Realtime publishers

Solutions for Automating IT Job Scheduling

Greg Shields

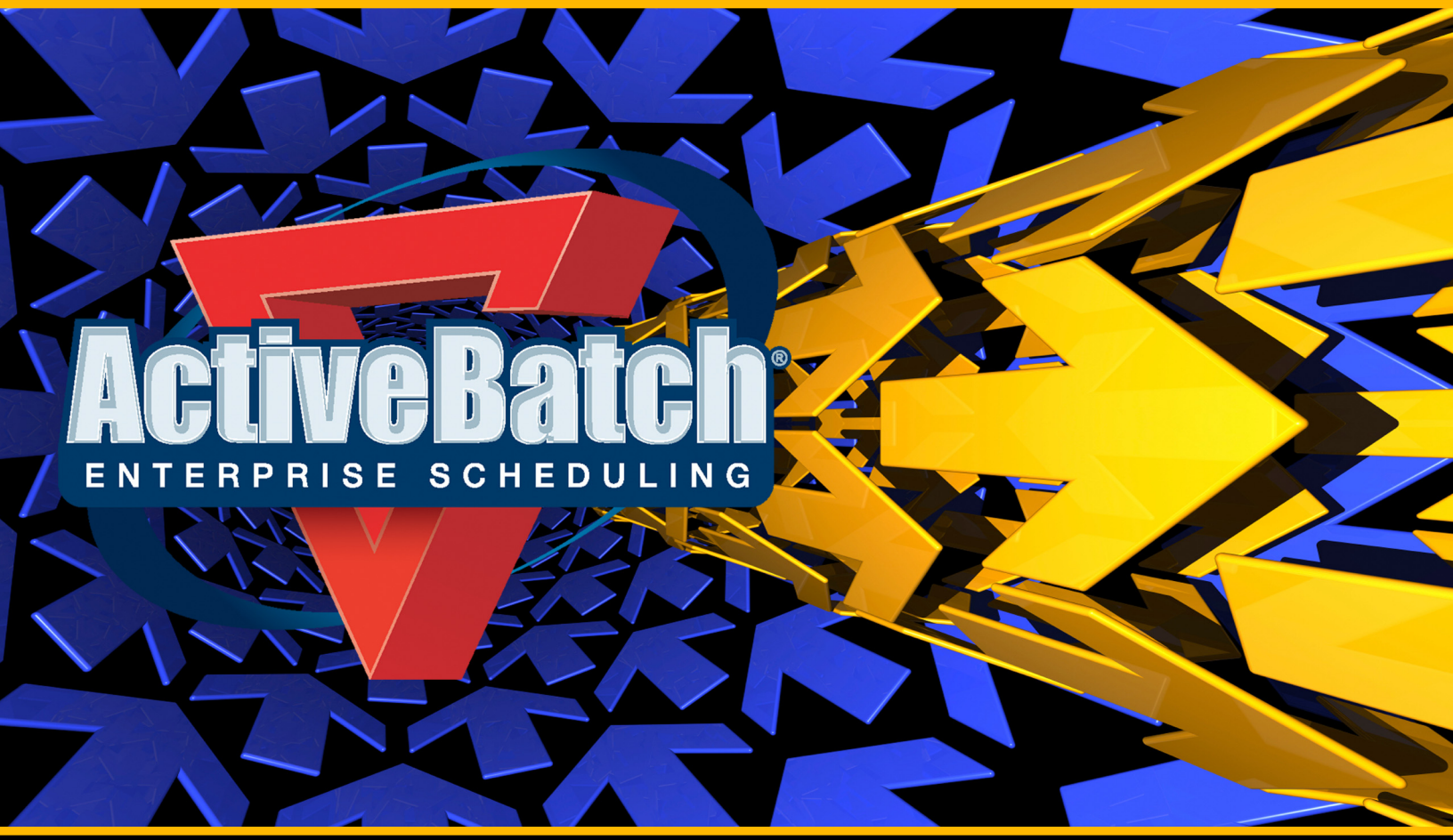
sponsored by

Advanced Systems Concepts, Inc.

Simplifying integration

Removing complexity

Shortening time to implementation



Integrating Your Business Applications Across Distributed Computing Environments

ActiveBatch® Enterprise Job Scheduling and Workload Automation, from Advanced Systems Concepts, removes the complexity from integrating business applications, databases, and mainframes. Focused on shortening the time to implementation and improving service levels by eliminating custom scripting via its Integrated Jobs Library, ActiveBatch automates job scheduling across diverse distributed computing environments, creating a centralized view of operations at the project, organizational or enterprise level.

Learn More at www.ActiveBatch.com

Advanced Systems Concepts, Inc.

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: Do I Need Job Scheduling? Ten Questions to Ask Yourself

I still remember that day the boss walked into my office. And I still remember her fateful yet enticing words that talked me into tackling the project, *The Project That Would Change Everything.* "You're the Scripting Guy. You'll make it work," she said. I agreed, and the rest is history.

You're probably familiar with this story: Big projects in IT start that way. Big projects that may seem great on paper but then get very, very complex as they're developed. Big projects whose multiple applications across multiple platforms require substantial integration effort.

But this story isn't only about that one big project. It's also about all the other little automations—scripts for Windows and Oracle and Active Directory (AD), SQL packages, Linux cron jobs, and so on—that creep into every IT environment over time. Those automations absolutely solve a business' immediate needs. But without central management, they also come with a cost. That cost arises as scripts age, technology changes, and the script owners relocate or leave the company.

