site, the client will use that share by default. This is a great feature because it allows you to reduce WAN traffic while providing users with quicker access to files that might otherwise be stored hundreds or thousands of miles away.

Once the client has determined which server to connect to, the Dfs root will be available on the system, filled with links to other shares in the Dfs structure. For the example we’ve been walking through, this will look somewhat like the Explorer window shown in Figure 5.15.

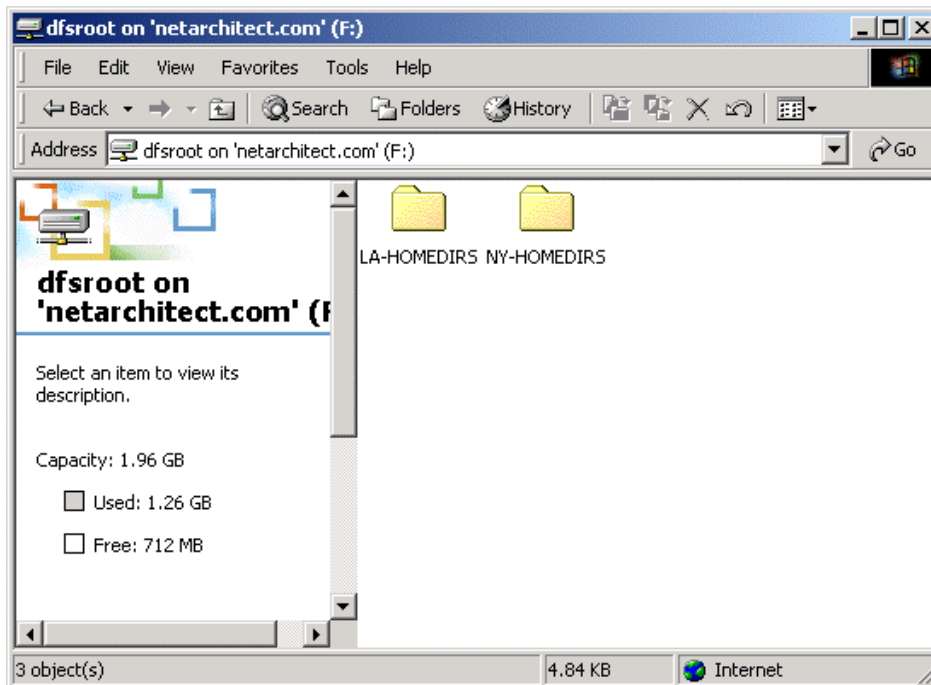


Figure 5.15: View of a Dfs root from a network client.

As you can see, our two Dfs links—LA-HOMEDIRS and NY-HOMEDIRS—are showing up as directories in the Dfs structure. If a user clicks either one, the AD querying process begins again, and the client determines which share to connect to. If a New York user double-clicks the NY-HOMEDIRS directory, he or she is transparently connected to the \\NEWYORK\HOMEDIRS directory (assuming appropriate site information has been defined in AD). If the same New York user double-clicks the LA-HOMEDIRS directory, he or she is transparently connected to the \\NEWYORK\LA-BACKUP share. The opposite is true for a Los Angeles user.

Recommended Uses for Dfs

By this point, you're probably thinking that Dfs is a pretty cool thing, something that can make your life as an administrator easier. You're correct on both counts. So what are some possible uses for Dfs? Well, it would make a great tool in any one of the following scenarios:

Creating a Distributed Network of Backups for Users' Home Directories

Just like in the example you've walked through, you can use Dfs to make all of your users' home directories redundant. This can completely remove the need to announce scheduled shutdowns to your users because if you bring down one system that is defined in a Dfs replica, alternative systems will continue to handle users' requests.

Organizing Disparate Shares and Naming Conventions into One Logical Structure

If you've ever worked with a network that has been running for a few years, you may have had the pleasure of working with systems that depending on which server you looked at, used different naming conventions for their shares. Your users can find it quite difficult to remember

what is stored where. Dfs allows you to consolidate all of those shares into one logical structure and correct any naming-convention “sins.”

