Realtime publishers

The Definitive Guide™ To

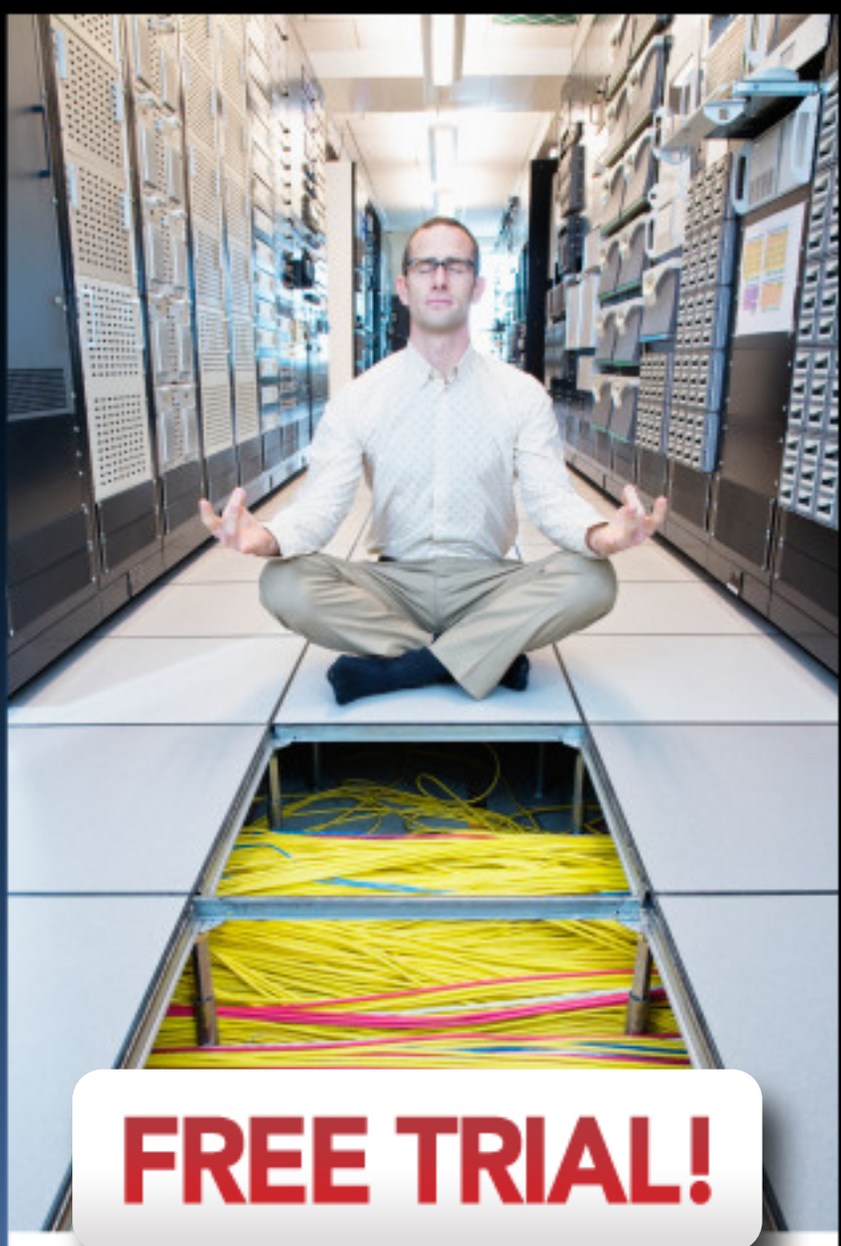
Windows Application and Server Backup 2.0

sponsored by

Don Jones

How can this guy be so calm when a mission-critical server just failed?

LIVE RECOVERY with Replay 4



Find out how you can be up, even when you are down.

FREE TRIAL!

www.appassure.com
703-547-8686

Replay4

Introduction to Realtime Publishers

by Don Jones, Series Editor

For several years now, Realtime has produced dozens and dozens of high-quality books that just happen to be delivered in electronic format—at no cost to you, the reader. We've made this unique publishing model work through the generous support and cooperation of our sponsors, who agree to bear each book's production expenses for the benefit of our readers.

Although we've always offered our publications to you for free, don't think for a moment that quality is anything less than our top priority. My job is to make sure that our books are as good as—and in most cases better than—any printed book that would cost you $40 or more. Our electronic publishing model offers several advantages over printed books: You receive chapters literally as fast as our authors produce them (hence the "realtime" aspect of our model), and we can update chapters to reflect the latest changes in technology.

I want to point out that our books are by no means paid advertisements or white papers. We're an independent publishing company, and an important aspect of my job is to make sure that our authors are free to voice their expertise and opinions without reservation or restriction. We maintain complete editorial control of our publications, and I'm proud that we've produced so many quality books over the past years.

I want to extend an invitation to visit us at http://nexus.realtimepublishers.com, especially if you've received this publication from a friend or colleague. We have a wide variety of additional books on a range of topics, and you're sure to find something that's of interest to you—and it won't cost you a thing. We hope you'll continue to come to Realtime for your educational needs far into the future.

Until then, enjoy.

Don Jones

Copyright Statement

Chapter 1: Introduction

The first backup—technically—was around 1951, when the first generation of digital computing appeared in the form of UNIVAC I. The "backups," such as they were, were the punch cards used to feed instructions to the massive machine. Once computers began to use more flexible forms of storage, reel-to-reel magnetic tape began to replace punch cards. In 1965, IBM introduced the first computer hard drives, although through the 1970s, these devices remained impractically expensive to use for backups. Floppy disks came into use in 1969—an 8-inch monster storing just 80 kilobytes of data. Recordable compact disks became available in the early 1990s, and flash drives became common in the early part of the 21st century. Shockingly, magnetic tape—the second-oldest form of backup storage—is *still* in use today. Figure 1.1 shows a timeline of data backup storage (excerpted from www.backuphistory.com), and you can see that tape is still alive and well—and has been for almost 50 years.

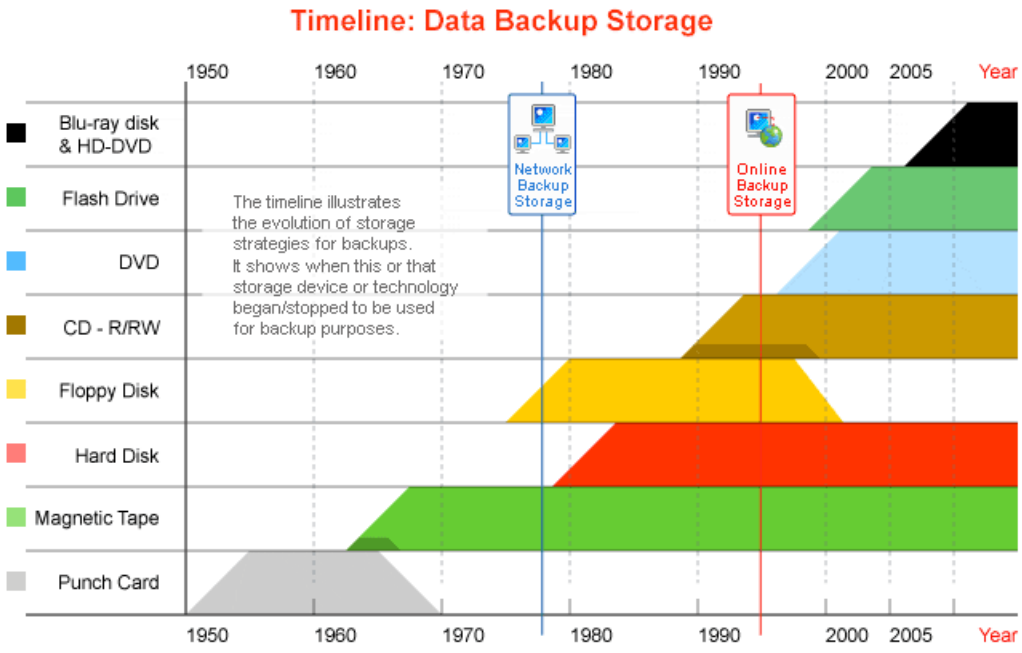


Figure 1.1: Backup storage timeline.