Back then, I was known as The Scripting Guy. If you needed a quick data transformation, or a scheduled movement of files from one system to another, I was your go-to person. I had developed a command of the major scripting languages, along with all the other necessary add-ons one needed to be The Guy. Over the course of several years, my integration prowess had grown to include platform- and application-specific technologies such as WMI, ADSI, SQL, some Oracle, and even a bit of Linux and IBM AIX.

I put that knowledge to what I thought was good use. Over the years, I had gotten to the point where much of my daily responsibility was automated…sort of. Sometimes my little automations broke. Sometimes they were accidentally deleted or otherwise wiped out through the regular change that happens in any data center. Sometimes their need went away, or a server's configuration was updated, and as a result, I had to go find them once again and remember what they were intended to do.

The result was a not-very-well-oiled machine; one that created even bigger problems the day I moved on to a new employer. You see, I had little automations scattered around the company servers with my name on them. Last I heard, they're still finding them years later, usually after some process breaks that nobody ever knew existed.

You can probably guess what we were missing. *We needed enterprise IT job scheduling.* That's why I'm writing this book, to explain what this approach is and help you realize you could probably use it as well. Throughout this book, I intend to return back to that big project's story along with a few of the other little ones to show you why.

The Pain of the Non-Homogeneous IT Environment

This book wouldn't exist if every IT technology seamlessly talked with each other, transferring data, events, and instructions across platforms and applications with ease. If every technology could perfectly schedule its activities with itself and others, you wouldn't be reading these pages.

But you are, and consequently this guide indeed exists.

It exists because IT job scheduling is a task that every enterprise needs, as do many small and midsize organizations as well. "Jobs" in this sense represent those little packages of automation I discussed earlier. Some work with a single system. Others integrate the services of multiple systems to create a kind of "mixed workload" that produces a result your business needs.

Consider a few of those jobs you're probably already creating today:

- Reports about activities on an email system need to be collected and distributed to security groups
- Customer files transferred inbound to a Linux FTP server need to be ingested upon receipt into a SQL database
- Provisioning AD users requires several additional steps across systems such as Linux, email applications, and Oracle databases
- New records in a Microsoft SQL or Oracle database trigger actions to occur in one or more middleware system

Your jobs might be less complex, working with only a single system or application. Or they might be exceedingly so, requiring the participation of multiple applications across different platforms in different parts of the world. At issue here isn't necessarily how complex your jobs are. The "simple" ones have many of the same requirements and necessitate as much due diligence as the "complex" ones. Rather, the issue has more to do with the workflow that surrounds those jobs, and the solutions you implement to manage, monitor, audit, and otherwise keep tabs on every activity at once.

It also has to do with the very different languages and techniques that each IT technology uses and requires. Those differences represent a big headache inside today's heterogeneous data centers. Your IT operating environment surely has Windows systems. But it probably also has Linux, Oracle, HP-UX, Solaris, and others. You probably need to transfer XML documents, DOCX files, and XLSX spreadsheets over multiple protocols like SMB, FTP, and SSH. Even your monitoring comes in many flavors: SNMP for switches and routers, WMI for Windows systems, and all the various UNIX and Linux widgets for keeping tabs on their activities.

You can imagine just a few of the headaches these radical differences in applications and platforms create:

- Every operating system (OS) speaks its own language
- Every application speaks its own language
- Every application and OS uses its own security model that doesn't necessarily integrate well with each other
- Transferring data or scheduling actions within and between each is difficult and sometimes impossible with native tools alone
- Most importantly, centrally managing every job everywhere just isn't possible

Solving these five problems is the primary mission of an IT job scheduling solution. From a central location (see Figure 1.1), an IT job scheduling solution creates a platform on which to run all your little "automation packages" that might otherwise be spread across technologies. Using a centralized solution, a database job, a UNIX/Linux job, and an FTP job are all parts of the same management framework. All begin their execution from the same place, and all are managed and monitored from that single location.

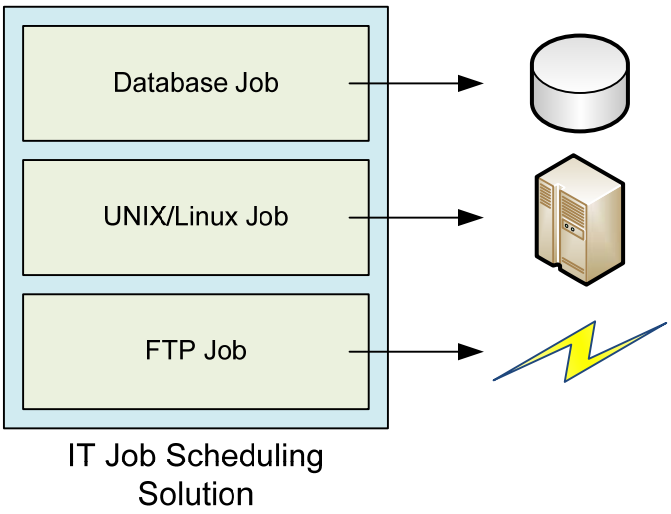


Figure 1.1: An IT job scheduling solution centralizes jobs of all types, across all platforms and applications.

As you can probably guess, such a solution has to be exceptionally comprehensive. A solution that works for your company not only needs to support your management, monitoring, and workflow needs. It must be more than just a more-powerful version of the Windows Task Manager. It also needs to support the integrations into every OS, platform, and technology that your business processes incorporate. That's why finding a solution for automating IT job scheduling can be such a challenging activity.

To help you out, you can consider this guide to be a kind of automation "idea factory." Its four chapters will present you with questions to ask yourself, helping you frame your need for a job scheduling solution. It delivers a set of real-world use cases for seeing scheduling in action. It deconstructs an IT job so that you can peer inside its internal machinery and understand the power of a centralized solution. And it will conclude with a checklist of requirements you should consider when seeking the software that creates your solution. I'll be your guide, and throughout this process I'll share a few of my own stories to bring some real-world experience into this complex topic.