Publishing a Distributed Web site

One of the coolest uses for Dfs is as a means of publishing a Web site that has HTML content distributed across your entire enterprise. For example, maybe your Sales Department is located in one state and is responsible for publishing a portion of the company’s Web site. Your accounting department might be located in another state and might be responsible for publishing your company’s financials on the company Web site. Each of those departments can be given access to a local share on their server for developing its department’s HTML content.

You can then join all of these separate directories together into one logical Dfs structure (perhaps called \\DOMAIN.COM\WWW) for Microsoft’s Internet Information Server (IIS) process to use. This would provide your Web site with some fault tolerance and protection—if one of your departments made a mistake and wiped out all of its HTML content, it wouldn’t take out the entire Web site, just its own area.

Avoiding Service Outages that Result from Scheduled Shutdowns

Even if you don’t choose to use Dfs on a regular basis, you can implement it when you need it—for example, when a scheduled shutdown is coming up. A few days before the shutdown, you can define a Dfs structure and enable file replication. However, to take advantage of the redundancy, your client workstations will need to access the files using the Dfs structure; if they use a typical server-based drive mapping, Dfs won’t do them any good.

Replicating Profile Directories

It goes without saying that users, in time, end up with some very large user profiles. If your users have to log on to the network while they’re away from the office, it can take a while for their profiles to come across the WAN, and logging on can take longer than usual. One way to work around this is to replicate users’ profile directories across all of your servers. That way, no matter where users log onto the network, they always have a local copy of their profiles available to them.

As you can see, Dfs has a number of possible uses. I’m sure you can think of more that will fit well within your organization.

Technical Overview of Dfs

If you’re at all like me, you’re the type of person who has to take apart technology to find out exactly how it works “under the hood”. If that’s the case, you’re probably wondering just how Dfs works its magic. Let’s find out exactly how Dfs does what it does.

AD Integration

As I mentioned earlier in this chapter, domain-based Dfs configuration information is replicated across enterprise networks as part of AD. Because of this integration, Dfs can leverage the multimaster replication already available in AD. Because AD-enabled clients always know where to find a domain controller, Dfs information is always readily available.

Dfs configuration information is stored in AD as a single object called a *Binary Large Object*, or *BLOB*. Because the entire configuration is stored in one item, the entire configuration must be replicated once any changes are made to the Dfs structure. Therefore, if you change the Dfs configuration, some clients may not see the changes for anywhere from 5 to 15 minutes, depending on which domain controllers they're referencing for their AD information.

File Replication Service

In my opinion, the real beauty of Dfs is how it uses FRS in Win2K. Because FRS represents such a critical part of Dfs' capabilities, there are several items about FRS that you need to know.

What's Replicated, What Isn't

Although FRS does a pretty good job of keeping files synchronized among servers, it's important to understand what FRS synchronizes and—what's more important—what it doesn't. FRS synchronizes the following items among servers:

- File data
- File permissions
- File security

The important thing to recognize is that FRS will synchronize the security and permission settings among your systems, so if you're replicating a set of users' home directories, the correct security will be synchronized to all the other servers participating in a replica. You won't be need to re-apply security on each system.



The current implementation of Dfs has a limitation of 256 replicas for a given link. File data is replicated at the file level, so it's important to realize that if a user makes a 10-byte change to a 20-MB file, the entire 20-MB file will be replicated to all other systems participating in the Dfs replica. This is a significant consideration if your organization is using a WAN with limited bandwidth among sites.

Before you start thinking that FRS will do everything, it's important to realize what FRS will *not* synchronize:

- Encrypted files (that is, using EFS)
- Any files beginning with a tilde character (~)
- Any files ending with .bak or .tmp
- NTFS reparse points of any kind
- File locks
- Files stored on FAT volumes

Now, the reason that some of these items won't synchronize is pretty obvious, but the impacts are important to recognize. For example, the fact that file locks aren't synchronized is a significant factor to consider when planning a Dfs structure. Lack of synchronization here means that two users could end up opening the same file on two different servers and working on it at the same time.