There's an interesting parallel to be drawn here: Despite numerous technical advances in storage, we continue to rely on one of the oldest mediums to store backups. The same applies to our backup techniques and procedures: Despite advances in *how* we perform backups, we tend to still use the same decades-old techniques, albeit wrapped up in pretty new tools.

Throughout computing history, backups have been practical, simple procedures: Copy a bunch of data from one place to another. Complexities arise with "always-on" data like the databases used by Exchange Server and SQL Server, and various techniques have been developed to access that form of in-use data; however, backups have ultimately always been about a fairly simple, straightforward copy. Even magnetic tape—much more advanced than in the 1960s, of course—is still a primary form of storage for many organizations' backups.

I call it "Backup 1.0"—essentially the same way we've all been making backups since the beginning of time, with the only major changes being the storage medium we use. Although many bright engineers have come up with clever variations on the Backup 1.0 theme, it's still basically the same. And I say it's no longer enough. We need to re-think *why* we do backups, and invent Backup 2.0—a new way to back up our data that meets today's business needs. Surprisingly, many of the techniques and technologies that support Backup 2.0 already exist—we just need to identify them, bring them together, and start using them.

The Philosophy of Backup

Let's start with the question, "Why do we back up?"

I suppose the first answer that comes to mind is simple enough: *So that we don't lose any data.* But that's not actually an accurate answer. Personally, I don't back things up *just* so they will never be lost; I back them up so that *I can continue using them.* That's a subtle difference, but an important one. If you're only concerned about never losing data, Backup 1.0 is probably sufficient: Copy your data to a long-term storage medium—probably magnetic tape—and stick it in a vault somewhere. You could call it *archiving* and be more accurate, really. But most organizations aren't as concerned about archiving as they are about making sure that data remains available, which means you not only need a *backup* but also a means of *restoring* the data to usability. So the real answer, for most organizations, is more complicated: *So that our data remains available to us all the time.*

That's where Backup 1.0—the backup techniques and technologies we've all used forever and are still native to operating systems (OSs) like Microsoft Windows—can really fail. Making a copy of data is one thing; putting that copy back into production use is often too slow, too complicated, and too monolithic. And that's where Backup 1.0 fails us. As we start considering Backup 2.0, and what we need it to do, we need to bear in mind our real purpose for backing up. Ultimately, we don't care about the backing up part very much at all—we care about the *restore* part a lot more.

Why Backup 1.0 Is No Longer Enough

Our decades-old backup techniques are *not* sufficient anymore. They may be great for creating backups—although in many cases, they aren't even good for that—but they do not excel at bringing the right data back into production as quickly as possible. Despite advances in specialized agents, compressed network transmissions, and so forth, we're still just making a copy of the data, and that doesn't always lend itself well to *restoring* the data. Why?

Backup Windows

One problem is the need for *backup windows,* periods of time in which our data isn't being used very heavily, so that we can grab a consistent copy of it. *Consistency* is critical for backups: All the data in any given copy needs to be internally consistent. We can't back up half a database now and the other half later because the two halves won't match. As our data stores grow larger and larger, however, getting a full backup becomes more and more difficult.

Microsoft's TerraServer, which stores and provides access to satellite photographs for the entire United States, has a data store in excess of 1 terabyte, and even with fairly advanced backup hardware, it still takes almost 8 hours to back it all up. That's a total data throughput of 137GB per hour—but if that data were in constant use, it would become less practical to make a complete copy on a regular basis.

As a workaround, we commonly use differential or incremental backups. These allow us to grab a lot less data all at once, making it easier to make our backup copy. The problem is that these ignore the *real reason* we made the backup in the first place—to enable us to *restore* that data. Consider a common backup approach that uses SQL Server's native backup capabilities:

- Sunday, full backup
- Every weekday evening, a differential backup
- Every hour during the day from 8am to 5pm, a log backup

Those weeknight differentials grab everything that has changed since the last full backup (as opposed to an incremental, which grabs everything that has changed since the last full *or* the last incremental). If something goes wrong on Friday at 4:05pm, there's a lot of data to restore:

- Last Sunday's full backup—which will be fairly large
- Thursday's differential—which will also have grown quite large
- The log backups from Friday at 8am, 9am, 10am, 11am, noon, 1pm, 2pm, 3pm, and 4pm—that's nine files in all, although each will be fairly small

Figure 1.2 illustrates these different backup types.

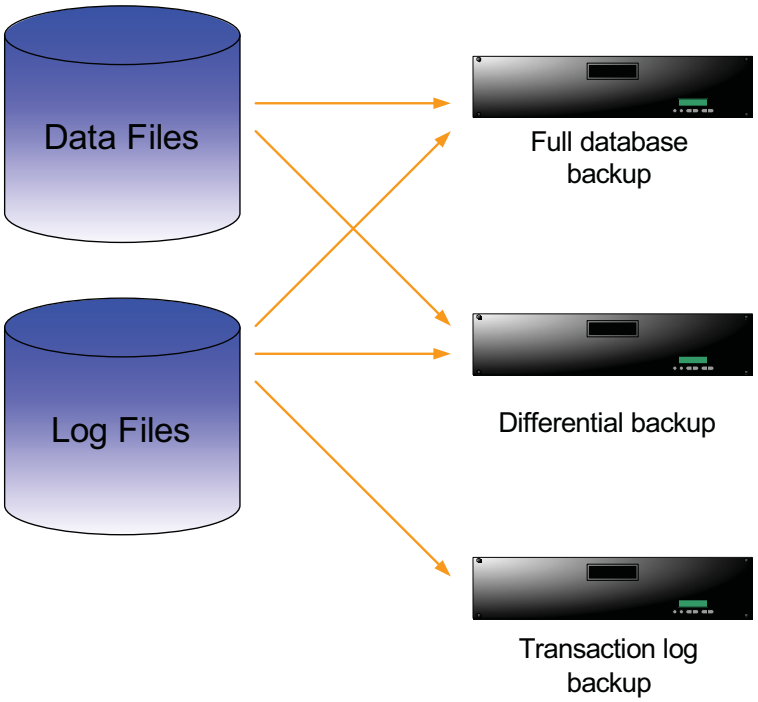


Figure 1.2: SQL Server backup types.

That's a ton of work—and a ton of waiting while tapes and hard drives spin, restoring all that data. Sure, we won't have lost much—just 5 minutes worth of work—but on a large database (say, a terabyte or so), you could easily be waiting for 16 hours or more.

And that's *if* the backups all work. Tape drives and even hard drives are not immune to corruption, errors, and failures, and one of the most common stories in the world is the administrator who realized that the backup tapes were no good—and realized it while trying to restore from one of them. We all know that we should test our backups, but honestly, do you do it? Out of more than 300 consulting clients I've worked with in the past 10 or so years, *one* of them had a regularly-scheduled plan to do test restores. One. Less than one percent. Why?

Well, the one customer who did regularly-scheduled test restores had a dedicated administrator who did almost nothing else. A full test restore of their environment, using Backup 1.0-style techniques and technologies, would require the entire IT team about a week to perform. That one administrator could test-restore various systems, one at a time, over the course of a month—and then start over. I think that pretty much answers the question about why so few people test their backups. It's a simple: Too much data for full backups results in workarounds such as differentials and incrementals that contribute to lengthy restore times, which is why we never bother to test—and why, when the rubber hits the road and we *need* those backups, *nobody* is happy about how long it takes to grab the needed data.

Non-Continuous

Backup 1.0 has another major weak point: It's always a point in time. A snapshot. *Non-continuous,* in other words. Consider one approach to backing up Active Directory (AD), which I see a lot of my customers using:

- Full backup of every domain controller's System State on the weekends. This is a quick, fairly small backup—even an exceptionally large domain can be backed up in a few minutes.
- On one or two domain controllers, a twice-daily backup of the System State. Again, this operation is quick.