> **Note**
> In this book, you'll hear me use the term *job scheduling*. Another commonly-used term for job scheduling is *workload automation*. For the purposes of this book, you can assume that the two are interchangeable.

Constructing an Unfriendly Application

Now, back to my story from long ago. Every OS and application comes equipped with multiple ways to perform its core functions. You already know this. An OS includes one or more scripting languages to enact change and read data. Every modern database has its own scheduling and automation functions, enabling the creation of packages for inserting and selecting data. Even middleware technologies and applications have their own APIs, which can be interfaced either inside or outside the application.

But the internal languages and automations that come with a product are rarely equipped to handle actions outside that product. Ever try to use an XML document to instruct a SQL Server to update an Oracle database row so that an SAP application can provision a process to an AIX mainframe? Whew! That's pretty close to the situation I experienced as I started on *The Project That Would Change Everything*.

Let's start with a little background. Why that project was needed is really unimportant, as is what we were doing with its data. What is important, however, are the interconnections between each of its disparate elements. Multiple applications running atop more than one OS, integrating with different databases, and requiring data from both inside and outside the organization was just the start.

To get going, I attempted to diagram its components, creating something close to what you see in Figure 1.2. At a high level, this system was constructed to aggregate a set of data from outside our organization with another set on the inside. Our problem was the many different locations where that data needed to go.

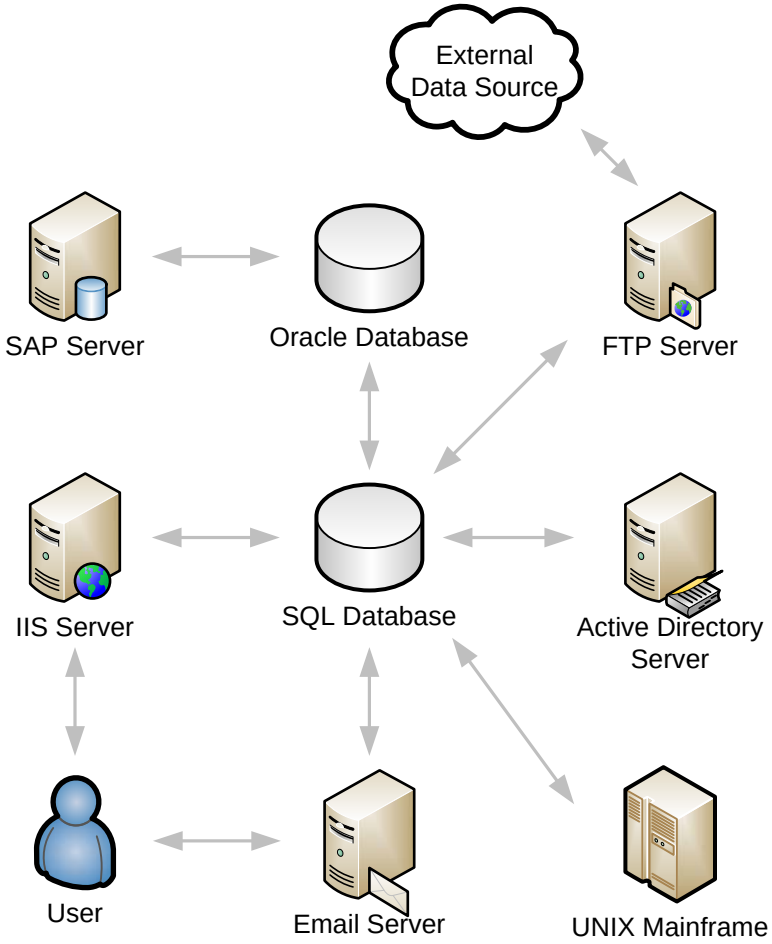


Figure 1.2: My unfriendly application.

Let me break down the mass of arrows you see in Figure 1.2. The flow of data in this system started via an FTP from an external data source. That data, along with all its metadata, needed to be stored in a single, centralized SQL database. There, permissions from Windows AD would be applied to various parts of the data set. Some data was appropriate for certain users, with other data restricted to only a certain few. Information inside the FTP data stream would identify who should have access to what.

Users could interact with that data through a Microsoft IIS server running a homegrown Web application. That Web application used XML to transfer data to and from the SQL database. Certain types of data also needed to be added to our company SAP system running atop Oracle, requiring data transformations and delivery between those two systems.

Occasionally, portions of that data would need to be ingested into a UNIX mainframe for further processing. There, it would be consolidated with data from other locations for greater use elsewhere in the company. An email server would ensure users were notified about updates, new data sets, and other system-wide notifications.

That's a lot of arrows, and each of those arrows represent an integration that needs to be laid into place in order for the entire system to function. Each arrow also represents an activity that needs to happen at a particular moment in time. Data heading towards the Oracle database obviously couldn't be scheduled to go there until it was actually received at the SQL Server system. Users shouldn't be notified unless something important to them was actually processed. Just the scheduling surrounding each arrow's integration was a complex task unto itself.

