Microsoft

SPECIAL EXCERPT
Complete book available Spring 2012

Introducing Microsoft® SQL Server® Code Name "Denali"

PREVIEW CONTENT

Ross Mistry and Stacia Misner

PREVIEW CONTENT

This excerpt provides early content from a book currently in development, and is still in draft, unedited format. See additional notice below.

About the Authors

Ross Mistry is a Principal Enterprise Architect with Microsoft, working out of the Microsoft Technology Center in Silicon Valley. He designs solutions for Microsoft largest customers and specializes in SQL Server high availability, appliances, consolidation, virtualization, and private cloud.

Ross is also an author, community champion, and seasoned architect. He has a tremendous amount of experience designing and deploying technology solutions for Internet startups and fortune 100 organizations located in the Silicon Valley. He is known in the world-wide community for his expertise in SQL Server, Windows, Exchange, Virtualization, and Private Cloud. Ross frequently speaks at technology conferences around the world and has published many books, whitepapers, and magazine articles. His recent books include *Introducing SQL Server 2008 R2* (Microsoft Press), *Windows Server 2008 R2 Unleashed* (SAMS) and *SQL Server 2008 Management and Administration* (SAMS).

You can follow him on Twitter @RossMistry.

Stacia Misner is a consultant, educator, mentor, and author specializing in Business Intelligence solutions since 1999. During that time, she has authored or co-authored multiple books about BI. Stacia provides consulting and custom education services through Data Inspirations and speaks frequently at conferences serving the SQL Server community. She writes about her experiences with BI at *blog.datainspirations.com*, and tweets as @StaciaMisner.

CONTENTS

CHAPTER 2

High Availability and Disaster Recovery Enhancements 1

CHAPTER 6

Integration Services 21

Chapter 2

High Availability and Disaster Recovery Enhancements

Microsoft SQL Server Denali delivers significant enhancements to well-known, critical capabilities like high availability and disaster recovery. These enhancements promise to assist organizations in achieving the highest mission-critical confidence to date. Server Core support along with breakthrough features like AlwaysOn Availability Groups, active secondaries, and key improvements to features such as failover clustering now offer organizations a range of accommodating options to achieve maximum application availability and data protection for SQL Server instances and databases within a datacenter and across datacenters.

With SQL Server's heavy investment in AlwaysOn, this chapter's goal is to bring readers up-to-date with the high availability and disaster recovery capabilities that are fully integrated into SQL Server Denali.

SQL Server AlwaysOn: An Integrated Solution

Every organization's success and service reputation is built on ensuring that its data is always accessible and protected. In the IT world, this means delivering a product that achieves the highest level of availability and disaster recovery while minimizing data loss and downtime. With the previous versions of SQL Server, organizations achieved high availability and disaster recovery by using technologies such as failover clustering, database mirroring, log shipping and peer-to-peer replication. Although organizations achieved great success with these solutions, they were tasked with combining these native SQL Server technologies to achieve their business requirements affiliated with Recovery Point Objective (RPO) and Recovery Time Objective (RTO).

Figure 2-1 illustrates a very common high availability and disaster recovery strategy used by organizations with the previous versions of SQL Server. This strategy includes failover clustering to protect SQL Server Instances within each datacenter combined with asynchronous database mirroring to provide disaster recovery capabilities for mission critical databases.

FIGURE 2-1 Achieving high availability and disaster recovery with failover clustering combined with database mirroring in SQL Server 2008 R2.

Likewise, the high availability and disaster recovery deployment for organizations that either required more than one secondary datacenter or who did not have shared storage incorporated synchronous database mirroring with a witness within the primary datacenter combined with Log Shipping for moving data to multiple locations. This deployment strategy is illustrated in Figure 2-2.

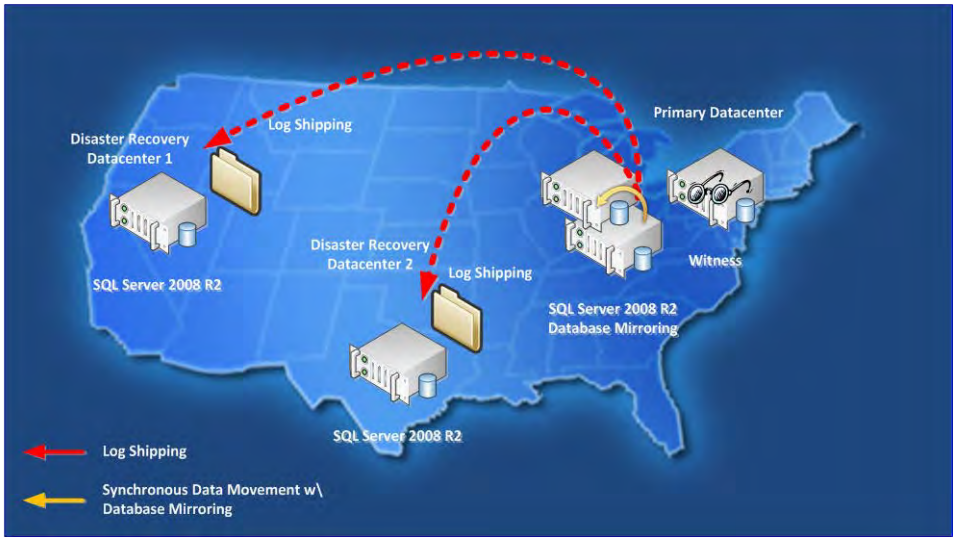


FIGURE 2-2 Achieving high availability and disaster recovery with database mirroring combined with Log Shipping in SQL Server 2008 R2.

Both Figures 2-1 and 2-2 reveal successful solutions for achieving high availability and disaster recovery. However, the approach to these solutions was fragmented instead of seamless which warranted changes. In addition, with organizations constantly evolving, it was only a matter of time until they voiced their own concerns and sent out a request for more options and changes.

One concern for many organizations was directed at database mirroring. Database mirroring is a great way to protect databases; however, the solution is a one-to-one mapping, making multiple secondaries unattainable. When confronted with this situation, many organizations reverted to Log Shipping as a replacement for database mirroring because it supports multiple secondaries. Unfortunately, organizations encountered limitations with Log Shipping because it did not provide zero data loss or automatic failover capability. Concerns were also experienced by organizations working with failover clustering because they felt that their shared storage devices, such as a SAN, were a single point of failure. Similarly, many organizations thought that from a cost perspective their investments were not being used to their full potential. For example, the passive servers in many of these solutions were running idle. Finally, many organizations wanted to offload reporting and maintenance tasks from the primary database servers, which was not an easy task to achieve.

SQL has evolved to answer many of these concerns and this includes an integrated solution called *AlwaysOn*. *AlwaysOn Availability Groups* and *AlwaysOn Failover Cluster Instances* are new features, introduced in SQL Server Denali, that are rich with options and promise the highest level of availability and disaster recovery to its customers. At a high level, *AlwaysOn Availability Groups* are used for database protection and offer multi-database failover, multiple secondaries, active secondaries, and integrated HA management. On the other hand, *AlwaysOn Failover Cluster Instances* are tailored towards instance-level protection and multi-site clustering and consolidation while consistently providing flexible failover polices and improved diagnostics.

AlwaysOn Availability Groups

The AlwaysOn Availability Groups provide an enterprise-level alternative to database mirroring and give organizations the ability to automatically or manually failover a group of databases as a single unit with support for up to four secondaries. The solution provides zero data loss protection and is flexible. It can be deployed on local storage or shared storage, and it supports both synchronous and asynchronous data movement. The application failover is very fast, supports automatic page repair, and the secondary replicas can be leveraged to offload reporting and a number of maintenance tasks such as backups.

Take a look at Figure 2-3 which simply illustrates an AlwaysOn Availability Group deployment strategy that includes one primary replica and three secondary replicas.

FIGURE 2-3 Achieving high availability and disaster recovery with AlwaysOn Availability Groups.

In this figure, synchronous data movement is used to provide high availability within the primary datacenter and asynchronous data movement is used to provide disaster recovery. Moreover, secondary replica 3 and replica 4 are employed to offload reports and backups from the primary replica.

It is now time to carry out a deeper dive on AlwaysOn Availability Groups through a review of the new concepts and terminology associated with this breakthrough capability.