The problem is that you always stand to lose a half-day's worth of work because you're only taking "snapshots" twice a day. What if you just imported a couple of hundred new users into the domain, added them to groups, and assigned them file permissions on various file servers? Losing that work not only means you have to start over, it also means you've got orphan Security Identifiers (SIDs) floating around on all those file permission Access Control Lists (ACLs). You'll not only have to repeat your work, you'll also have to clean up all those ACLs.

Businesses tend to design backup strategies around the concept of "How much data are we willing to lose, and how much work are we willing to repeat?" There's often a lot less consideration about "How quickly do we want to restore the data?" and much to my irritation, almost nobody thinks to answer, "We don't want to lose *any* data or repeat *any* work!" Backup 1.0 has conditioned us to accept data loss and repeated work as *inevitable*, and so we design backup schemes that tradeoff between the inevitable loss of data and the amount of backup resources we want to devote. Frankly, the attitude that I *have* to accept data loss and repeated work is nonsense. I can't believe that, a decade into the 21st century, we're all so complacent about that attitude.

Just the Data—Not the Application

Another problem with Backup 1.0 is that we often tend to just back up *data*—databases, files, System State, or whatever—we rarely back up the *applications* that use the data. I took a brief survey on my blog at ConcentratedTech.com, and about 95% of the respondents said they don't back up applications because they keep the original installation media so that they can always reinstall the application if necessary.

Really? Let's think about that: Microsoft Exchange Server takes about 45 minutes to an hour to install properly. Then you have to apply the latest service pack—another half-hour or so—and any patches released since the service pack—call that another 20 to 30 minutes. *Then* you can start restoring the data, which may take another few hours. If you're rebuilding an entire server, of course, you'll have to start with reinstalling Windows itself, and its service pack and patches, which will add another 2 or 3 hours to the process. The total? Maybe a full work day. And for some reason, people find that acceptable—because Backup 1.0 is all about *archiving,* really, not *restoring.*

One valid counter-argument is that *most* restorations are for just the data, or even a part of the data (like a single database table, or a single email message), and aren't a full-on disaster recovery rebuild. Well, okay—but does that mean it's still acceptable for a full-on disaster recovery rebuild to take a full day? Typically not, and that's why some organizations will *image* their servers, using software that takes a snapshot of the entire hard drive and often compresses it to a smaller size. Used for years as a deployment technique, it works well for backing up an entire server. For *backing up* the server but often not *restoring it.* Snapshot, or point-in-time, images take time to produce, and the server may even have to be shut down to make an image—meaning you'll need a frequent maintenance window. A traditional snapshot image won't contain the latest data, so even after restoring the image, you still have to rely on traditional backups to get your most recent data back. It just amazes me that we accept these limitations.

Disaster Recovery Is Too Inflexible

Closely related to the previous discussion is the fact that people do backups for two different reasons. Reason one, which I think is probably more commonly cited as a reason to back up is to restore small pieces of data. You've doubtless done this: Restored a single file that someone deleted by accident, or an email message, or a database table. Nearly everyone has dealt with this, and it isn't difficult to "sell" this reason to management when acquiring backup technologies.

The second reason is for what I called "full-on disaster recovery" in the previous section. This is when an entire server—or, goodness help you, an entire data center—is lost, and has to be restored on-site or at a different location. Unfortunately, this level of disaster is actually quite rare, and it's a tough sell for management if the organization hasn't encountered this type of disaster in the past.

The ultimate problem is that Backup 1.0 technologies lend themselves to one or the other scenarios—but not usually to both. In other words, if you have a product that does great bare-metal recovery, it may not do single-item recovery as well. Some products compromise and do an okay job at both—backing up the entire server to enable bare-metal recovery (and often providing bootable CDs or other techniques so that you can initiate a bare-metal recovery), and then keeping a separate index of every backed-up piece of data to make single-item recovery easier. Frankly, I've not used many solutions that do a great job at both tasks—and the fact that they're all essentially snapshot-based still makes them pretty limited. I *never* want to have to agree that losing a certain amount of work is acceptable.

Backup 1.0: The Verdict

If you're just archiving data, Backup 1.0 is pretty awesome. It starts to fail, though, when you need to *restore* that data, and want to do so in an efficient manner that enables both single-item and full-on disaster recovery restoration. The snapshot-oriented nature of Backup 1.0 means you're always at risk of losing some work, and that snapshot-oriented nature also imposes rigid requirements for maintenance and backup windows—windows that might not always be in the business' best interests.

So let's rethink backup. I want to go back to basics and really define what backups *should* do. Consider this definition a wish list for Backup 2.0.

Backup Basics

I have no doubt that you're a pretty experienced administrator or IT manager, and you might not think that "backup basics" is a particularly enticing section. Bear with me. I'll try to keep each of the following sections succinct, but I really want to step back from the existing technologies and techniques to focus on what people and businesses—not software vendors—want their backup programs to do. We've been doing backups more or less the same way for so long that I think it's beneficial to just forget everything we've learned and done and start over without any assumptions or preconceptions.

Why Back Up?

We've covered this topic pretty well, but let me state it clearly so that there's no confusion:

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

That statement implies a few basic business capabilities:

- When a problem occurs, we want to experience as little data loss as possible
- We need to be able to recover data as quickly as possible
- We place equal importance on recovering a single bit of data as we do in dealing with a complete disaster

The statement also means a few things traditionally associated with Backup 1.0 probably aren't acceptable:

- Snapshots that grab only a certain point-in-time image are less desirable
- Any system that is weighted toward disaster recovery or toward single-item recovery is less desirable—we need *both* capabilities
- Any system that requires lengthy, multi-step restore processes is less desirable
- Backups that do not lend themselves to some form of physically protected storage are less desirable
- Backups that require hours and hours to complete will require hours and hours to restore—both of which are less desirable

So given *why* we back up, we can take a fresh look at *what* we back up.

What Do You Back Up?

Even the relatively primitive backup software included with Windows Server 2003 (and prior versions of Windows) understood that you don't always need to back up *everything* every time you run a backup. Figure 1.3 shows how that utility allowed you to select the items you wanted to back up or restore—a user interface (UI) duplicated in some form by most commonly-used backup software.