Does this look like one or more of the systems that are currently in your data center? If you're doing much with data transformation and movement, you might have the same scheduling headaches yourself. That's why there are four critical points that are important to recognize:

- *You can find interconnected systems like this everywhere*. In companies across the globe, IT systems are constructed in ways just like this, using technologies just like these. So although this project would indeed change everything for my company, others elsewhere are already dealing with slightly-different-but-mostly-the-same integrations for other reasons.
- *Constructing and managing this system is not necessarily an activity limited to just developers*. I'm not a developer. I'm a systems administrator who just happened to be considered The Scripting Guy. Thus, in the minds of my superiors, I made the perfect candidate for leading this project to completion. The same holds true with the world's other systems as well. IT administrators are tasked with constructing "systems" that are made up of many moving parts. Getting them all to talk with each other and schedule their activities is only the first step in that battle.
- *It is entirely possible to build this using scripting and application-specific automations, but doing so is a really bad idea*. My strewn-about SQL SSIS packages and VBScripts and cron jobs never prepared me for the administrative overhead of a real "system" rather than some jumble of unconnected actions. Successfully scheduling the activities across every piece of the infrastructure isn't a task that works well unless it's centralized. Even worse, managing and maintaining those automations becomes a risk to the system's operation as they grow in number and get more complex.
- *The scheduling of activities in a system like this requires more than just the clock on the wall*. Think about the data flow between these individual components, and the events that need to trigger other events. Simple actions inside simple systems might work just fine with a time and date-based schedule. Is it 3:00AM? Start the backups! But systems with greater complexity and more dependencies require more powerful means of determining when to do something. Those decisions can be made based on receiving a piece of data, seeing a change, reading a log file, or any of the myriad other state changes that occur in any IT system. A good IT job scheduling system will give you plenty of conditions that you can customize for identifying when an action should occur.

It is the combination of these four realizations that helped me understand that I needed to step outside my application-specific mindset. It helped me realize I needed to look to solutions that schedule activities across every platform and every application. That's when I started looking into enterprise IT job scheduling solutions.

Defining IT Job Scheduling

Let's now take a step back from the storyline and think for a minute about what IT job scheduling should be. I've already suggested that a "job" represents some sort of automation that occurs within an IT system. But let's get technical with that definition. I submit that an IT job represents *an action to be executed*. An IT job might be running a batch file or script file. It might be running a shell command. It could also be the execution of a database job or transformation. Essentially, anything that enacts a change on a system is wrapped into this object we'll call a job.

Using an object-oriented approach, it makes sense to consolidate individual actions into separate jobs. This single-action-per-job approach ensures that jobs are re-usable elsewhere and for other purposes. It means that I can create a job called "Connect to Oracle Database" and use that job any time I need to make an Oracle database connection anywhere.

Now if each job accomplishes one thing, this means that I can string together multiple jobs to fully complete some kind of action. I'll call that string an IT plan. A plan represents *a series of related jobs that can be executed* with the intended goal of carrying out some change. Figure 1.3 shows a graphical representation of how this might work.

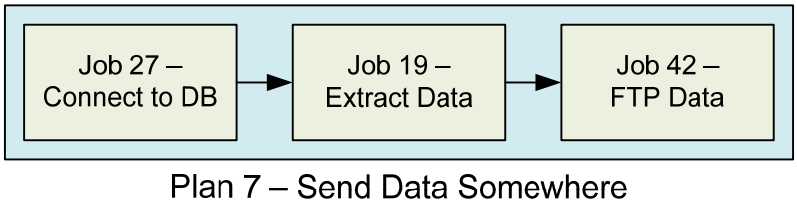


Figure 1.3: Multiple jobs are connected to create a plan.

In Figure 1.3, you can see how three different jobs are connected to create the plan. Job 27 connects to an Oracle database. It passes its result to Job 19, which then extracts a set of data from that database. Once extracted, the data needs to be sent somewhere. Job 42 completes that task, as it FTPs the data to a location somewhere.

There's obviously an art to creating good jobs. That's a topic that I'll discuss in greater detail in Chapter 3, but I need to introduce some of the basics here. A good job, for example, might not necessarily have any specific data or hard information that's stored inside the job. Rather than a connection string to a specific server for Job 27, a much better approach would be to use some kind of variable instead.

Developers use the techie term *parameterization* to represent this generalizing of job objects and the subsequent application of variables at their execution. Figure 1.4 shows how a parameterized plan can link three generic jobs. At the point this plan is run, those jobs are fed the variable information they need to connect to the right database, extract the right data, and eventually pass it on to the correct FTP site.

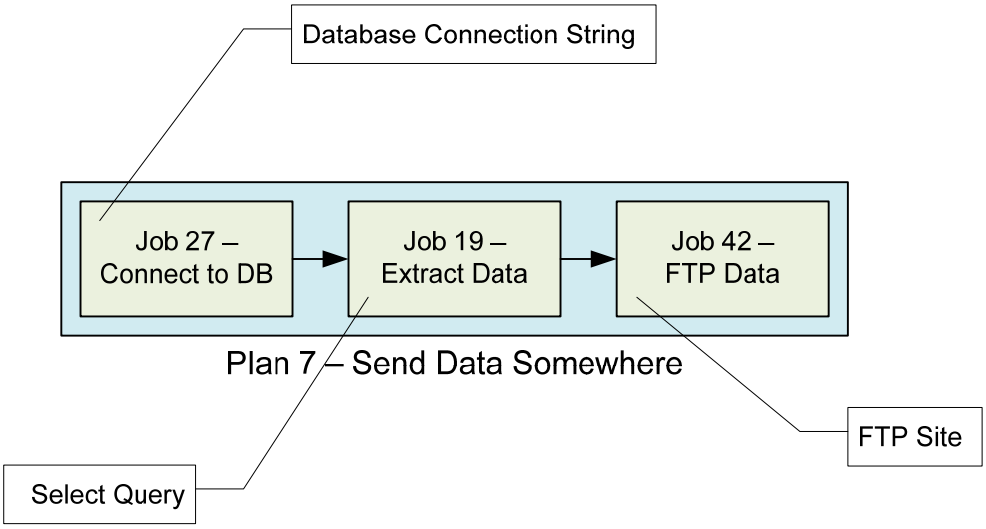


Figure 1.4: Feeding parameters to jobs in a plan.