Understanding Concepts and Terminology

Availability groups are built on top of Windows Failover Clustering and support both shared and non-shared storage. Depending on an organization's Recovery Point Objective (RPO) and Recovery Time Objective (RTO) requirements, availability groups can use either an asynchronous-commit mode or a synchronous-commit mode to move data between primary and secondary replicas. Availability groups include built in compression and encryption as well as a support for file-stream replication. Failover between replicas are either automatic or manual.

When deploying AlwaysOn Availability Groups, the first step is to deploy a Windows Failover Cluster. This is completed by using the Failover Cluster Manager Snap-In within Windows Server 2008 R2. Once the Windows Failover Cluster is formed, the remainder of the Availability group configurations is completed in SQL Server Management Studio. When using the Availability Group Wizards to configure availability groups, SQL Server Management Studio automatically creates the appropriate services, applications, and resources in Failover Cluster Manager.

Now that the fundamentals of the AlwaysOn Availability Group has been laid down, the most natural question that follows is how is an organization's operations are enhanced with this feature. Unlike

database mirroring which supports only one secondary, AlwaysOn Availability Groups support one primary replica and up to four secondary replicas. Availability groups can also contain more than one availability database. Equally appealing, it is possible to host more than one availability group within the solution. As a result, it is possible to group databases with application dependencies together within an availability group and have all the availability databases seamlessly failover as a single cohesive unit as depicted in Figure 2-4.

FIGURE 2-4 Dedicated availability groups for Finance and HR availability databases.

In addition, as shown in Figure 2-4, there is one primary replica and two secondary replicas with two availability groups. One of these availability groups is called Finance and it includes all the Finance databases; the other availability group is called HR and it includes all the Human Resources databases. The Finance Availability Group can failover independently of the HR availability group and unlike database mirroring, all availability databases within an availability group failover as a single unit. Moreover, organizations can improve their IT efficiency, increase performance, and reduce total cost of ownership with better resource utilization of secondary hardware because these secondary replicas can be leveraged for backups and read-only operations such as reporting and maintenance. This is covered in the "Active Secondaries" section later in this chapter.

Now that you have been introduced to some of the benefits the AlwaysOn Availability Group offers for an organization's database, take the time to get a stronger understanding of the AlwaysOn Availability Group concepts and how the new capability operates. The concepts covered include:

- Availability Replica Roles
- Failover and Synchronization Modes
- Data Synchronization Options
- Connection Mode in Secondaries
- Availability Group Listener

Availability Replica Roles

Each AlwaysOn Availability Group is comprised of a set of two or more failover partners that are referred to as *availability replicas*. The availability replicas can consist of either a primary role or a secondary role. It is worth noting that there can be a maximum of four secondaries and of these four secondaries only a maximum of two secondaries can be configured in synchronous-commit mode.

The *roles* affiliated with the availability replicas in AlwaysOn Availability Groups follow the same principals as the legendary Sith rule of two doctrines in the Star Wars[1] saga. In Star Wars, there can only be two Sith at one time, a master and an apprentice. Similarly, a SQL Server instance in an availability group can only be a primary replica or a secondary replica. At no time can it be both because the role swapping is controlled by Windows Server Failover Cluster (WSFC).

Each of the SQL Server instances in the availability group is hosted on either a SQL Server Failover Cluster Instance (FCI) or a stand-alone instance of SQL Server Denali. Each of these instances resides on different nodes of a WSFC. WSFC is typically used for providing high availability and disaster recovery for well-known Microsoft products. As such, availability groups use WSFC as the underlying mechanism to provide inter-node health detection, failover coordination, primary health detection, and distributed change notifications for the solution.

Each availability replica hosts a copy of the availability databases in the availability group. Since there are multiple copies of the databases being hosted on each availability replica, there isn't a prerequisite for utilizing shared storage like there was in the past when deploying traditional SQL Server Failover Clusters. On the flip side, when using non-shared storage, an organization must keep in mind that storage requirements increase depending on the number of replicas it plans on hosting.

Failover and Synchronization Modes

When configuring AlwaysOn Availability Groups, database administrators can choose from three modes to distribute data from the primary to the secondaries. For those of you who are familiar with

[1] Trademark of Lucasfilm Ltd.

database mirroring, the modes for providing data movement to obtain high availability and disaster recovery are very similar to the modes in database mirroring. The three modes available when using the New Availability Group Wizard include:

- **Automatic failover** This synchronous-commit mode supports both automatic failover and manual failover.
- **High safety** This synchronous-commit mode supports only manual failovers.
- **High performance** This asynchronous-commit mode supports only forced failover with possible data loss.

Data Synchronization Modes

To move data from the primary replica to the secondary replica, each mode uses either synchronous-commit or asynchronous-commit mode. Give consideration to the following items when selecting either option:

- When using the *synchronous-commit* mode, a transaction is committed on both replicas to guarantee transactional consistency. This, however, means increased latency. As such, this option might not be appropriate for partners who don't share a high-speed network or reside in different geographical locations.
- The *asynchronous-commit* mode commits transactions between partners without waiting for the partner to write the log to disk. This maximizes performance and is well suited for disaster-recovery solutions.

Connection Mode in Secondaries

As indicated earlier, each of the secondaries can be configured to support read-only access for reporting or other maintenance tasks such as backups. During the final configuration stage of the AlwaysOn Availability Groups, database administrators decide on the connection mode for the secondary replicas. There are three connections modes available:

- **Disallow connections** In the secondary role, this availability replica does not allow any connections.
- **Allow only read-intent connections** In the secondary role, this availability replica allows only read-intent connections.
- **Allow all connections** In the secondary role, this availability replica allows all connections for read access, including connections running with older clients.

Availability Group Listeners

The Availability Group Listener provides a way of connecting to an availability group via a virtual network name that is bound to the primary replica. Applications can specify the network name affiliated with the Availability Group Listener in connection strings. After the availability group fails over from the

primary replica to a secondary replica, the network name directs connections to the new primary replica. The Availability Group Listener concept is very similar to a Virtual SQL Server Name when using failover clustering; however, with an Availability Group Listener, there is a virtual network name for each availability group, whereas with SQL Server failover clustering, there is one virtual network name for the instance.

You can specify your Availability Group Listener preferences when using the Create a New Availability Group Wizard in SQL Server Management Studio or you can manually create or modify an Availability Group Listener after the availability group is created. Alternatively, you can use Transact-SQL to create or modify the listener. Notice in Figure 2-5 that each Availability Group Listener requires a DNS name, an IP Address, and a Port such as 1433. Once the Availability Group Listener is created, a server name and an IP address cluster resource are automatically created within Failover Cluster Manager. This is certainly a testimony to the availability group's flexibility and tight integration with SQL Server as the majority of the configurations are done within SQL Server.

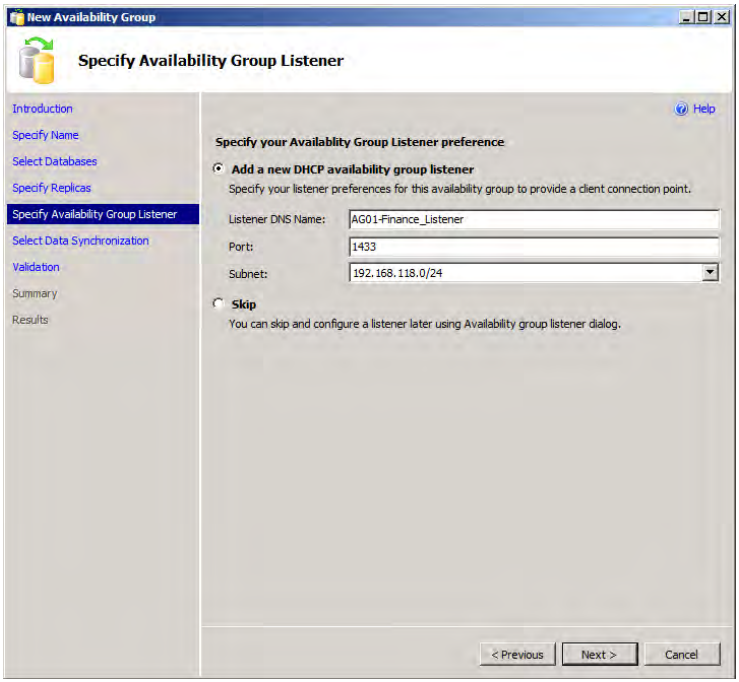


FIGURE 2-5 Specifying Availability Group Listener Properties.