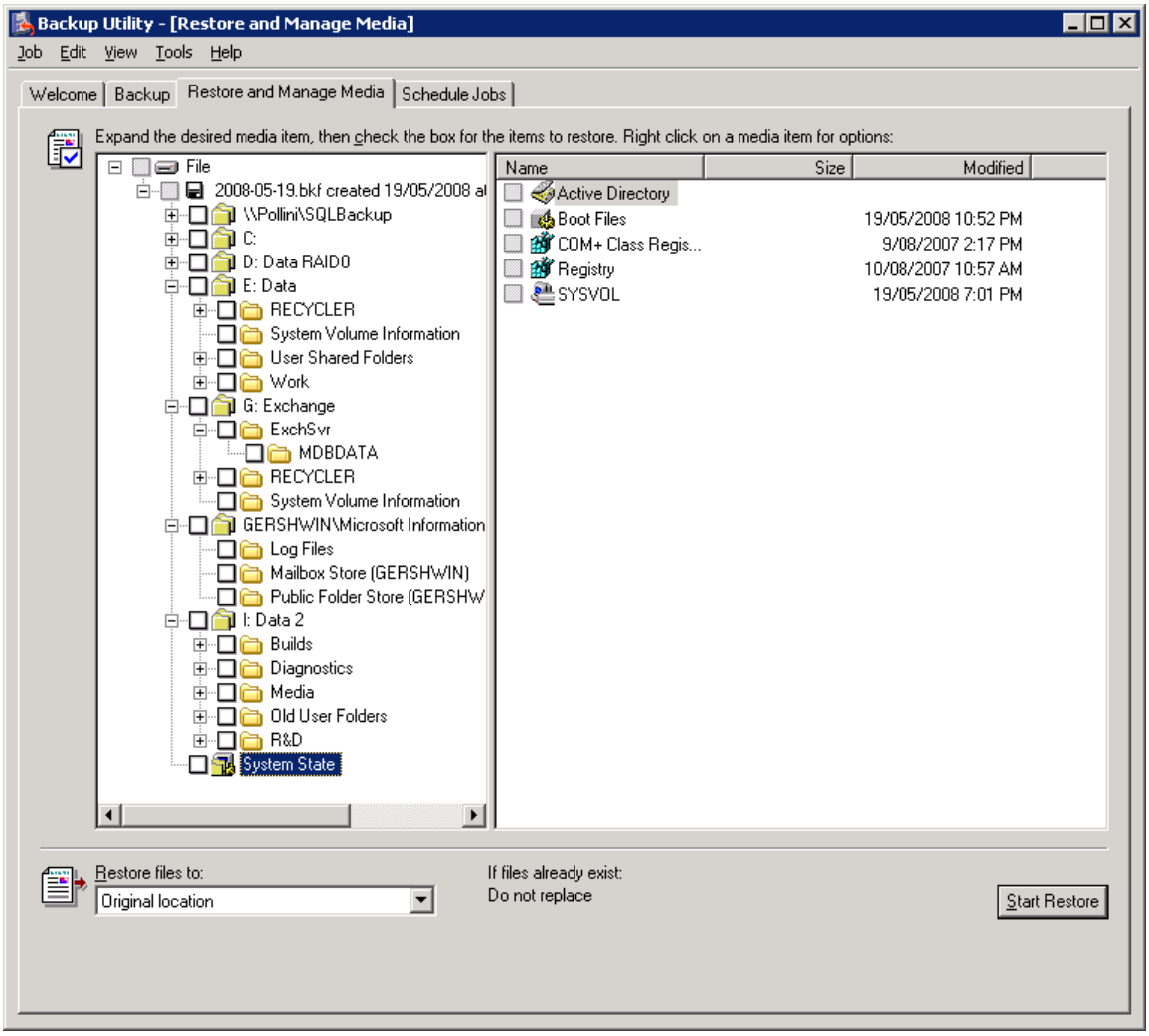


Figure 1.3: Selecting what to back up or restore in Windows Backup.

So what do you back up? On any given server, you have many choices:

- Back up the entire server—every file on every disk
- Back up just data—shared files, application databases, and so forth
- Back up applications and their data—including the applications' executable files and settings
- Back up the OS and its settings but not any application files or any data

The permutations are practically limitless. If you're simply after archiving—creating backups, that is—then anything that grabs the data is fine. In fact, *just* grabbing the data is probably all you need to do *if* all you want to do is create a point-in-time snapshot for archival purposes. Sure, you *could* back up the entire server—and if your goal is to be able to handle a full-on disaster *or* recover individual items, then backing up the entire server would provide both capabilities. But backing up the entire server would probably take a lot longer. It might not even be entirely possible because there will always be some open files, running executables, and other items that Backup 1.0-style backup techniques can't get to. So maybe backing up the entire server isn't really practical. After all, we can always reinstall the OS and any applications from their original media, right?

Wait, a second—let's go back to *why* we're backing up:

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

Okay, this statement clearly indicates that we need to grab the data, but that last phrase—"…have access to our data with as little downtime as possible"—adds something important. In order to *access* our data, we need the OS and associated applications to be up and running! A data backup is useless without an application and OS to restore it to. I've already explained why rebuilding a server using the installation media is so slow—you have to perform lengthy installs, then install service packs and patches, and *then* restore your data.

This tells me, then, that our backup *must* be of the entire server. After all, I'm not just here to archive my data—I also need to be able to restore it quickly, and get access to it quickly, and that means I need to restore the OS and any applications quickly. So regardless of practicality—for now—I'm going to say that backing up *the entire server* is the only way to go. I just need to figure out how to do it quickly.

When Do You Back Up?

How often will I be making backups? Under Backup 1.0, this was a real question. Often, you might take a full backup during an evening or weekend maintenance window, then grab smaller backups more frequently. When I started out in IT, I was an AS/400 operator. Every evening, we made tape backups of our most important data files from the AS/400; on weekends, we ran a full backup of the entire system. Backing up the data files took a few hours, and we had to do it in the evening after pretty much all the work for the day was finished because the data was unavailable while the backups were running (in fact, we pretty much kicked everyone off the system while backups were being done). The weekend full backups could take a full day, and everyone had to be offline then.

But maintenance windows are a Backup 1.0 concept, so let's disregard them. In fact, let's review our whole reason for being here one more time:

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

Right there is my answer: "…from losing *any* data or losing *any* work…." That tells me the Backup 1.0 method of point-in-time snapshots is useless because no matter how often I'm making incremental or differential backups, I'm still going to be at risk for losing *some* work or data, and that's not acceptable.

So if you ask, "When will you make backups?" I have to answer "Always." Literally—continuous backups. In fact, the industry already has a term for it: *continuous data protection.* Although specific techniques vary, you can think of this—very roughly—as being similar to RAID 1 drive mirroring. In a RAID 1 array, as Figure 1.4 shows, the drive controller writes blocks of data to two (or more) hard drives at once. Disk 2 is a complete, block-level backup of Disk 1. Restoring is fast—simply swap the two if Disk 1 goes belly-up.

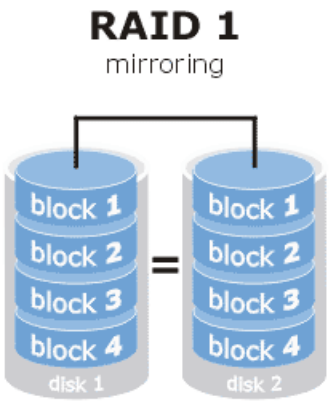


Figure 1.4: RAID 1 disk mirroring.

Of course, RAID 1 is good for a certain type of scenario but it isn't practical for all situations, and it doesn't meet all our backup requirements. For one, server disks are still pretty expensive. In a server using a large number of disks, mirroring every one of them isn't practical. Some organizations will set up a RAID 5 array and then set up a second array to mirror the first—but that can be incredibly expensive. A further problem is that the backup disks coexist with the primary disks, so the backup disks are still at risk for damage due to fire, flood, and other physical threats. Further, a mirror is only a backup for the *current* condition of the primary disk: You can't "roll back" to a previous point in time. So mirroring alone is a great tool for certain situations, but it isn't a complete backup solution. What it *is,* though, is a good idea for how to make continuous backups. We just need to leverage the technique a bit differently.

What Type of Backup Will You Use?

Full backup? Differential? Incremental? I have to say none of these because those Backup 1.0 terms are associated with point-in-time snapshots, and we're not going to use those. Instead, we're going to use a Backup 2.0 technique, borrowing from the RAID 1 concept of block-level mirroring. Part of continuous data protection, we'll call this *block-level backups.* Figure 1.5 shows how it might work.