By parameterizing the plan in this way, I now get reusability of the plan in addition to all the individual jobs that make up that plan. Should I down the road need to attach to a different database somewhere, pull off a different set of data, and send it to some other FTP site, I can accomplish this by reusing the plan and modifying its variable information. That's reusability on top of reusability!

There's obviously quite a bit more to this whole concept of working with jobs and plans. I'll spend more time in Chapter 3 helping you understand the various characteristics that can be assigned to a job and a plan as well as other objects a typical IT job scheduling solution will use.

But there is one characteristic that merits attention before moving on. That characteristic is the schedule itself, which needs to be applied to the object to tell it when to run. I mentioned earlier that scheduling for large systems like The Project requires a kind of flexibility you just can't get by looking at the clock on the wall. Rather, the kinds of jobs that project needs tend to be more related to actions or state changes that occur within the system.

Let's assume that Figure 1.4's "Plan 7" relates to some data transfer that needs to happen inside The Project. In this case, let's assume that the data transfer occurs between its SQL Server and UNIX mainframe. Figure 1.5 shows a graphical representation of how this might be applied. There, you can see how three different schedules could potentially be attached to the newly-created plan:

- Invoke the plan at 3:00PM
- Invoke the plan when a set quantity of new data arrives
- Invoke the plan when processor utilization is less than 30%

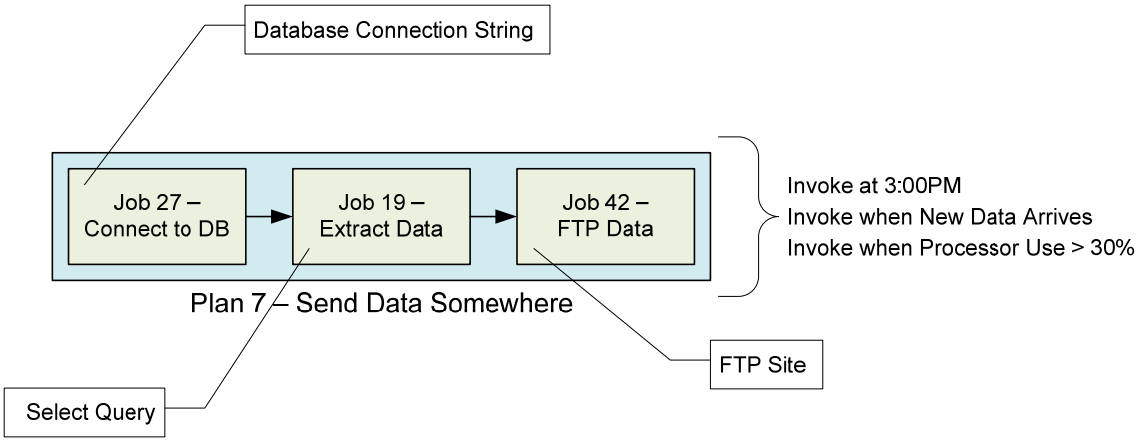


Figure 1.5: Applying a schedule to a plan.

Any of these three schedules can be appropriate, depending on the needs of the system and its components. For example, the first schedule might be appropriate if a daily data dump is all that's necessary. In that case, a date/time-centric schedule might be all that's necessary to complete the action. Very simple.

The second and third tasks highlight some of the more powerful scheduling options that could also drive the invocation of the plan. In the first, the plan is executed not based on any time of day. Rather, it executes when a set quantity of new data has been added to the database. This could be a smart solution if you want these two databases to stay roughly in sync with each other. It is really powerful when you consider how difficult that kind of scheduling would be to create if you were using just the native SQL or UNIX tools alone.

That third schedule is particularly interesting, because it could be used alone or in combination with the second. That third schedule instructs the plan to run only if the server isn't terribly busy. Using it in combination with the second allows you to maintain a level of synchronization while still throttling the use of the server. A good job scheduling solution will include a wide range of conditions that you can apply to plans to direct when they should kick off.

Ten Questions to Ask Yourself

Again, I'll dive deeper into this deconstruction of an IT workflow's components in Chapter 3. But before you can truly appreciate the power of this modular approach, there are likely a few questions that you're probably asking yourself. If you're not, let me help you out with a list of ten good questions about your own environment that you should probably ask yourself. Your answers to these ten questions will determine whether you'll want to turn the next chapters. If you're experiencing zero headaches with the tools you have today for scheduling your IT activities, you won't need the rest of this book.

Everyone else will.

1. How Much Time Have You Wasted in Completing Tasks Manually?

It wasn't many years ago that one of my jobs was in keeping a set of servers updated. Monthly updates were de rigueur with some on even shorter schedules. Each came with a very short time window when they could and should be applied. The big problem resulted from the fact that these updates typically required a server reboot to get them applied.

At that time, our reboot window was in the wee hours of the morning, many hours past the usual 8-to-5 workday. For me, sticking around once a month to complete these updates represented a hardship on self and family. That's why I created my own automation that wrapped around these updates' installation. For my solution, when updates were dropped into a particular location, they were applied at the next window. My mobile device notified me should any problems occur.

From that part on, adding updates to servers meant simply adding them to the right location and making sure my mobile device was near the bedside. Yes, sometimes they'd experience a problem, but those could be fixed through a remote control session. Successful months could go by without loss of sleep or important family time.

Although your IT job scheduling needs might not necessarily go down the path of system update installation, this time (and money) savings becomes an important parable. Computers are designed by nature to be automation machines. Thus, it stands to reason that any manual activity should have an automation-friendly adjunct. It is that adjunct that can be a part of your greater scheduling solution.