It is worth stating that there is a one-to-one mapping between Availability Group Listeners and availability groups. This means you can create one Availability Group Listener for each availability group. However, if more than one availability group exists within a replica, it is possible to have more than one Availability Group Listener. For example, there are two availability groups shown in Figure 2-6; one is for the Finance availability databases and the other is for the HR availability databases. Each availability group has its own Availability Group Listener that clients and applications connect to.

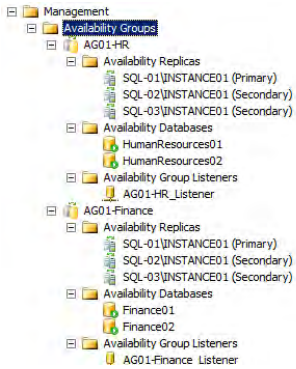


FIGURE 2-6 Illustrating two Availability Group Listeners within a replica.

Configuring Availability Groups

When creating a new availability group, a database administrator needs to specify an availability group name such as AvailablityGroupFinance, and then select one or more databases to partake in the availability group. The next step involves first specifying one or more instances of SQL Server to host secondary availability replicas and then specifying your Availability Group Listener preference. The final step is selecting the data synchronization preference and connection mode for the secondary replicas. These configurations are conducted with the New Availability Group Wizard or with Transact-SQL PowerShell scripts.

Prerequisites

To deploy AlwaysOn Availability Groups, the following prerequisites must be met:

- All computers running SQL Server, including the servers that will reside in the disaster recovery site, must reside in the same Windows-based domain.
- All SQL Server computers must partake in a single Windows Server failover cluster even if the servers reside in multiple sites.
- A Windows Server failover cluster must be formed.
- AlwaysOn Availability Groups must be enabled on each server.
- All the databases must be in full recovery mode.
- A full backup must be conducted on all databases before deployment.

Deployment Examples

Figure 2-7 illustrates the Specify Replicas page in the New Availability Group Wizard. In this example, there are three SQL Server instances in the availability group called Finance: SQL01\Instance01, SQL02\Instance01, and SQL03\Instance01. SQL01\Instance01 is configured as the Primary Replica whereas SQL02\Instance01 and SQL03\Instance01 are configured as secondaries. SQL02\Instance01 and SQL03\Instance01 support Automatic Failover with Synchronous data movement, whereas

SQL-03\Instance01 uses Asynchronous-commit availability mode and only supports a forced failover. Finally, SQL01\Instance01 and SQL02\Instance01 do not allow connections to the secondary. SQL03\Instance01 only allows read-intent connections. SQL01\Instance01 and SQL02\Instance01 reside in a primary datacenter for high availability within a site and SQL03\Instance01 can reside in the disaster recovery datacenter and be brought online manually in the event the primary datacenter becomes unavailable.

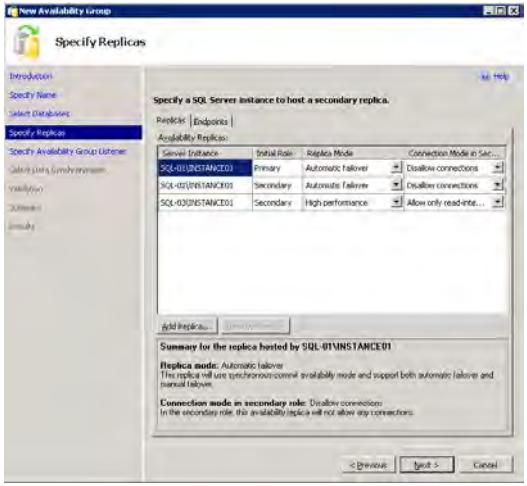


FIGURE 2-7 Specifying the SQL Server instances in the availability group.

One thing becomes vividly clear from Figure 2-7 and the preceding example: there are many different deployment configurations available to satisfy any organization's high availability and disaster recovery requirements. See Figure 2-8 for additional deployment alternatives.

FIGURE 2-8 Additional AlwaysOn Deployment Alternatives.

Monitoring Availability Groups with the Dashboard

Administrators have an opportunity to leverage a new and remarkably intuitive manageability dashboard in in SQL Server Denali to monitor availability groups. The dashboard, as shown in Figure 2-9 reports the health and status associated with each instance and availability database in the availability group. Moreover, the dashboard displays the specific replica role of each instance and provides synchronization status. If there is an issue or more information on a specific event is required, a database administrator can click the Availability Group State, Server Instance name, or health status hyperlinks for additional information. The dashboard is launched by right-clicking the Availability Groups Folder and selecting Show Dashboard.

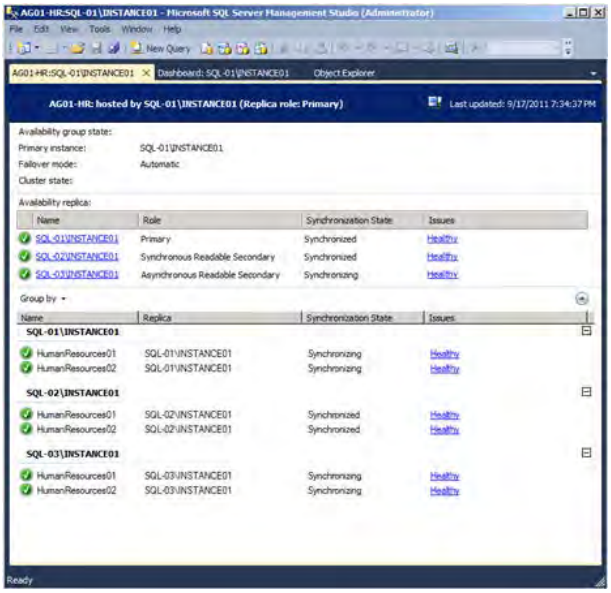


FIGURE 2-9 Monitoring availability groups with the dashboard.

Active Secondaries

As indicated earlier, many organizations communicated to the SQL Server team their need to improve IT efficiency by optimizing their existing hardware investments. Specifically, organizations hoped their production systems for passive workloads could be used in some other capacity instead of remaining in an idle state. These same organizations also wanted reporting and maintenance tasks offloaded from production servers because these tasks negatively impacted production workloads. With SQL Server Denali, organizations can leverage the AlwaysOn Availability Group capability to configure a secondary replica, also referred to as *Active Secondaries*, to provide read-only access to databases affiliated with an availability group.

All read-only operations on the secondary replicas are supported by row versioning and are automatically mapped to snapshot isolation transaction level, which eliminates reader/writer contention. In addition, the data in the secondary replicas is near real time. In many circumstances, data latency between the primary and secondary databases should be within seconds. It is worth noting that the latency of log synchronization impacts data freshness.

For organizations, active secondaries are synonymous with performance optimization on a primary replica and increases to overall IT efficiency and hardware utilization.

Read-Only Access to Secondary Replicas

Recall that when configuring the connection mode for secondary replicas, you can Disallow Connections, Allow Only Read-Intent Connections, and Allow All Connections. Allow Only Read-Intent Connections and Allow All Connections both provide read-only access to secondary replicas. The Disallow Connections alternative does not allow read-only access as implied by its name.

Now let's look at the major differences between Allow Only Read-Intent Connections and Allow All Connections. The Allow Only Read-Intent Connections option allows connections to the databases in the secondary replica when the Application Intent connection property is set to Read-only in the SQL Server Native Client. When using the Allow All Connection settings, all client connections are allowed independent of the Application Intent property. What is the Application Intent property in the connection string? The Application Intent property declares the application workload type when connecting to a server. The possible values are Read-only and Read Write. Commands that try to create or modify data on the secondary replica will fail.

Backups on Secondary

Backups of availability databases participating in availability groups can be conducted on any of the replicas. Although backups are still supported on the primary replica, log backups can be conducted on any of the secondaries. It is worth noting that this is independent of the replication commit mode being used—synchronous-commit or asynchronous-commit. Log backups completed on all replicas form a single log chain, as shown in Figure 2-10.