For example, let's assume that a user named Tom opens a 50-page Word document in a replicated Dfs structure and adds another 40 pages of content to it. Around the same time, Mary opens up the same Word document, but because she's in another state, she gets her copy of the document from her local server. There is no file lock on it because she's the only one accessing it on her local server, so she goes about her work. Instead of adding content, she deletes 15 pages from the Word document. Assuming that Tom finishes his work before Mary, the file synchronization goes somewhat like this:

1. Tom's copy of the document (the 90-page version after he adds his 40 pages of content) is replicated to all the other servers participating in the Dfs replica set. While Mary is still working on her document, it's changed on her local server. However, her copy of Word doesn't recognize that fact.
2. When Mary is finished, she saves her 35-page version of the document on the server in her state. Her changes are synchronized, so her 35-page copy of the document is replicated to the server in Tom's state, overwriting his 90-page version.

In synchronization terms, this logic is known as "Last writer wins." Unfortunately, there really isn't any way around this problem, but you can design your Dfs structure to minimize these types of circumstances. FRS watches the NTFS journal for changes to files. (Any change or replication of a file has an event time associated with it.) If a change needs to be replicated, FRS notifies a target server, which uses the following logic to determine whether to accept the file:

- If the event time for the incoming file is more than 30 minutes earlier than the event time for the local copy of the file, the receiving server discards the changes.
- If the event time for the incoming file is more than 30 minutes later than the event time for the local copy of the file, the receiving server accepts the new version of the file.
- If the event time for the incoming file is less than 30 minutes later than the event time for the local copy of the file, FRS checks the version numbers on the files to determine which is the more recent version. FRS creates version numbers and increments the numbers on replicated files each time they're closed.
- If the event time for the incoming file is less than 30 minutes later than the event time for the local copy of the file, and if the version numbers are the same, the 30-minute window is thrown out, and the most recent file is kept—whether it's the local copy or the incoming copy.

Based on this logic, you need to design your Dfs structure to minimize potential "Last writer wins" conflicts. Also, based on this logic, it should be obvious that Dfs isn't a suitable mechanism for replicating some types of data, such as client/server databases that are constantly open. Most likely, databases would never end up replicating unless you took the server service offline. Even if they did replicate, there would be so many versioning-control problems that the database would most likely become corrupted in some way.

Tuning Replication

Although file replication works mostly by itself, you may find that you want to fine-tune the process. For example, your organization might use an application that has a habit of creating temporary files with an .old extension. In that case, you might not want to replicate all of the .old

temporary files to your other systems, so you can add an additional filter to FRS to tell it to ignore those files.

To modify the configuration parameters of FRS, open the AD Users and Computers MMC snap-in and select View>Advanced Features from the MMC menu. This will let you view a number of additional AD components, including FRS (located under System), as shown in Figure 5.16.