Can you afford to pay the risk of inappropriate execution, forgetfulness, or user error in your critical activities? If not, creating flexible and reusable workflows via an IT job scheduling solution should pay for itself in a very short period of time.

2: What Is the Cost of an Error Incurred During a Manual Task?

That first question introduces the possibility of three kinds of risk in any manual system. First is the risk of inappropriate execution. Any task that requires manual intervention also introduces the notion that it could be executed at an inappropriate time. Or, more dangerously, such a task could be re-parameterized to send data to the wrong location or execute it in an inappropriate way. There is a recognizable cost associated with this risk.

I remember a situation where a script was created that would apply a set of data to a specific server upon execution. That script took as parameters a list of servers to send the data. One day, a junior administrator accidentally invoked the script with the "*" wildcard in place of a list of servers. As a result, data was distributed to every server all across the company. That single invocation cost the company significantly to clean up the mess.

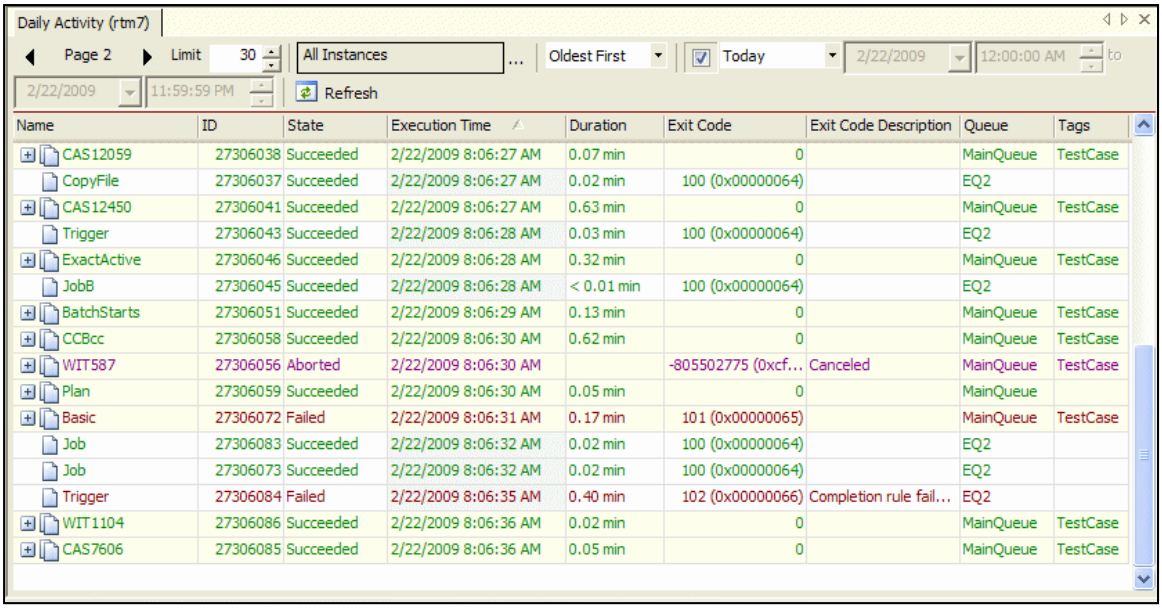
Forgetfulness and user error are both additional risks that can be addressed through a job scheduling solution. In such a solution, jobs and plans are run within the confines of the system and its security model. Dangerous jobs can be specifically restricted against certain individuals or execution models. Centralizing your job execution security under a single model protects the environment against all three of these costly manual errors.

3: Where Can You Go for a Heads-Up Display of IT Activities?

You probably have monitoring in place to watch servers. You've probably got similar monitoring for network components, perhaps even as part of the same product. But does your data center also leverage a unified heads-up display for monitoring jobs along with their execution success? A failure in a job can cause the same kinds of outages and service losses as a failure in the network or its servers.

If your business systems interconnect through multiple scheduling utilities across multiple products and platforms, there usually isn't a way to centralize all those activities under one pane of glass. What you need is the same kind of monitoring for IT jobs that you've already got in place for your other components.

As you can see in Figure 1.6's mock-up, you can get that by using a centralized approach. There, every action across every system and application is centralized into a single screen. Determining which jobs ran successfully is accomplished by looking in one place.



Name	ID	State	Execution Time	Duration	Exit Code	Exit Code Description	Queue	Tags
CAS12059	27306038	Succeeded	2/22/2009 8:06:27 AM	0.07 min	0		MainQueue	TestCase
CopyFile	27306037	Succeeded	2/22/2009 8:06:27 AM	0.02 min	100 (0x00000064)		EQ2	
CAS12450	27306041	Succeeded	2/22/2009 8:06:27 AM	0.63 min	0		MainQueue	TestCase
Trigger	27306043	Succeeded	2/22/2009 8:06:28 AM	0.03 min	100 (0x00000064)		EQ2	
ExactActive	27306046	Succeeded	2/22/2009 8:06:28 AM	0.32 min	0		MainQueue	TestCase
JobB	27306045	Succeeded	2/22/2009 8:06:28 AM	< 0.01 min	100 (0x00000064)		EQ2	
BatchStarts	27306051	Succeeded	2/22/2009 8:06:29 AM	0.13 min	0		MainQueue	TestCase
CCBcc	27306058	Succeeded	2/22/2009 8:06:30 AM	0.62 min	0		MainQueue	TestCase
WIT587	27306056	Aborted	2/22/2009 8:06:30 AM		-805502775 (0xcf...	Canceled	MainQueue	TestCase
Plan	27306059	Succeeded	2/22/2009 8:06:30 AM	0.05 min	0		MainQueue	
Basic	27306072	Failed	2/22/2009 8:06:31 AM	0.17 min	101 (0x00000065)		MainQueue	TestCase
Job	27306083	Succeeded	2/22/2009 8:06:32 AM	0.02 min	100 (0x00000064)		EQ2	
Job	27306073	Succeeded	2/22/2009 8:06:32 AM	0.02 min	100 (0x00000064)		EQ2	
Trigger	27306084	Failed	2/22/2009 8:06:35 AM	0.40 min	102 (0x00000066)	Completion rule fail...	EQ2	
WIT1104	27306086	Succeeded	2/22/2009 8:06:36 AM	0.02 min	0		MainQueue	TestCase
CAS7606	27306085	Succeeded	2/22/2009 8:06:36 AM	0.05 min	0		MainQueue	TestCase

Figure 1.6: Daily activity under one pane of glass.