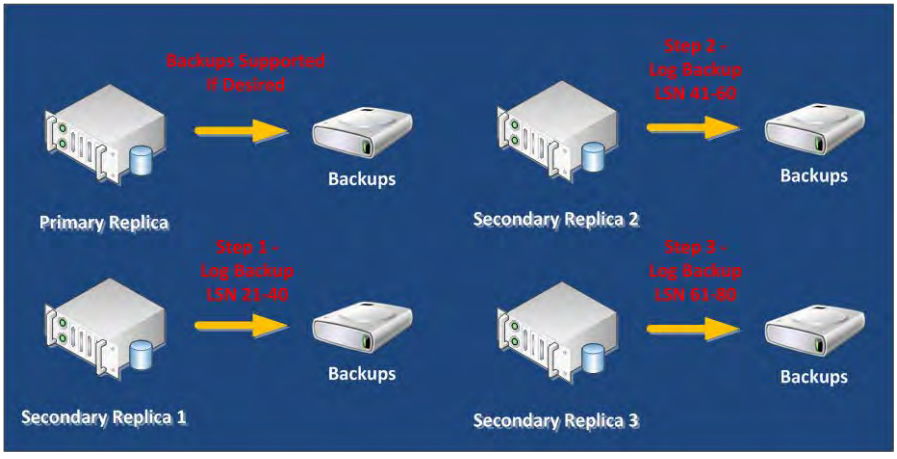


FIGURE 2-10 Forming a single log chain by backing up the transaction logs on multiple secondary replicas.

As a result, the transaction log backups do not all have to be performed on the same replica. This in no way means that serious thought should not be given to the location of your backups. It is recommended to store all backups in a central location because all transaction log backups are required to perform a restore in the event of a disaster. Therefore, if a server is no longer availability and it con-

tained the backups, you will be negatively affected. In the event of a failure, use the new Database Recovery Advisor Wizard; it provides many benefits when conducting restores. For example, if performing backups on different secondaries, the wizard generates a visual image of a chronological timeline by stitching together all of the log files based on the Log Sequence Number (LSN).

AlwaysOn Failover Cluster Instances (FCI)

You've seen the results of the development efforts in engineering the new AlwaysOn Availability Groups capability for high availability and disaster recovery, and the creation of active secondaries. Now you'll explore the significant enhancements to traditional capabilities like SQL Server failover clustering that leverages shared storage. The following list itemizes some of the improvements that will appeal to database administrators looking to gain high availability for their SQL Server instances. Specifically, this section discusses the following features:

- **Multi-Subnet Clustering** This feature provides a disaster recovery solution in addition to high availability with new support for multi-subnet failover clustering.
- **Support for TempDB on Local Disk** Another storage level enhancement with failover clustering is associated with TempDB. TempDB no longer has to reside on shared storage as it did in previous versions of SQL Server. It is now supported on local disks, which results in many practical benefits for organizations. For example, it is now possible to offload TempDB I/O from share storage devices like a SAN and leverage fast SSD storage locally within the server nodes to optimize TempDB workloads, which are typically random I/O.
- **Flexible Failover Policy** Denali introduces improved failure detection for the SQL Server Failover Cluster Instance (FCI) by adding failure condition-level properties which allow you to configure a more flexible failover policy.

Note It is also worth mentioning that AlwaysOn Failover Clustering Instances can be combined with Availability Groups to offer maximum SQL Server instance and database protection.

With the release of Windows Server 2008, new functionality enabled cluster nodes to be connected over different subnets without the need for a stretch VLAN across networks. The nodes could reside on different subnets within a datacenter or in another geographical location such as a disaster recovery site. This concept is commonly referred to as *multi-site clustering*, *multi-subnet clustering*, or *stretch-clustering*. Unfortunately, the previous versions of SQL Server could not take advantage of this Windows failover clustering feature. Organizations that wanted to create either a multi-site or multi-subnet SQL Server failover cluster still had to create a stretch VLAN to expose a single IP address for failover across sites. This was a complex and challenging task for many organizations. This is no longer the case because SQL Server Denali supports multi-subnet and multi-site clustering out-of-the-box; therefore, the need for implementing stretch VLAN technology no longer exists.

Figure 2-11 illustrates an example of a SQL Server multi-subnet failover cluster between two subnets spanning two sites. Notice how each node affiliated with the multi-subnet failover cluster resides on a different subnet. Node 1 is located in Site 1 and resides on the 192.168.115.0/24 subnet whereas; Node 2 is located in Site 2 and resides on the 192.168.116.0/24 subnet.

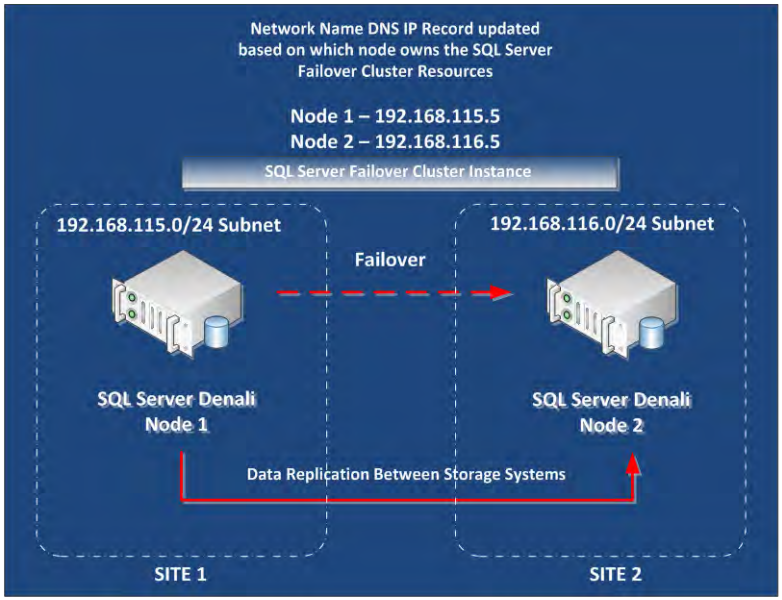


FIGURE 2-11 A multi-subnet failover cluster instance example.

For clients and applications to connect to the SQL Server failover cluster, they need two IP addresses registered to the SQL Server failover cluster resource name in WSFC. For example, imagine your server name is SQLFCI01 and the IP addresses are 192.168.115.5 and 192.168.116.5. WSFC automatically controls the failover and brings the appropriate IP address online depending on the node that currently owns the SQL Server resource. Again, if Node 1 is affiliated with the 192.168.115.0/24 subnet and owns the SQL Server failover cluster then the IP address resource 192.168.115.6 is brought online as shown in Figure 2-12. Similarly, if a failover occurs and Node 2 owns the SQL Server resource, then IP address resource 192.165.115.6 is taken offline and the IP address resource 192.168.116.6 is brought online.

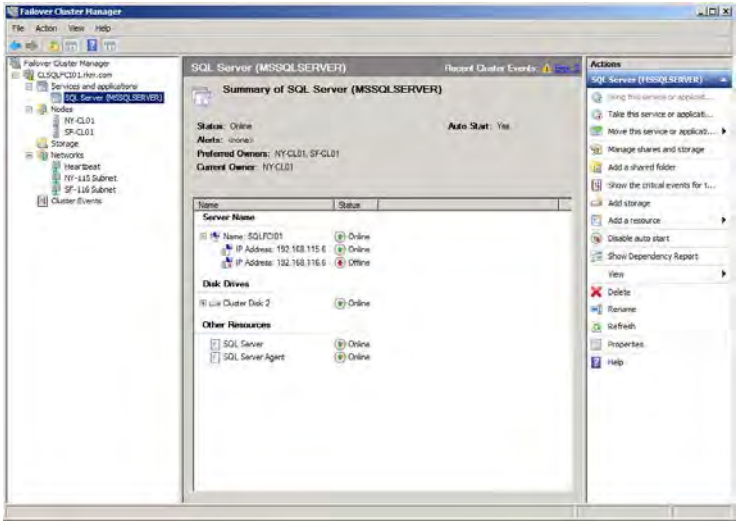


FIGURE 2-12 Multiple IP addresses affiliated with a multi-subnet failover cluster instance screenshot.

Since there are multiple IP addresses affiliated with the SQL Server failover cluster instance virtual name, the online address will change automatically when there is a failover. In addition, Windows failover cluster will issue a DNS update immediately after the network name resource name comes online. The IP address change in DNS might not take effect on clients due to cache settings, therefore, it is recommended to minimize the client downtime by configuring the HostRecordTTL in DNS to 60 seconds.