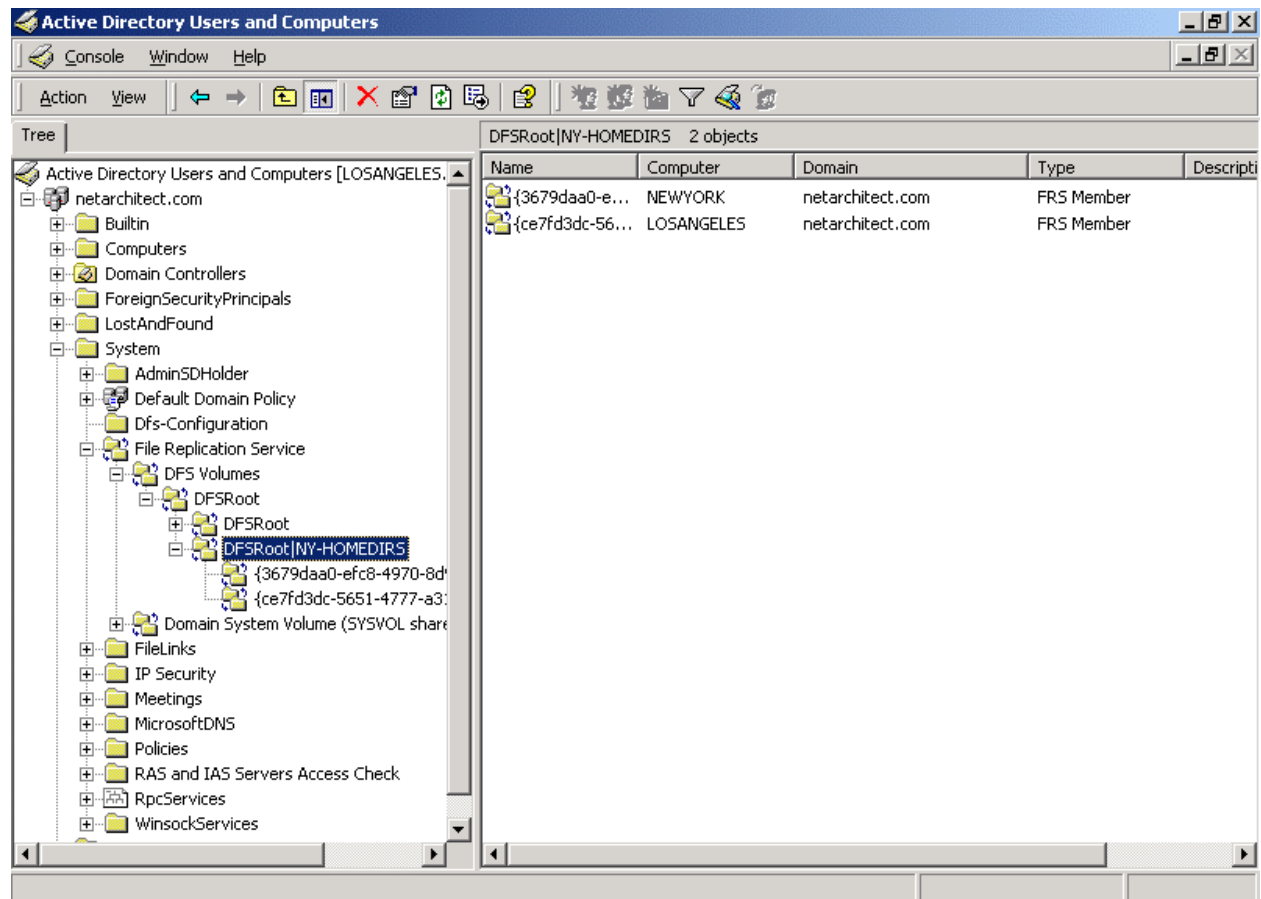


Figure 5.16: Modifying FRS parameters using the AD Users and Computers MMC snap-in.

As you can see in the structure shown in this figure, you can navigate to each Dfs link in your Dfs structure. You can set individual replication parameters and filters for each link by modifying the properties of the item. For example, modifying the properties of the NY-HOMEDIRS Dfs link in our example displays the property page shown in Figure 5.17.

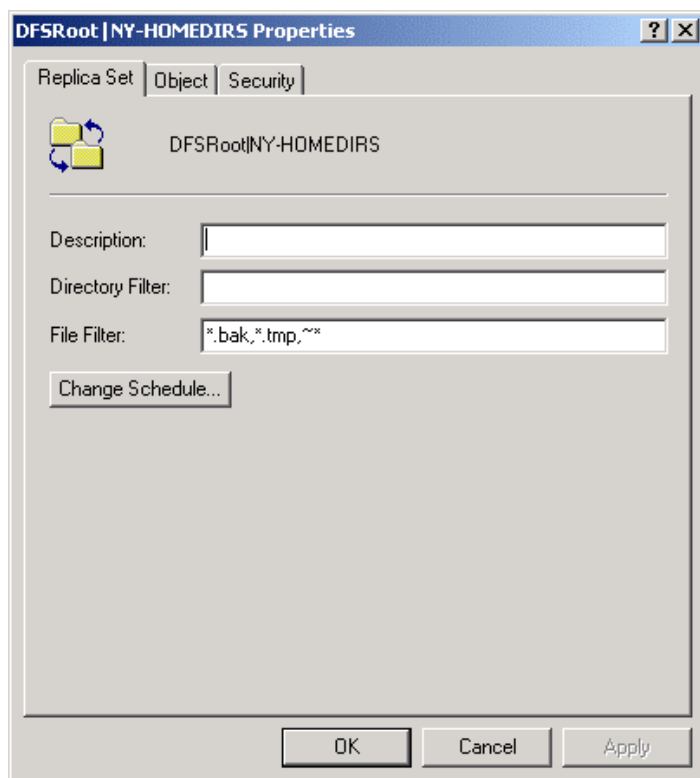


Figure 5.17: Replication parameters for a Dfs link.