4: How Do You Manage Cross-System Communications?

Your data center environment already has multiple scheduling engines in place today. Nearly every major business service technology comes with its own mechanism for scheduling its activities. In fact, those mechanisms are likely already performing a set of duties for your services.

Yet the problem, as you can see in Figure 1.7, has to do with the languages each of these platform-specific and application-specific scheduling tools speaks. SQL, for example, comes equipped with a wide range of tools for manipulating SQL data and SQL Server systems; but how rich are those tools when data needs to exit a SQL Server and end up on a UNIX mainframe?

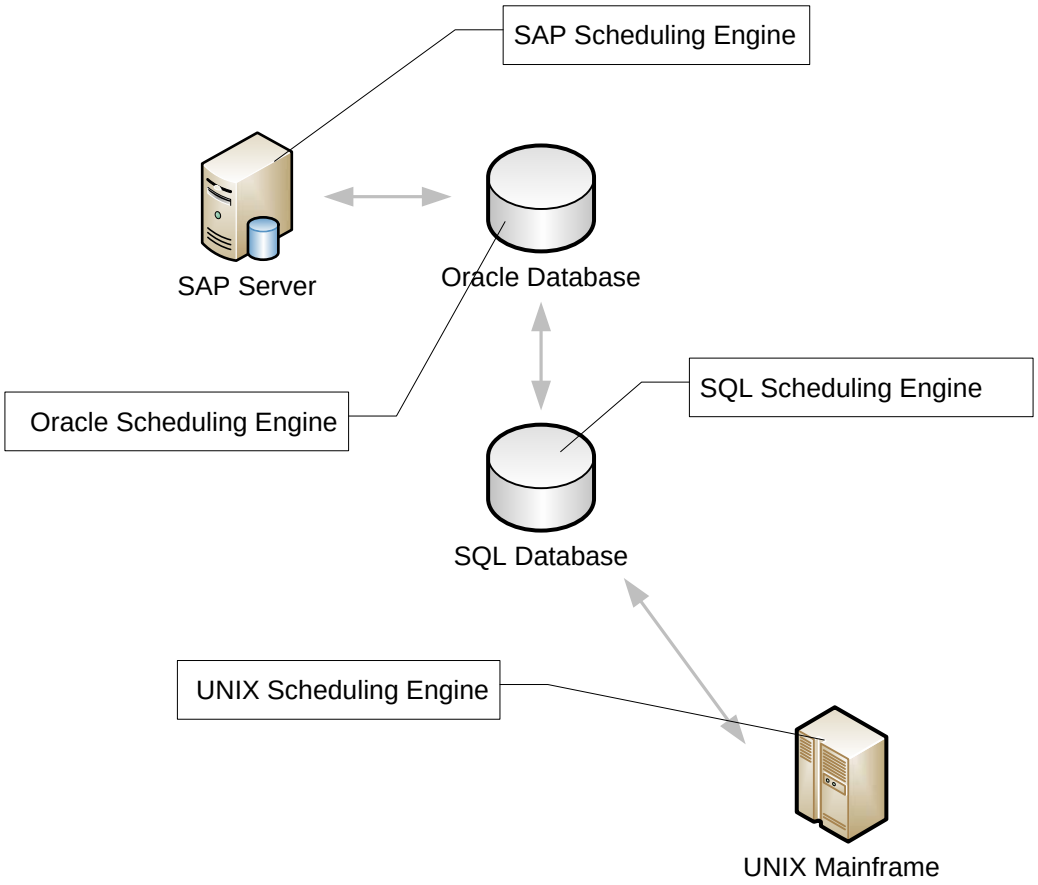


Figure 1.7: Multiple scheduling engines.

Often, the native tools aren't sufficient, forcing an external solution to bridge the gap. That solution can be in the form of individual "little automations" like the scripts this chapter started with. Or they can be wrapped underneath the banner of a holistic job scheduling solution. Chapter 4 will discuss the capabilities you'll want to look for in the best-fit solution.

5: Are You Concerned About Idle Time During a Task?

Considering the answer to question 4, some platform- and application-specific scheduling tools indeed include limited cross-platform support. Their scheduling capabilities may be able to fire jobs based on actions or state changes.

However, one state change that is particularly difficult to measure across platforms is when tasks *take too much time*. Task idling in a state-based scheduling system can cause the entire workbook of plans to come to a halt if not properly compensated for. Essentially, this idle time represents when part of when a task does not complete, leaving the next one waiting.

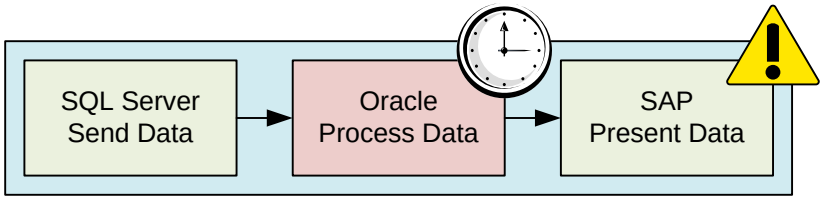


Figure 1.8: Unmanaged task idling can kill a non-automated workflow.