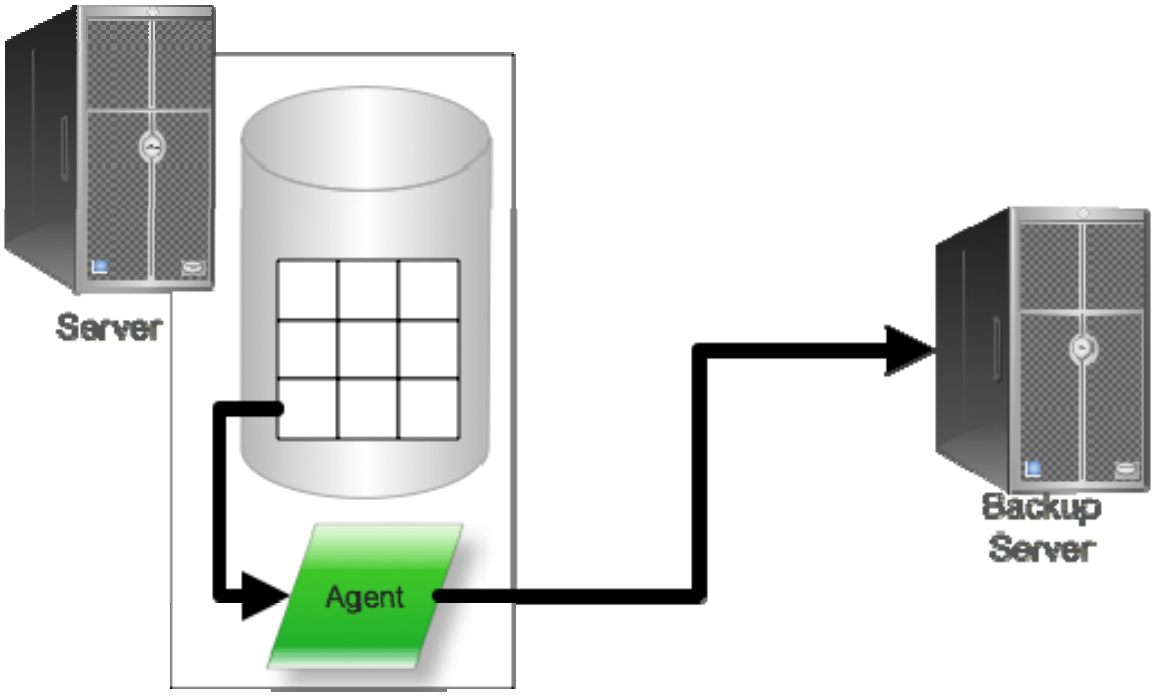


Figure 1.5: Agent-based block backups.

Here, a software agent of some kind runs on the server. As blocks are changed on disk, this agent transmits those blocks to a backup server, which stores them. That server might store blocks for many different servers. The agent would likely tap into a very low level of the OS to accomplish this. The benefits:

- We can have nearly real-time backups of all changes, as they happen
- We get the entire server, not just the data
- We can store more than just the *current* blocks; in fact, we can store changes as far back as we want, meaning we can restore any given file—which consists of multiple disk blocks—to any point in time we want
- We can restore the entire server by simply writing all the latest backed-up blocks to a server—either the original server or a replacement
- In the event of corrupted blocks, we might still be backing those up—but we've also got older, non-corrupt versions of those same blocks, so we can potentially repair files that have become corrupted to their most recent, non-corrupted point in time

This is powerful magic, but it's a reality: Today's market includes solutions that follow this technique. It delivers Backup 2.0: continuous backups *that are designed for restores,* not for archiving.

Where Do You Store Backups?

Do you need offsite storage of your backups? Probably yes. In 1996, Paris-based Credit Lyonnais had a fire in their headquarters. Administrators ran into the burning building to *rescue backup tapes* because nothing was stored off-site. Let me write that again: *Ran into a burning building.* Folks, fire is a constant possibility, as is the possibility of damage from floods (bad plumbing anyone?) and other disasters. If the data is worth backing up, it's worth keeping copies somewhere else. At the very least, have some sort of on-site storage that's disaster-proof—a waterproof fire safe, for example. Does that mean you have to use magnetic tape? No, but you probably will, simply because it's relatively inexpensive, fairly durable, and easy to work with. You'll likely end up using tape in conjunction with something else, in fact, with tapes being the sort of last-resort place to restore from. The point is this: ***Don't assume that a major disaster will not strike you.*** Past performance is no guarantee of future results; just because you've never been hit by a disaster before doesn't mean you won't be hit by one eventually. That's why people buy insurance policies, and backups are basically a form of insurance.

In terms of Backup 2.0, we might combine our block-based backups with some tape-based storage. Our backup server, for example, might periodically dump all its backed-up blocks to a tape array, allowing us to carry a snapshot offsite for archival purposes and to protect against a total disaster to our data center.

Should You Test Backups?

This is a trick question. The answer is, "Of course." As I've already explained, though, few folks actually do. Why? Well, there are really a few reasons, many of which are related to our Backup 1.0 mindset.

First, as I mentioned earlier, is the time commitment. Spending hours doing a test restore isn't in most folks' budgets these days. Of course, a block-based restore can actually be done more quickly: You're streaming the restore from disks over a high-speed network, not reading them ever-so-slowly from a tape drive.

Second, there's the availability of hardware. Now, if you're testing single-item recovery, most backup solutions will allow you to restore files to any location you want, so you just need a small spot on an existing file server. But you should also be testing full-on disaster recovery, where you restore a server that was completely lost (say, to fire) to a different piece of hardware. The problem is that many Backup 1.0-style solutions require you to restore to identical hardware, meaning you have to have a lot of spare servers around. Not gonna happen. A good solution will let you restore to dissimilar hardware, which is actually more practical from a disaster recovery perspective; an ideal solution will let you restore to a *virtualized* server, which is absolutely perfect. So yes: Test your backups. Regularly. Perform both single-item restores and the type of bare-metal restore you'd associate with a total disaster, ideally utilizing modern virtualization technologies to eliminate or reduce the need for extra "test restore" hardware.

Virtualization: More than Just Consolidation

As I'll discuss in later chapters, virtualization is an integral part of the Backup 2.0 mentality. In Backup 1.0, a full-site disaster might mean retreating to a special-purpose, leased offsite recovery facility and starting a lengthy restore process using your most recent off-site backup tapes.

In Backup 2.0, that "facility" might live on the Internet and consist of one or more virtualization hosts, each running a dozen or so virtual servers. You stream your latest backups over the wire to the virtual servers, performing a "bare virtual metal" recovery rather than recovering to actual, physical machines. This approach makes it more practical to recover a set of servers, makes that recovery faster and cheaper (no leased facilities), and makes it more practical to conduct occasional test runs of a complete disaster scenario.

Are You Keeping an Eye on Your Backups?

Do you monitor your backups? Other than just checking an error log, I mean? You should. In fact, checking error logs—aside from being incredibly boring—is the kind of old-school Backup 1.0 mentality I'm trying to change in this book. A modern, Backup 2.0-style solution should alert *you* to problems, and ideally might even integrate with an existing monitoring solution, such as Microsoft's System Center Essentials or System Center Operations Manager.

What might those alerts actually inform you of? Primarily, problems with the backups themselves, such as corruption. Nothing's worse than suddenly finding you *need* your backups and then realizing that they're no good due to corruption; you should be informed of corruption as soon as it occurs. A good Backup 2.0-style solution might inform you via email, might drop something in a Windows event log (which System Center Operations Manager could then pick up and raise to your attention), or perform some similar style of notification.

Figure 1.6 shows how System Center Operations Manager can be used to configure an alert for a given type of event log entry—such as an event log entry created by your backup solution, alerting you to corruption. Note that System Center Essentials works similarly for this type of alert.