Support for Deploying SQL Server Denali on Windows Server Core

Windows Server Core was originally introduced with Windows Server 2008 and saw significant enhancements with the release of Windows Server 2008 R2. For those who are unfamiliar with Server Core, it is an installation option for the Windows Server 2008 and Windows Server 2008 R2 operating systems. Since Server Core is a minimal deployment of Windows, it is much more secure because its attack surface is greatly reduced. Server Core does not include a traditional Windows graphical interface and, therefore, is managed via a command prompt or by remote administration tools.

Unfortunately, previous versions of SQL Server did not support the Server Core operating system, but that has all changed. For the first time, Microsoft SQL Server "Denali" supports Server Core installations for organizations running Server Core based on Windows Server 2008 R2 with Service Pack 1 or later.

Why is Server Core so important to SQL Server and how does it positively impact availability? When running SQL Server Denali on Server Core, operating system patching is drastically reduced—by up to

60 percent. This translates to higher availability and a reduction in planned downtime for any organization's mission-critical databases and workloads. In addition, surface area attacks are greatly reduced and overall security of the database platform is strengthened, which again translates to maximum availability and data protection.

When first introduced, Server Core required the use and knowledge of command line syntax to manage it. Most IT professionals at this time were accustomed to using a graphical user interface (GUI) to manage and configure Windows so they had a difficult time embracing Server Core. This impacted its popularity and, ultimately, its implementation. To ease these challenges, Microsoft introduced SCONFIG. SCONFIG is an out-of-the-box utility that was introduced with the release of Windows Server 2008 R2 to dramatically ease server configurations. To navigate through the SCONFIG options, you only need to type one or more numbers to configure server properties as displayed in Figure 2-13.

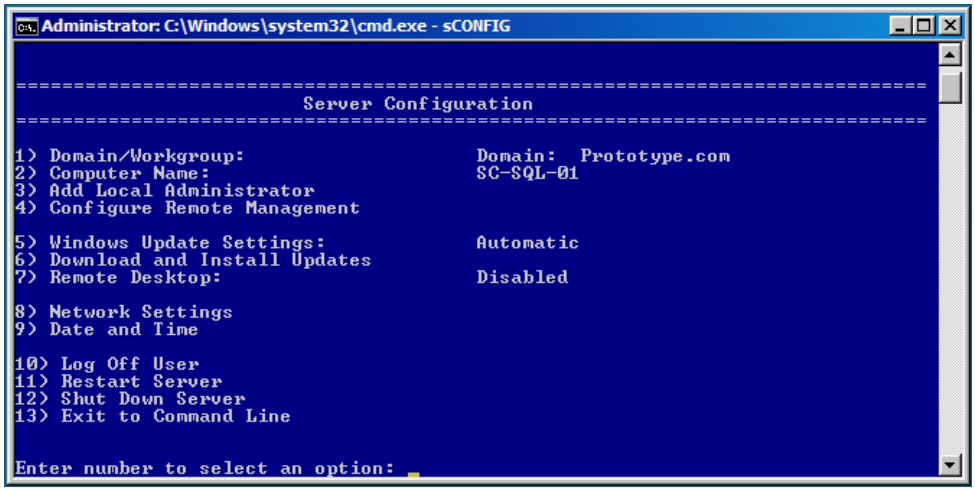


FIGURE 2-13 SCONFIG utility for configuring server properties in Server Core.

The following sections articulate the SQL Server Denali prerequisites for Server Core, SQL Server features supported on Server Core, and the installation alternatives.

SQL Server Denali Prerequisites for Server Core

Organizations installing SQL Server Denali on Windows Server 2008 R2 Server Core must meet the following operating system, features, and components prerequisites:

Operating system:

- Windows Server 2008 R2 SP1 64-bit x64 Data Center Server Core
- Windows Server 2008 R2 SP1 64-bit x64 Enterprise Server Core
- Windows Server 2008 R2 SP1 64-bit x64 Standard Server Core
- Windows Server 2008 R2 SP1 64-bit x64 Web Server Core

Features and components:

- .NET Framework 2.0 SP2
- .NET Framework 3.5 SP1 Full Profile
- .NET Framework 4 Server Core Profile
- Windows Installer 4.5
- Windows PowerShell 2.0

Once the prerequisites are fulfilled, it important to become familiar with the SQL Server components supported on Server Core.

SQL Server Features Supported on Server Core

There are numerous SQL Server features that are fully supported on Server Core. They include Database Engine Services, SQL Server Replication, Full Text Search, Analysis Services, Client Tools Connectivity and Integration Services. Likewise, Sever Core does not support the many other features including Reporting Services, Business Intelligence Development Studio, Client Tools Backward Compatibility, Client Tools SDK, SQL Server Books Online, Distributed Replay Controller, SQL Client Connectivity SDK, Master Data Services and Data Quality Services. Some features like Management Tools – Basic, Management Tools – Complete, Distributed Replay Client and Microsoft Sync Framework are only supported remotely. Therefore, these features can be installed on editions of the Windows operating system that are not Server Core, and then used to remotely connect to a SQL Server instance running on Server Core.

> **Note** To leverage Server Core, it is important to plan your SQL Server installation ahead of time. Give yourself the opportunity to fully understanding which SQL Server features that are required to support your mission critical workloads.

SQL Server on Server Core Installation Alternatives

The typical SQL Server Installation Setup Wizard is not supported when installing SQL Server Denali on Server Core. As a result, there is a need to automate the installation process by either using a command-line installation, a configuration file, or leveraging the DefaultSetup.ini methodology. Details and examples for each of these methods can be found in Books Online.

> **Note** When installing SQL Server Denali on Server Core, ensure you use Full Quiet mode by using the /Q parameter or the Quiet Simple mode by using the /QS parameter.

Additional High Availability and Disaster Recovery Enhancements

This section summarizes some of the new features and enhancements you can expect to see.

Support for Server Message Block

A common movement for organizations in recent years has been towards consolidating databases and applications onto few servers—specifically, hosting many instances of SQL Server running on a failover cluster. When using failover clustering for consolidation, the previous versions of SQL Server required a single drive letter for each SQL Server failover cluster instance. Because there are only 23 drive letters available, without taking into account reservations, the maximum amount of SQL Server instances supported on a single failover cluster was 23. Twenty-three instances sounds like an ample amount; however, the drive letter limitation negatively impacts organizations running powerful servers that have the compute and memory resources to host more than 23 instances on a single server. Going forward, SQL Server Denali and failover clustering introduces support for Server Message Block (SMB).

> **Note** Some of you must be thinking you can use mount points to alleviate the drive letter pain point. When working with previous versions of SQL Server, even with mount points, you require at least one drive letter for each SQL Server Failover Cluster Instance.

Some of the SQL Server Denali benefits brought about by SMB are, of course, database storage consolidation and the potential to support more than 23 clustering instances in a single WSFC. To take advantage of these features, the file servers must be running Windows Server 2008 or later versions of the operating system.

Database Recovery Advisory

The Database Recovery Advisor is a new feature aimed at optimizing the restore experience for database administrators conducting database recovery tasks. This tool includes a new timeline feature that provides a visualization of the backup history, as shown in Figure 2-14.

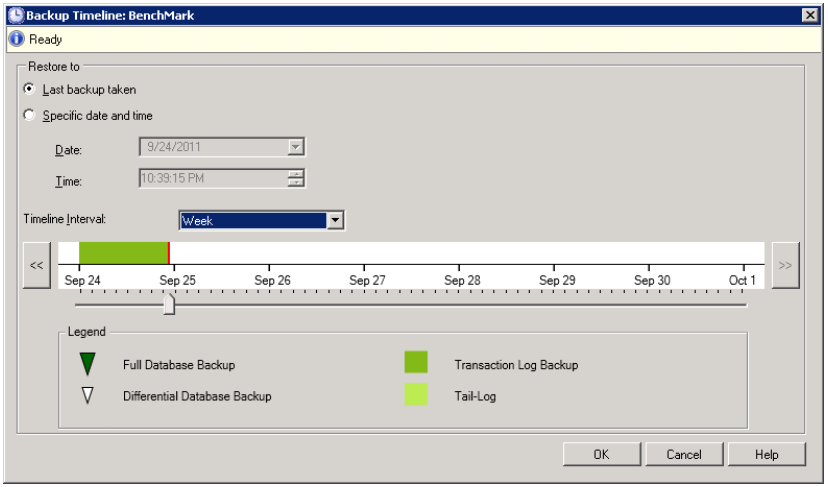


FIGURE 2-14 Database Recovery Advisory backup and restore visual timeline.