Idling need not necessarily be a problem within a piece of code or script. It can be simply the waiting that is natural in some types of on-system activities. For example, not knowing when a person will submit a file or not knowing when a piece of data is ready for the next step in its processing.

These idle states are notoriously difficult to plan for using time-based scheduling alone. With time-based scheduling, your jobs are built with no intelligence about changes that occur within a system. Rather, they simply run an action at some set point in time. Your job scheduling solution must include the logic necessary to add that intelligence. As you'll learn in later chapters, that intelligence can occur through event-based scheduling or trigger-based scheduling. In either of these cases, an on-system event or trigger recognizes when a change has been made and initiates the next step in processing.

6: How Many Tasks Exist in Your Environment that You Don't Know About?

If you haven't yet standardized on an enterprise job scheduling solution, can you honestly say how many tasks are operating everywhere in your data center? I used to think I knew where all of them were in that former job of mine. But then I left, and took with me the sum total of that knowledge. As I mentioned at the beginning of this chapter, those little automations are still being found years later—often after one of them breaks and causes downtime. More importantly, those are only *my* scripts. There were others in that company as well with scripts of their own that probably eventually got lost.

A centralized job scheduling solution creates a single point of control for automation. It enables auditors and IT teams to know where changes are being sourced from. It is essentially a single point of control, which makes auditors, security officers, and the troubleshooting administrators very happy.

7: Can You Monitor IT Tasks Across Every Team, Platform, and Application?

If you can't, you how can you correlate issues across those teams, platforms, and applications? If you can't, troubleshooting becomes a game of finger-pointing and proving why not.

I was once told a story about a company in real need of an enterprise-wide job scheduling solution. Their business system was much like The Project in that it involved multiple technologies across some very different platforms. Like The Project, managing that system fell to a somewhat distributed group of individuals. SQL Server was managed by the SQL Server team. SAP was administered by SAP administrators. Even the AD had its own group of people responsible for its daily care and feeding.

The problem in this company was not necessarily its application-specific scheduling tools. It was in its people. Those widely-distributed people feared the centralization that a job scheduling solution brings. That fear in part was due to the usual technologist's fear of centralization, but it was also a result of the assumption that a centralized tool would mean re-creating SQL, SAP, AD, and other jobs on a new and completely different system outside their direct control.

An effective enterprise job scheduling solution shouldn't require the complete re-creation of existing jobs within each platform and application. Recall that a job itself represents the change that is to be made, the individual script or package that must be executed. A job scheduling solution represents the wrapper around that invoked action.

This story ends as you'd expect, after a very small but very major problem in one subsystem impacted the system as a whole. Fully unable to track down that minor job with a major impact, the company discovered why centralizing is a good idea.

8: Build Versus Buy: Is a Homegrown Scheduler Good Enough?

I once built my own scheduling system in the now-ancient scripting language of VBScript. VBScript is still in use many places, and it has a long history. But it's known for not having superior built-in methods for scheduling activities. That said, its scheduler worked fine for the task I assigned to it. But the next time I needed a scheduler, I found myself reinventing the wheel. Even with the limited code modularization VBScript can present into a script, my scheduler's reusability was very limited.

Imagine having to replicate that scheduling across multiple applications and platforms using different languages—and even using different approaches, both object-oriented and structured. Homegrown schedulers are indeed an acceptable way of handling the triggering needs of individual scripts and packages; however, a global scheduler that works across all jobs and plans obviously creates a superior framework for job execution.

More importantly, the human resources that are necessary to keep a homegrown job scheduler can be much greater than they seem at first blush. Those resources need to keep an eye on the logical code itself alongside the jobs that the scheduler attempts to run. In many cases, you'll find that the extra costs associated with creating your own scheduler include the fact that this project will take away from time that could be better served by working on other more value-oriented projects.

9: What Are Your Steps for Linking the Results of One Task with the Actions of Another?

Nowhere is that value-add more pertinent than when two jobs rely on each other. This kind of job construction happens all the time within distributed systems. In it, the first task in a string completes with a set of data. That data is needed by the next task in the string. This sharing of information can be handled through file drop-boxes or richer mechanisms like Microsoft Message Queue or database triggers.

Like with the problem of multiple languages, these queuing solutions tend towards being very platform-centric. It becomes very difficult using a database trigger to invoke an action in AD, for example. A centralized job scheduling solution with rich support for applications will become the central point of control for all cross-task action linking.

10: How Do You Handle Errors in a Custom-Coded Script?

Last is the handling of error messages in custom-coded scripts, a process that itself consumes a vast quantity of script development time. Errors are notoriously difficult to track down, and become even more challenging when scripts need to span platforms and applications. Error handling requires special skills in trapping variables and determining their intended and actual values. All of these activities grow even more difficult when scripts are run automatically as opposed to interactively because error messages in many cases cannot be captured. Chapter 3 will go into greater detail on the error-handling functionality of a good job scheduler. For now, recognize that a homegrown script without error handling is ripe for troubleshooting headaches down the road.

Do You Need Job Scheduling?

So do you have good answers to these questions? Do you feel that your existing scheduling tools bring you zero headaches? If yes, then thanks for reading. If no, then it is likely that your next thoughts will be toward the types of IT challenges that an enterprise job scheduling solution can support. With the basics discussed and the initial questions answered, the next chapter introduces seven real-world use cases for automating IT job scheduling. It should be an interesting read because the types of use cases outlined in that chapter are probably pretty close to those you've already got under management today.