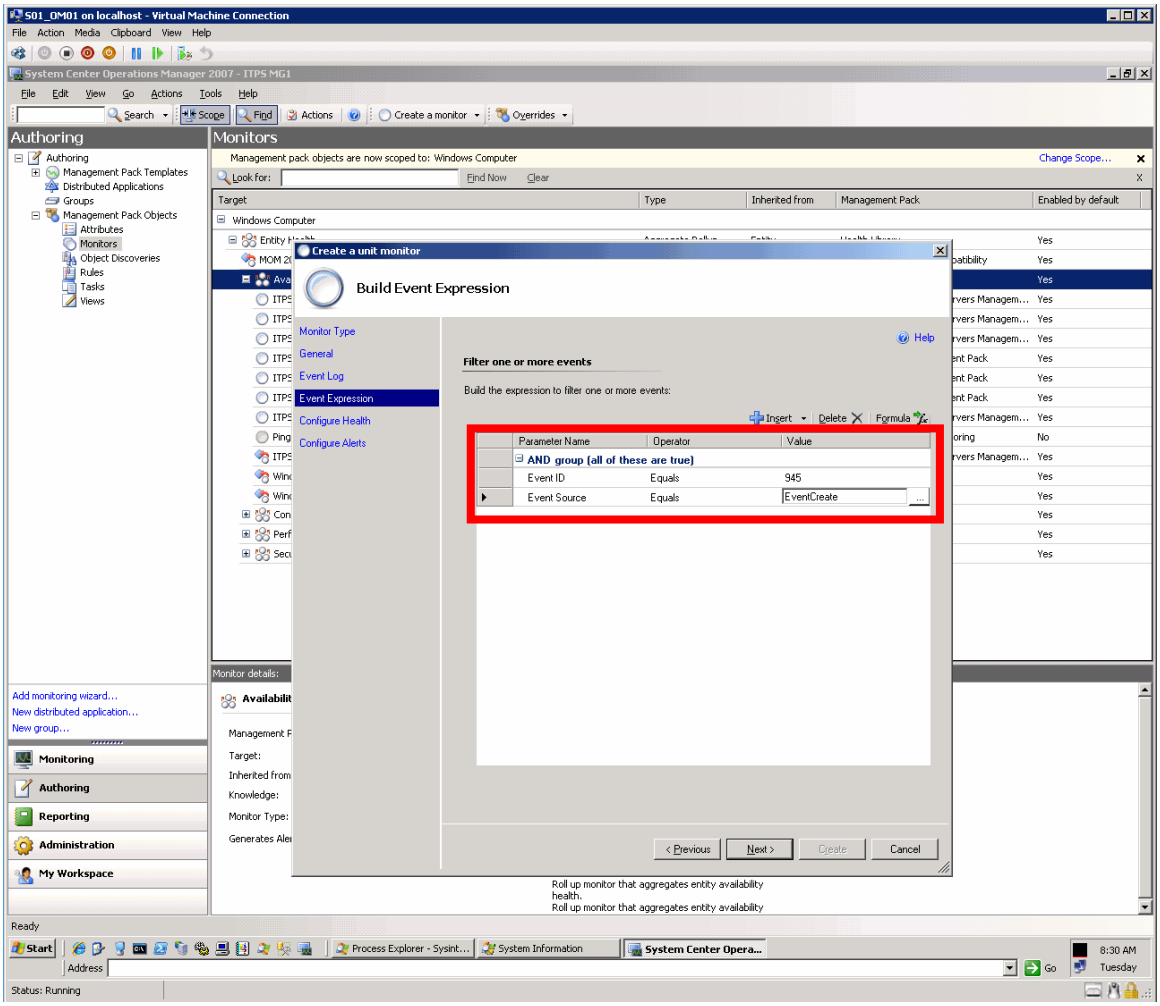


Figure 1.6: Creating an event log entry alert in SCOM.

A really "with it" backup solution might even come complete with a System Center Management Pack. A Management Pack is basically a preconfigured set of rules for monitoring and alerting on a specific application; the pack tells System Center Operations Manager what to look for (such as specific event log IDs) and what to do (such as sending email alerts). But if your backup solution doesn't come with a Management Pack, at least make sure that it has its own built-in features for providing these types of notifications.

Approaches to Backing Up

Having advocated for block-level backups, I want to take a step back and briefly review the entire gamut of backup possibilities. Although some of these don't meet my requirements for a good backup *and recovery* system, they nonetheless offer some business value that you should be aware of.

File-Based Backups

File-based backups are the oldest type of backup, and probably still the most common. It involves simply taking a snapshot—a point-in-time copy—of one or more files. These types of backups may have difficulty working with open files, and they don't capture *every* change made to a file—they just grab a copy of it at a specific time.

But there's value here. For example, Windows' built-in Volume Shadow Copy feature is essentially an on-server file-based backup, grabbing copies of files as users change them and storing them in a local cache on the server. Users can access these cached fie versions on their own, using Windows Explorer's Previous Versions feature, as Figure 1.7 shows.

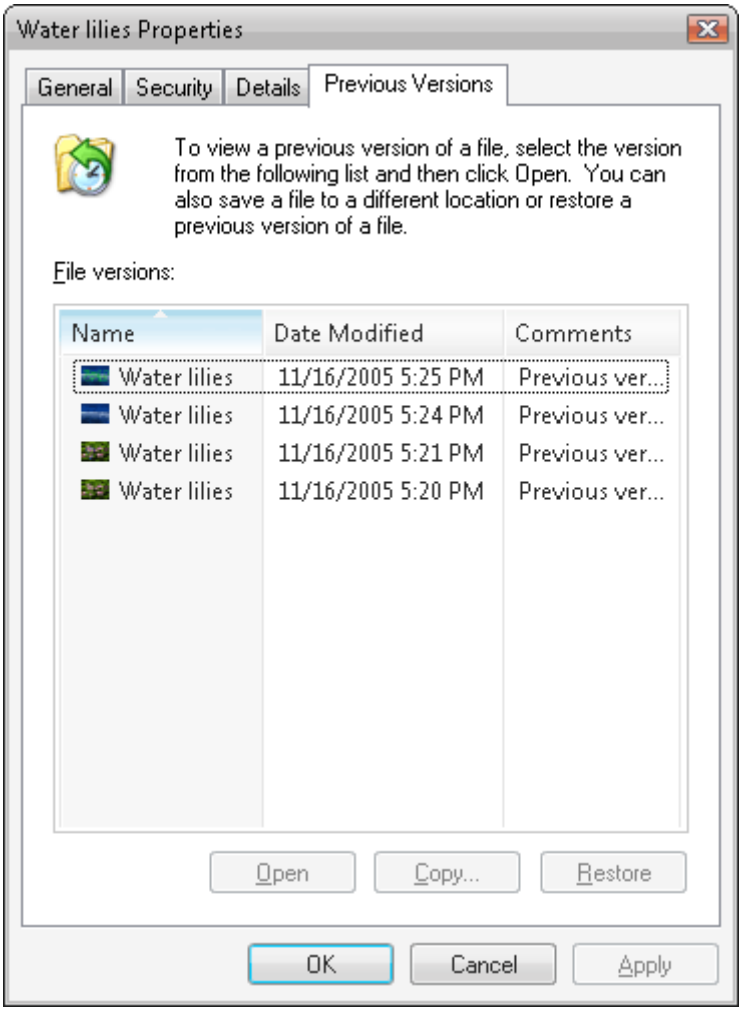


Figure 1.7: Accessing previous versions.

The key phrase here is *on their own.* Unlike many data center-class backup solutions, Volume Shadow Copy / Previous Versions is designed for user self-service. Properly used (meaning your users will need a bit of training), this feature can help prevent calls to the Help desk—and overhead for you—when users need to roll back a file to a somewhat older version. In fact, I've seen this feature—again, with a bit of end-user training—reduce single-file recovery Help desk calls by 90% in several of my clients. Those organizations tell me that the average cost for completing a file recovery request is about $75, and the ones that keep really good records average about four calls a week. That's a savings of more than $15,000—all for free, since the feature is built-in to Windows (well, you do have to spend a bit extra for the disk space needed to store the cache).

In a Backup 2.0 world, there should be room for complementary solutions. Previous Versions meets a specific need: user self-service for individual file rollback. Whatever data center backup solution you select shouldn't interfere with complementary solutions, and in fact, should embrace them. Where file-based backups tend to fall short—as I've discussed—is in the whole-server backup scenario, which you'd be performing in the data center.

Image Backups

Image backup is another term for what I've been calling block-based backups. There are really two ways to achieve an image-style backup: by using solutions such as the tried-and-true image snapshot software and via what I'll call "streaming" images.