As you can see in this property page, file filters are automatically set for *.bak, *.tmp, and ~* files. These filters exclude the temporary files of most applications; to add additional filters, add new wildcard file names to the File Filter text box.

You can also modify the replication schedule. For example, maybe you have a Dfs link that doesn't change very often—maybe your company publishes updated sales figures to all of your offices every night. In this case, constant replication might not be necessary, so you can set replication simply to occur overnight, after the new figures are available. To change the replication schedule for a Dfs link, click Change Schedule. By default, all Dfs replicas are synchronized 24 hours a day, seven days a week, at 15-minute intervals.

☞ Win2K's Dfs auto-builds a replication topology for propagating changes to Dfs replicas across the enterprise, but given the variations in bandwidth that may exist among links, this topology isn't necessarily ideal for every network. To accommodate this, you can also modify the Dfs replication topology by adding or removing NTFRS connection objects. For information about how to do this, check out Microsoft Support Knowledge Base article Q224512 at <http://support.microsoft.com>.

Troubleshooting Dfs

In most circumstances, Dfs is pretty reliable and usually doesn't need much troubleshooting. Assuming that your Dfs structure is sound and that all your servers can communicate with each other, you probably won't run into too many problems. However, on the off-chance that you do have trouble with Dfs, you can use the primary tools for troubleshooting:

- The Dfs event log

- FRS logging

Dfs Event Log

One of the best troubleshooting tools for Dfs and FRS is the event log. More often than not, you'll be able to find a number of helpful clues in the event logs that will point you in the right direction. You can see a sample file-replication event in Figure 5.18.