Online Operations

SQL Server Denali also includes a few enhancements for online operation that reduce downtime during planned maintenance operations. Line of business (LOB) re-indexing and adding columns with defaults are now supported.

Rolling Upgrade and Patch Management

All of the new AlwaysOn capabilities reduce application downtime to only a single manual failover by supporting rolling upgrades and patching of SQL Server. This means a database administrator can apply a service pack or critical fix to the passive nodes if using a failover cluster or secondary replicas if using availability groups. Once the installation is complete on all passive nodes or secondaries, a database administrator can conduct a manual failover and then apply the service pack or critical fix to the node in a FCI or replica. This rolling strategy also applies when upgrading the database platform.

Chapter 6

Integration Services

Since its initial release in Microsoft SQL Server 2005, Integration Services has had incremental changes in subsequent versions of the product. However, those changes were trivial in comparison to the number of enhancements, performance improvements, and new features introduced in SQL Server Code-Named "Denali" Integration Services. This product overhaul affects every aspect of Integration Services, from development to deployment to administration.

Developer Experience

The first change that you notice as you create a new Integration Services project is that Business Intelligence Development Studio (BIDS) is now a Visual Studio 2010 shell. The Visual Studio environment alone introduces some slight user interface changes from the previous version of BIDS. However, there are several more significant interface changes of note that are specific to Integration Services. These enhancements to the interface help you to learn about the package development process if you are new to Integration Services, and enable you to develop packages more easily if you are already have experience with Integration Services. If you are already an Integration Services veteran, you will also notice the enhanced appearance of tasks and data flow components with rounded edges and new icons.

Add New Project Dialog Box

To start working with Integration Services in BIDs, you create a new project by following the same steps that you use to perform the same task in earlier releases of Integration Services. From the File menu, point to New, and then select Project. The Add New Project dialog box displays. In the Installed Templates list, you can now select the type of Business Intelligence template you want to use and then view only the templates related to your selection, as shown in Figure 6-1. When you select a template, a description of the template displays on the right side of the dialog box.

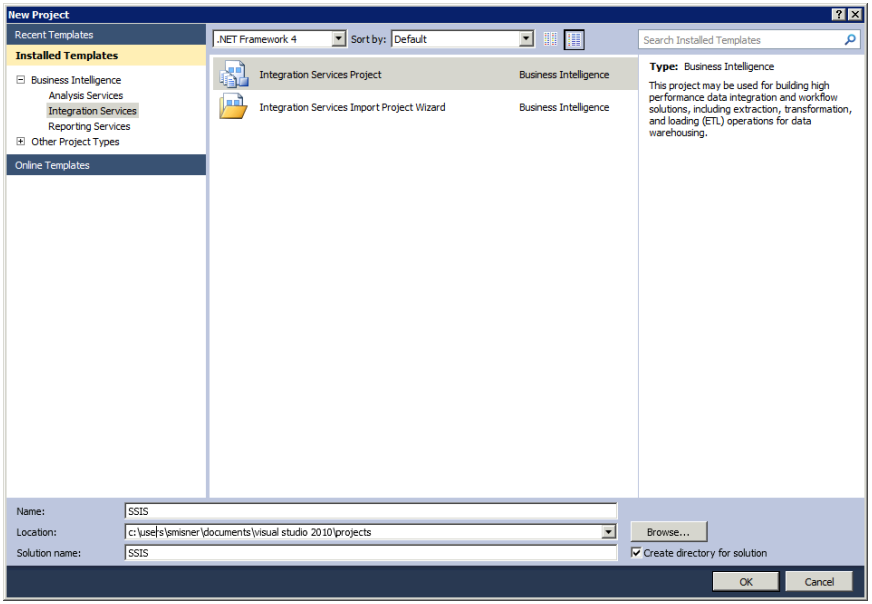


FIGURE 6-1 Add New Project dialog box displays installed templates.

There are two templates available for Integration Services projects:

- **Integration Services Project** You use this template to start development with a blank package to which you add tasks and arrange those tasks into workflows. This template type was available in previous versions of Integration Services.
- **Integration Services Import Project Wizard** You use this wizard to import a project from the Integration Services catalog or from a project deployment file. (You learn more about project deployment files in the "Deployment Models" section of this chapter.) This option is useful when you want to use an existing project as a starting point for a new project, or when you need to make changes to an existing project.

Note The Integration Services Connections Project template from previous versions is no longer available.

General Interface Changes

After creating a new package, there are several changes visible in the package designer interface, as you can see in Figure 6-2:

- **SSIS Toolbox** You now work with the SSIS Toolbox to add tasks and data flow components to a package, rather than the Visual Studio toolbox that you use in earlier versions of Integration Services. You learn more about this new toolbox in the "SSIS Toolbox" section of this chapter.

- **Parameters** The package designer includes a new tab to open the Parameters window for a package. Parameters allow you to specify run-time values for package, container, and task properties or for variables, as you learn in the "Parameters" section of this chapter.
- **Variables button** This new button on the package designer toolbar provides quick access to the Variables window. You can also continue to open the window from the SSIS menu or by right-clicking the package designer and selecting the Variables command.
- **SSIS Toolbox button** This button is also new in the package designer interface and allows you to open the SSIS Toolbox when it is not visible. As an alternative, you can open the SSIS Toolbox from the SSIS menu or by right-clicking the package designer and selecting the SSIS Toolbox command.
- **Getting Started** This new window displays below the Solution Explorer window and provides access to links to videos and samples that you can use to learn how to work with Integration Services. This window includes the Always Show In New Project checkbox, which you can clear if you prefer not to view the window after creating a new project. You learn more about using this window in the next section, "Getting Started Window.".
- **Zoom control** Both the control flow and data flow design surface now include a zoom control in the lower-right corner of the workspace. You can zoom in or out to a maximum size of 500 percent of the normal view or to a minimum size of 10 percent, respectively. As part of the zoom control, a button allows you to resize the view of the design surface to fit the window.

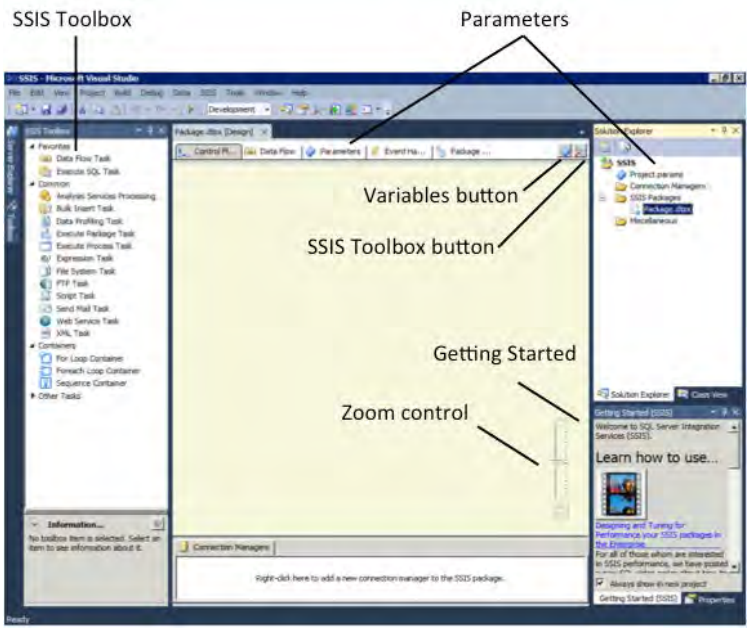


FIGURE 6-2 Package Designer Interface changes

Getting Started Window

As explained in the previous section, the Getting Started window is new to the latest version of Integration Services. Its purpose is to provide resources to new developers. It will display automatically when you create a new project unless you clear the checkbox at the bottom of the window. You must use the Close button in the upper-right corner of the window to remove it from view. Should you want to access the window later, you can choose Getting Started on the SSIS menu or right-click the design surface and select Getting Started.