Traditional imaging software, often used for deploying operating system images to new computers, commonly makes point-in-time snapshots of a disk, typically compressing the disk blocks so that the resulting image file is much smaller. This software isn't usually positioned as a backup and recovery solution but rather as a deployment solution: You make a "template" computer the manual way, image it, and then deploy that image rather than installing other computers manually. It's most commonly used for desktop deployment, and it can serve as a kind of last-ditch recovery tool for desktops, essentially re-deploying the image to get a machine back to "square one." But as a point-in-time snapshot, it's not useful for capturing data, changes to applications, and so forth.

Another weak spot is that snapshot imaging software usually requires the source computer to be unavailable. Typically, the image is captured using a pre-boot environment (like the one shown in Figure 1.8)—a sort of stripped-down OS that runs in lieu of Windows. This ensures that none of the files on disk are open and changing so that a single, consistent snapshot can be gained. You can see where this would be a bit burdensome as a continual backup technique.

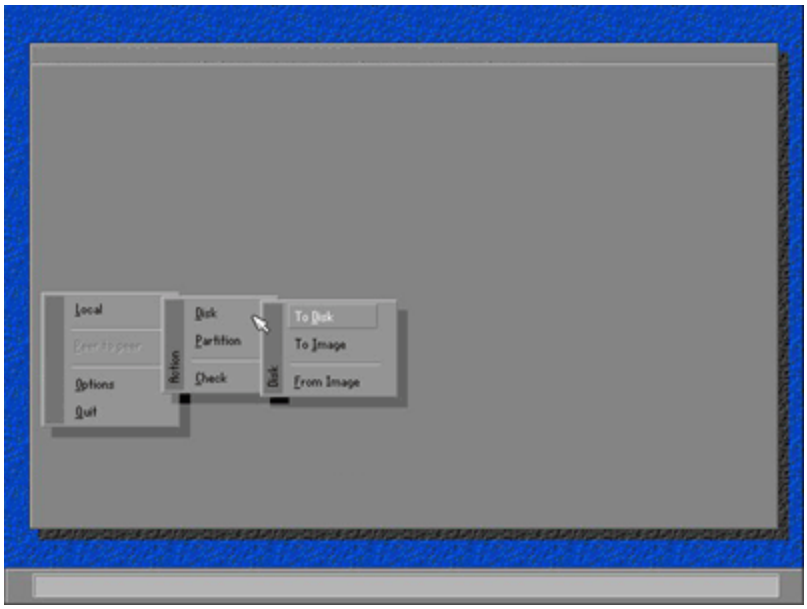


Figure 1.8: Imaging software's pre-boot environment.

"Streaming" images are the kind of block-based backup I illustrated in Figure 1.5, earlier in this chapter. This technique is the basis for almost real-time, continuous data protection of multiple servers. You could use the technique with desktop computers, too, although I suspect doing so wouldn't be practical—it would involve a *lot* of backup data flying around your network, not to mention a lot of storage. No, I think this technique is really geared best for the data center, where you're backing up servers and where you can easily set up high-speed or even dedicated network connections between those servers.

Application-Specific Ideas

Some applications—primarily mission-critical, always-on applications such as Microsoft SQL Server and Exchange Server—present their own challenges for backup and recovery. These applications' executables are always running, and they always have several data files open, making it difficult for file-level backup software to get a consistent snapshot.

To help address this, the applications' designers take varying approaches. SQL Server, for example, has its own internal backup and recovery capability, which is tied to the product's own unique architecture. Traditionally, the best way to get a SQL Server backup is to ask SQL Server to do it. You might, for example, use SQL Server's own tools to produce a backup file, then grab that file with a traditional file-based backup solution. Or, you might create an agent that taps into SQL Server and gets the data that way—the approach used by most enterprise-level, Backup 1.0-style backup solutions. Figure 1.9 shows a common dialog box for a backup solution's configuration, showing that a SQL Server-specific agent is loaded and able to stream data from SQL Server to the backup software. Exchange Server functionality might work similarly—in fact, you can see that tab in the figure as well.

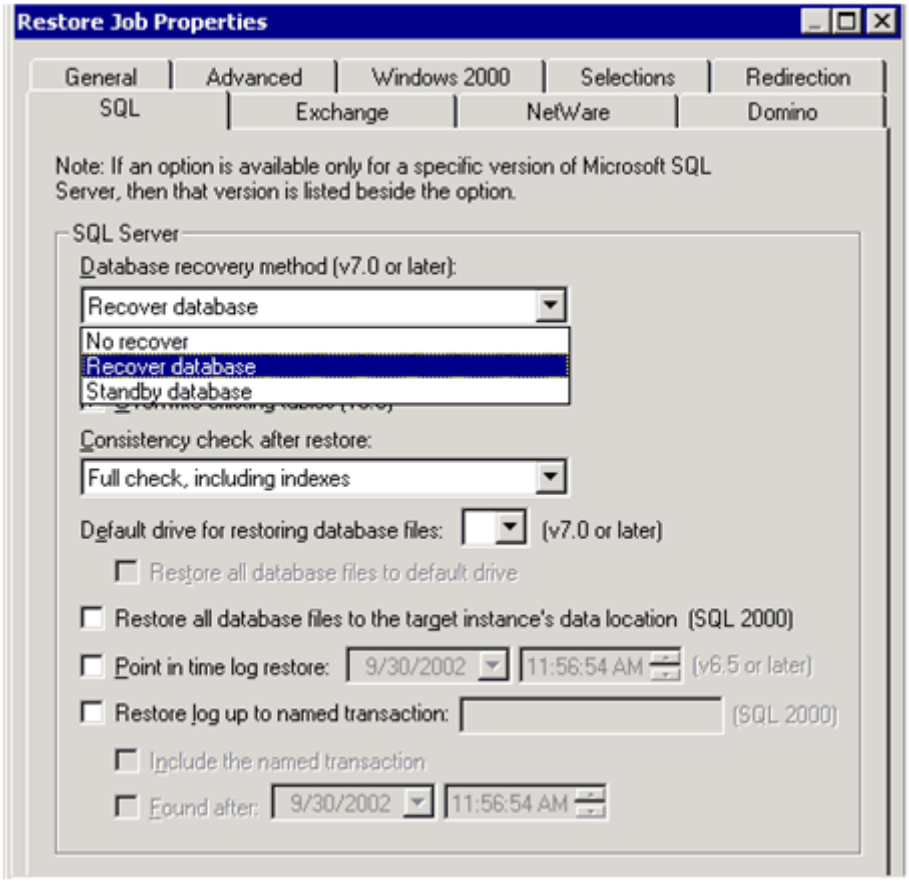


Figure 1.9: Application-specific backup agents.

Exchange Server's developers took a slightly different approach, choosing to integrate with Windows' Volume Shadow Copy service. Essentially, they provide a copy of the Exchange data files through Volume Shadow Copy; a backup solution simply needs to access the Volume Shadow Copy Application Programming Interfaces (APIs) and request the "latest copy" of the database. Again, it's Exchange Server that's doing most of the work, but a dedicated agent of some kind is usually needed to get to the right APIs. As Figure 1.10 shows, even Windows Server 2008's built-in backup can be extended to "see" the Exchange Server databases.