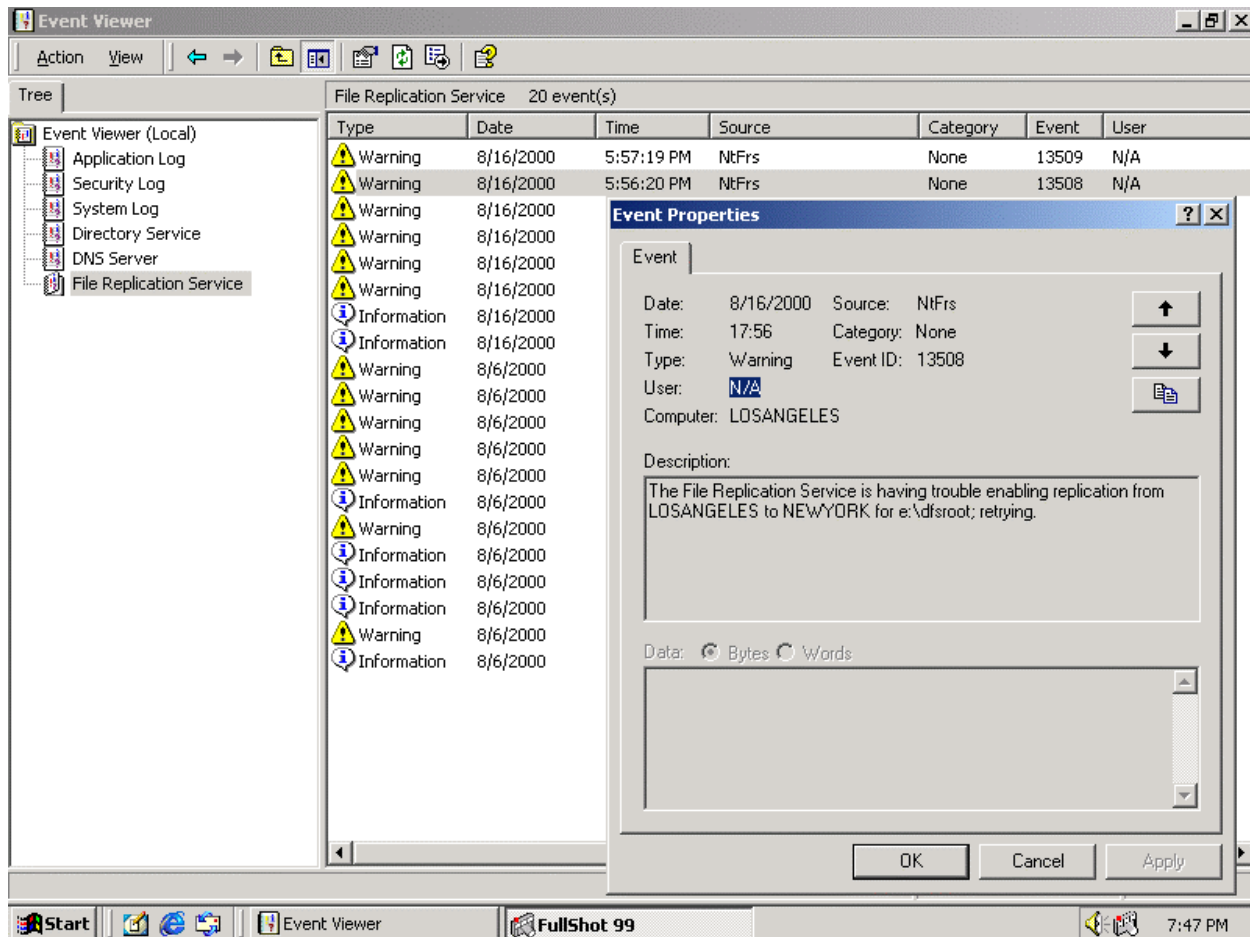



Figure 5.18: Sample event generated by FRS.

As you can see from the sample event #13508 shown, the LOSANGELES server is having trouble replicating the content for the Dfs root directory to the NEWYORK server. Now at the time this event was logged, the NEWYORK server was still starting up, which most likely caused this event. The event that followed it, event #13509, finally told me that replication was established between my two servers.

Microsoft's papers on FRS indicate that event #13508 is an indication of an RPC communication failure between the two replication partners. However, reports on the Internet seem to indicate that it's rather common for servers to come up with a few #13508 (failure) events every now and then. As long as they're eventually followed by corresponding #13509 (success) events, the general consensus is that they're not much cause for concern.

Another common FRS event is #13522 (Staging directory is full). If you see this message in your event log, refer to “FRS Staging” later in this chapter.

 If you find that simple event logging doesn't give you enough clues about problems with your Dfs structure, the command-line tool Dfsutil.exe can probably help. Using this utility, you can directly access the Dfs metadata information. You can add and remove roots, verify the metadata for a particular root, and even tell the utility to query specific controllers for information. This last capability is particularly helpful if some sites are seeing a Dfs root and others are not. Dfsutil is part of the Win2K Support Tools package, which can be found in the \Support\Tools folder of the Win2K CD-ROM.

FRS Logging

If you're having trouble with file replication but the event viewer isn't giving any helpful clues, you can dig a bit deeper by taking a look at the logging that FRS keeps by default. Win2K keeps detailed log files for FRS in the %systemroot%\debug directory. By default, Win2K will reserve five files—NtFrs_0001 to NtFrs_0005—to store the logging information generated by FRS.

The highest-numbered file contains the most recent data; therefore, in a default setup, NtFrs_0005 will contain the most recent log entries. Once that file fills up, it'll be renamed NtFrs_0004, and a new NtFrs_0005 file will be created. This process will continue until there are five full log files on your system. Once that has occurred, the oldest log file (NtFrs_0001) will be purged each time a new log file needs to be generated.