In the Getting Started window, you find several links to videos and Integration Services samples. To use the links in this window, you must have Internet access. By default, the following topics are available:

- **Designing and Tuning for Performance your SSIS Packages in the Enterprise** This link provides access to a series of videos created by the SQL Server Customer Advisory Team (SQLCAT) that explain perform how to monitor package performance and techniques to apply during package development to improve performance.
- **Parameterizing the Execute SQL Task in SSIS** This link opens a page from which you can access a brief video explaining how to work with parameterized SQL statements in Integration Services.
- **SQL Server Integration Services Product Samples** You can use this link to access the product samples available on Codeplex, Microsoft's open source project hosting site. By studying the package samples available for download, you can learn how to work with various control flow tasks or data flow components.
- **Microsoft SQL Server Community Samples: Integration Services** Another collection of sample packages is available for download from Codeplex. Many of these samples demonstrate the development and use of custom components.

Note Although the videos and samples accessible through these links were developed for previous versions of Integration Services, the principles remain applicable to the latest version. When opening a sample project in BIDS, you will be prompted to convert the project.

You can customize the Getting Started window by adding your own links to the SampleSites.xml file located in the Program Files\Microsoft SQL Server\110\DTS\Binn folder.

SSIS Toolbox

Another new window for the package designer is the SSIS Toolbox. Not only has the overall interface been improved, but you will find there is also added functionality for arranging items in the toolbox.

Interface Improvement

The first thing you notice in the SSIS Toolbox is the updated icons for most items. Furthermore, the SSIS toolbox includes a description for the item that is currently selected, allowing you to see what it does without the need to add it first to the design surface. You can continue to use drag-and-drop to place items on the design surface, or to double-click the item. However, the new behavior when you double-click is to add the item to the container that is currently selected, which is a welcome timesaver for the development process. If no container is selected, the item is added directly to the design surface.

Item Arrangement

At the top of the SSIS Toolbox, you will see two new categories, Favorites and Common, as shown in Figure 6-3. All categories are populated with items by default, but you can move items into another category at any time. To do this, right-click the item and select Move To Favorites or Move To Common. If you are working with control flow items, you have Move To Other as another choice, but if you are working with data flow items, you can choose Move To Other Sources, Move To Other Transforms, or Move To Other Destinations. You will not see the option to move an item to the category in which it already exists, nor are you able to use drag-and-drop to move items manually. If you decide to start over and return the items to their original locations, select Restore Toolbox Defaults.

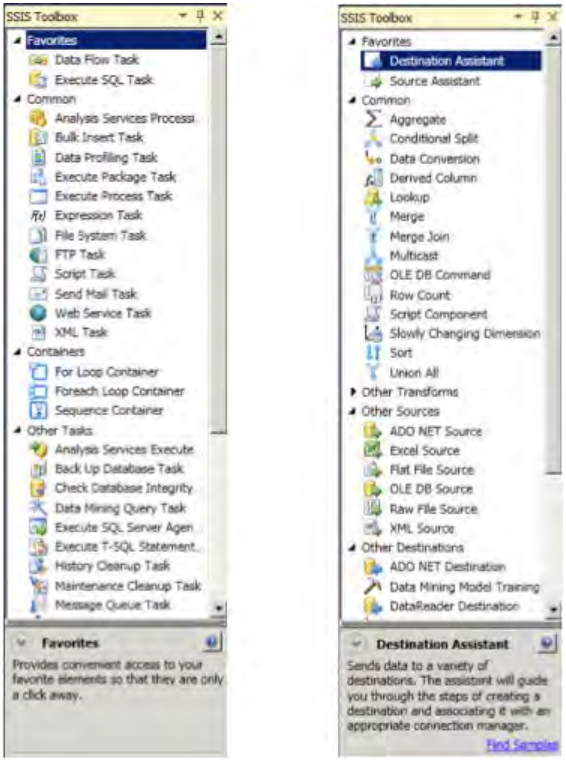


FIGURE 6-3 SSIS Toolbox for control flow and data flow.

Shared Connection Managers

If you look carefully at the Solution Explorer window, you will notice that the Data Sources and Data Source Views folders are missing, and have been replaced by a new file and a new folder. The new file is Project.params, which is used for package parameters and discussed in the "Package Parameters" section of this chapter. The Connections Manager folder is the new container for connection managers that you want to share among multiple packages.

To create a shared connection manager, follow these steps:

1. Right-click the Connections Manager folder and select New Connection Manager.
2. In the Add SSIS Connection Manager dialog box, select the desired connection manager type, and then click the Add button.
3. Supply the required information in the editor for the selected connection manager type, and then click OK until all dialog boxes are closed.

A file with the CONMGR file extension displays in the Solution Explorer window within the Connections Manager folder. In addition, the file also appears in the Connections Manager tray in the package designer in each package contained in the same project. It displays with a bold font to differentiate it from package connections. If you select the connection manager associated with one package and change its properties, the change affects the connection manager in all other packages.

If you change your mind about using a shared connection manager, you can convert it to a package connection. To do this, right-click the connection manager in the Connection Manager tray, and select Convert To Package Connection. The conversion removes the CONMGR file from the Connections Manager folder in Solution Explorer and from all other packages. Only the package in which you execute the conversion contains the connection.

Expression Indicators

The use of expressions in Integration Services allows you as a developer to create a flexible package. Behavior can change at run-time based on the current evaluation of the expression. For example, a very common reason to use expressions with a connection manager is to dynamically change connection strings to accommodate the movement of a package from one environment to another, such as from development to production. However, earlier versions of Integration Services did not provide an easy way to determine whether a connection manager relies on an expression. In the latest version, an extra icon appears above the connection manager icon as a visual cue that the connection manager uses expressions, as you can see in Figure 6-4.

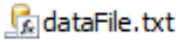


FIGURE 6-4 A visual cue that the connection manager uses an expression.

This type of expression indicator also appears with other package objects. If you add an expression to a variable or a task, the expression indicator will appear on that object.

Undo and Redo

A minor feature, but one that you will likely appreciate greatly, is the newly added ability to use Undo and Redo while developing packages in BIDS. You can now make edits in either the control flow or data flow designer surface, and use Undo to reverse a change or Redo to restore a change that you had just reversed. This capability also works in the Variables window, and on the Event Handlers and Parameters tabs. You can also use Undo and Redo when working with project parameters.

To use Undo and Redo, click the respective buttons in the standard toolbar. You can also use the Ctrl+Z and Ctrl+Y, respectively. Yet another option is to access these commands on the Edit menu.

> **Note** The Undo and Redo actions will not work with changes that you make to the SSIS Toolbox, nor will they work with shared connection managers.

Package Sort By Name

As you add multiple packages to a project, you might find it useful to see the list of packages in Solution Explorer display in alphabetical order. In previous versions of Integration Services, the only way to resort the packages was to close the project and then reopen it. Now you can easily sort the list of packages without closing the project by right-clicking the SSIS Packages folder, and selecting Sort By Name.

Status Indicators

After executing a package, the status of each item in the control flow and the data flow displays in the package designer. In previous versions of Integration Services, the entire item was filled with green to indicate success or red to indicate failure. However, for people who are color-blind, this use of color was not helpful for assessing the outcome of package execution. Consequently, the user interface now displays icons in the upper-right corner of each item to indicate success or failure, as shown in Figure 6-5.

FIGURE 6-5 Item status indicators appear in the upper-right corner.

Control Flow

Apart from the general enhancements to the package designer interface, there are two notable updates for the control flow. The Expression Task is a new item available to easily evaluate an expression during the package workflow. In addition, the Execute Package task has some changes to make it easier to configure the relationship between a parent package and child package.

Expression Task

Many of the developer experience enhancements in Integration Services affect both control flow and data flow, but there is one new feature that is exclusive to control flow. The Expression Task is new item available in the SSIS Toolbox when the control flow tab is in focus. The purpose of this task is to make it easier to assign a dynamic value to a variable.

Rather than use a Script Task to construct a variable value at run-time, you can now add an Expression Task to the workflow and use the SQL Server Integration Services Expression Language. When you edit the task, the Expression Builder opens. You start by referencing the variable and including the equals sign (=) as an assignment operator. Then provide a valid expression that resolves to a single value with the correct data type for the selected variable. Figure 6-6 illustrates an example of a variable assignment in an Expression Task.