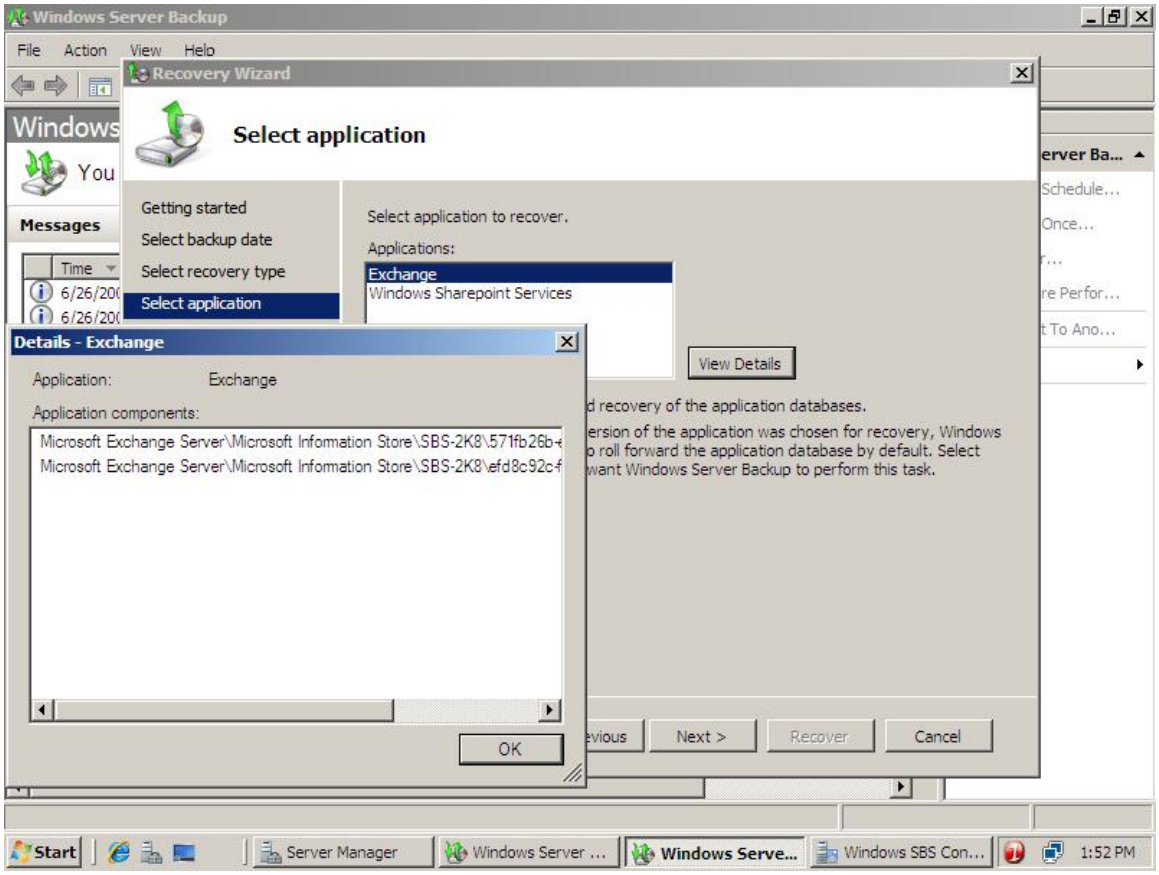


Figure 1.10: Backing up Exchange Server in Windows Server Backup.

The downside is that these application-specific approaches are still Backup 1.0 in nature, meaning they're grabbing a snapshot. You're still at risk for losing data and work that occurs between snapshots; particularly with these mission-critical applications, I think that's just unacceptable.

Block-level backups can certainly solve the problem because they're grabbing changes at the disk level and don't particularly need to "understand" what those disk blocks are for. A disk block that's part of a file looks the same as one that's part of a SQL Server database, so the backup solution just grabs 'em all. But from a *recovery* viewpoint, your backup solution does need some additional smarts. Here's why: A simple file—say, a Word document—consists of several blocks of disk space. It's easy to keep track of which blocks make up any given file, and no disk block will ever share data from two files. If you need to restore Salaries.xls, you figure out which blocks that file lives on, and restore just those. Easy.

With complex data stores—such as SQL Server and Exchange Server—things aren't so easy. A single mail message might occupy multiple blocks of disk space, but those same blocks might also contain data related to other messages. The database also has internal pointers and indexes that need to be restored in order for a given message to be accessible. So a block-based backup doesn't need much in the way of extra smarts to *make* a backup, but it will need some cleverness in order to *restore* single items from that backup. Solution vendors tend to approach this by using plug-ins: It's easy to think of these as being similar to the Backup 1.0-style agents, but they're not. These plugins don't necessarily assist with the backup process (although they may record special information to assist with recoveries), but they do contain the smarts necessary to peer "inside" complex data stores to recover single items.

Our Backup 2.0 Wish List

The following list highlights desirable capabilities in a Backup 2.0-style backup solution:

- Continuous data protection that's always on, always working, and as close to real-time as is practical
- The ability to move snapshots of my backups to tape (or other portable media) for off-site storage
- The ability to restore anything from a single file to an entire server—quickly, and to different hardware (or virtual hardware) if needed
- Block-level imaging that provides the ability to roll back to any point in time and keeps backups physically separate from the source
- Automatic notifications of problems—such as corruption—with the backups
- Doesn't interfere with complementary solutions such as Windows' own Volume Shadow Copy feature
- The ability to easily test restores, from a single file to a complete disaster, ideally using dissimilar hardware or virtualization
- The ability to restore single items from complex stores like SQL Server or Exchange Server

I'll add a few things to this list as we progress through the upcoming chapter, but this is a good start. It represents everything that the Backup 2.0 philosophy is all about, and it meets all the implied requirements in our business-level statement:

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

What's Ahead

I've got a full plate coming up for you, starting with the next chapter in this book, where I get to share some of the horror stories I've run across with my consulting clients, in the news, and so forth. It's kind of interesting to see the problems others have had, but it can be instructional, too: I'll examine each case and draw conclusions about what went wrong and what you would need to do to avoid that situation.

Chapter 3 is where I'll dive into the actual technology of whole-server backups. This is an area where I think "Backup 2.0" really has some immediate benefit, but I'll start by examining more traditional whole-server backup techniques and identifying things that don't always work so well. If you're responsible for backing up domain controllers, infrastructure servers, Web servers, a public key infrastructure, or similar servers, then this is the chapter for you.

In Chapter 4, I'll tackle the tough topic of Exchange Server backups—tough enough that the backup software that shipped with Windows Server 2008 couldn't even do it. Again, I'll lay out some of the more traditional ways that Exchange backups have been made, and then rethink the process and come up with a wish list of backup capabilities that include several Exchange-specific concerns. Chapter 5 will follow the same approach for SQL Server, and Chapter 6 will examine SharePoint in the same way.

Chapter 7 will be a bit of a departure, as I'll focus on virtualization server backups. This is still a relatively new field, and I'll look at ways in which traditional backup techniques are being used and examine how well they're actually getting the job done. I'll cover some of the unique aspects of virtualization backups and examine innovative techniques that Backup 2.0 can bring to the table.

In Chapter 8, I'll pull back a bit for a broad look at other backup concerns and capabilities: Bare-metal recovery, data retention concerns, compliance and security issues, mobile infrastructure problems, and so on. I'll also look at instances where old-school backups might still provide some value, and I'll offer advice for integrating Backup 2.0 with more traditional techniques.

All of the backups we're making are going to require some serious storage, so I'll use Chapter 9 to focus on storage architecture. I'll look at how backup data is structured, and compare the advantages and disadvantages of things like storage area networks (SANs), tape drives, local storage, and so forth, and examine pressing issues of storage: compression, encryption, security, de-duplication, and so on. I'll also look at unique ways that Backup 2.0 allows you to interact with your backed-up data more easily and efficiently.

Chapter 10 will focus on disaster recovery—and I mean *real* disasters. I'll look at things like bare-metal recovery, and I'll cover some of the more interesting capabilities that today's technologies offer, such as using virtualization—rather than dedicated off-site facilities—as part of a disaster recovery plan.

Chapter 11 is for all the business-minded readers out there; this chapter is where I'll discuss the costs involved in re-architecting backup and recovery to use Backup 2.0 techniques. Naturally, I'll also help you determine whether doing so is actually worth it to your organization, and even tackle some of the non-technical—what I like to call "political"—issues that you may have to solve in order to make your backup situation more modern and efficient.

Finally, Chapter 12 will be the chance for me to share stories from my own experiences with Backup 2.0—a sort of "tales from the trenches" chapter, with case studies that describe successes (and challenges) I've seen with Backup 2.0.

We have a long journey ahead of us, but you've made a good start. I look forward to seeing you again in the next chapter.