By default, FRS keeps a ton of logging information—almost too much to deal with. Below is a sample of the information that FRS logged as a file named logobw.bmp was copied into a replicated share in a Dfs structure. Because the file was copied into the share, FRS noticed that it needed to be replicated from one server to another. You can see from the descriptive tags associated with each line that FRS staged the item, then sent the file to the other server.

```
<StageCsCreateStage: ... FN: logobw.bmp    , [Stage Gen]
<StageCsCreateStage: ... FN: logobw.bmp    , [Stage Gen Steal OID]
<StageAcquire: ... FN: logobw.bmp    , [StageAcquire Entry]
<StageRelease: ... FN: logobw.bmp    , [Stage Release Entry]
<StageCsCreateStage: ... FN: logobw.bmp    , [Stage Gen Retire]

...

<RcsSendCoToOneOutbound: ... FN: logobw.bmp    , [Sending]
<OutLogSendCo: ... FN: logobw.bmp    , [OLSend DFSROOT|NY-HOMEDIRS\
> NETARCHITECT\NEWYORK$ RemoteCxt]
```

FRS Staging

When FRS begins the replication process, it copies a file into a *staging area* on the server, then replicates the staged file to another staging area on all of the other systems participating in the replica. After the file has been staged on the target server(s), the target server(s) will move the file into its final directory.

Although this might not make sense at first, the reason for these staging areas is that all of the systems participating in a Dfs replica might not be online at the same time. For example, a file might be changed on Server A while Server B is offline. Before Server B comes back online, the file might be changed a second time. To be true to the replication process, both versions of the changed file need to be replicated to Server B once it comes back online. In most instances, the staging areas shouldn't be a concern for administrators, but in certain circumstances, they may cause problems.

First, Win2K has a habit of choosing any old drive letter for the staging area. In some cases, if Win2K chooses a drive that doesn't have a lot of space available on it, this might be a problem. Therefore, it's important to know which drive is being used for the staging area. You can find out by looking at the root of all of the volumes for a hidden directory called FRS-Staging.

Now, by default, FRS uses no more than 660 MB of disk space, but if the volume starts to run out of space, you might encounter erratic replication problems. It's important to know where the FRS staging area is so that you can manage the space on that volume appropriately.

As far as staging areas for inbound files are concerned, FRS creates a directory called `DO_NOT_REMOVE_NtFrs_PreInstall_Directory`. Because files that appear in this directory are inbound files, there is no need to wait before moving them to their final destination. Once the server has received the files, it'll quickly move them to their final location in the Dfs structure.

A second problem you might run into is files getting "stuck" in the staging area. This can happen for a number of reasons, but it's usually a result of creating or deleting Dfs replicas that haven't fully propagated yet or because of slow links between replicas. For example, a 128KBps link between two offices might have trouble keeping up in real time with a very active Dfs structure. If the files are stuck because of slow links between replicas, you either need to increase the staging area or decrease the frequency of replication between systems, if possible (for example, stage all replications to occur during off-hours).

If you find that files are getting stuck in the staging area but all of the Dfs replicas on all of your servers are correct, you'll have to purge the files by hand. This is recommended only if the files that are stuck are taking up a significant amount of space. As long as they're several days or weeks out-of-date, you can probably assume that they're left over from an FRS failure and delete them. In my opinion, unless it's a major issue, consider just leaving them where they are.

Dfs and FRS Registry Tweaks

To help you fine-tune replication, a number of registry keys are available that change FRS behavior. The keys listed below can all be found (or should be set if they don't exist) in the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\NtFrs\Parameters` section of the Registry.

- **Debug Log Severity**—Five levels of logging can be used for FRS, numbered 0 through 5. Level 0 represents the least amount of logging—only the most significant messages will be kept. Level 5 represents the maximum amount of logging—every single thread and message will be logged. The default is set to 4, which I've found to be rather exhaustive, more than I've ever needed. You may want to try level 3 instead.

- **Debug Log File**—This value manages the number of log files (NtFrs_?????.log) that the system will keep. By default, this is set to 5. Change this value according to how much logging you intend to keep, the amount of replication that will be taking place on your network, and the importance of checking logging data.
- **Staging Space Limit in KB**—This value controls the maximum amount of storage space the staging area can consume. By default, this is set to a hexadecimal value of A5000, which translates to 660 MB of disk space. If the staging area fills up and interrupts replication among systems, either because of slow downstream partners or other issues, try increasing this value.
- **Working Directory**—FRS uses a jet database, along with some associated log files, to keep track of what it's working on. If you need to change the location of these files, change the working directory value. By default, this is set to %Systemroot%/Ntfrs.

Copyright Statement

© 2001 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at info@realtimepublishers.com