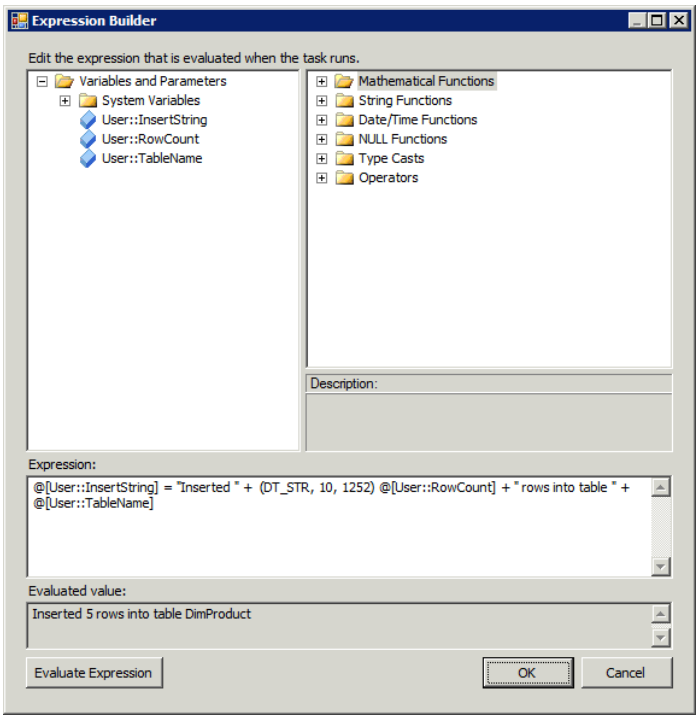


FIGURE 6-6 Variable assignment in an Expression Task.

Note The Expression Builder is an interface commonly used with other tasks and data flow components. Notice in Figure 6-6 that the list on the left side of the dialog box includes both variables and parameters. In addition, system variables are now accessible from a separate folder rather than listed together with user variables.

Execute Package Task

The Execute Package task has been updated to include a new property, ReferenceType, which you use to specify the location of the package to execute. If you select External Reference, you configure the path to the child package just as you do in earlier versions of Integration Services. If you instead select Project Reference, you then choose the child package from the drop-down list.

In addition, the Execute Package Task Editor has a new page for parameter bindings, as shown in Figure 6-7. You use this page to map a parameter from the child package to a parameter value or variable value in the parent package.

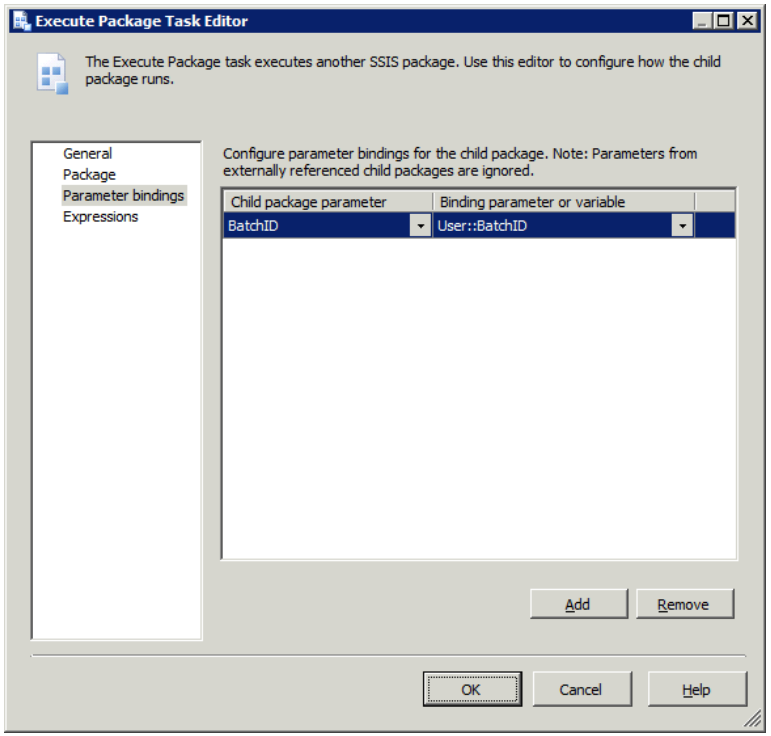


FIGURE 6-7 Parameter bindings between a parent package and a child package.

Data Flow

The data flow also has some significant updates. There are some new items, such as the Source and Destination Assistants and the DQS Cleansing transformation, and there are some improved items such as the Merge and Merge Join transformation. Some user interface changes have also been made to simplify the process and help you get your job done faster when designing the data flow.

Sources and Destinations

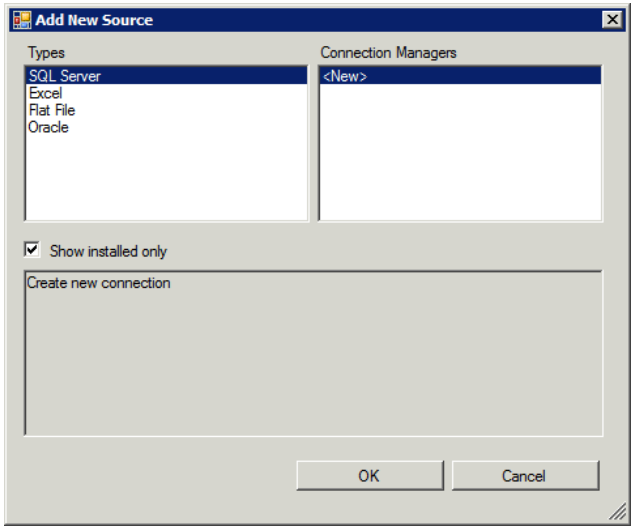
As we explore the changes in the data flow, let's start with sources and destinations.

Source and Destination Assistants

The Source Assistant and Destination Assistant are two new items available by default in the Favorites folder of the SSIS Toolbox when working with the data flow designer. These assistants help you easily create a source or a destination and its corresponding connection manager.

To create a SQL Server source in a data flow task, perform the following steps:

1. Add the Source Assistant to the data flow design surface by using drag-and-drop or by double-clicking the item in the SSIS Toolbox, which opens the Add New Source dialog box as shown here.



Note Clear the Show Installed Only checkbox to display the additional available source types that require installation of a client provider: DB2, SAP BI, Sybase, and Teradata.

2. In the Connection Managers list, select an existing connection manager or select <New> to create a new connection manager, and click OK.
3. If you selected the option to create a new connection manager, specify the server name, authentication method, and database for your source data in the Connection Manager dialog box, and click OK.

The new data source appears on the data flow design surface and the connection manager appears in the Connection Managers tray. You next need to edit the data source to configure the data access mode, columns, and error output.

Flat File Source

You use the Flat File source extract data from a CSV or TXT file, but there were some data formats that this source did not previously support without requiring additional steps in the extraction process. For example, you could not easily use the Flat File source with a file containing a variable number of columns. Another problem was the inability to use a character that has been designated as a qualifier as a literal value inside a string. The current version of Integration Services addresses both of these problems.

- **Variable columns** A file layout with a variable number of columns is also known as a *ragged right delimited file*. Although Integration Services supports a ragged right format, a problem arises when one or more of the rightmost columns do not have values and the column delimiters for the empty columns are omitted from the file. This situation commonly occurs when the flat file contains data of mixed granularity, such as header and detail transaction records. Although a row delimiter exists on each row, Integration Services ignored the row delimiter and included data from the next row until it processed data for each expected column. Now the Flat File source correctly recognizes the row delimiter and handles the missing columns as NULL values.

  > **Note** If you expect data in a ragged right format to include a column delimiter for each missing column, you can disable the new processing behavior by changing the AlwaysCheckForRowDelimiters property of the Flat File connection manager to False.

- **Embedded qualifiers** Another challenge with the Flat File source in previous versions of Integration Services was the use of a qualifier character inside a string encapsulated within qualifiers. For example, consider a flat file that contains the names of businesses. If a single quote is used as a text qualifier but also appears within the string as a literal value, the common practice is to use another single quote as an escape character, as shown here.

```
ID,BusinessName
404,'Margie''s Travel'
406, 'Kickstand Sellers'
```

  In the first data row in this example, previous versions of Integration Services would fail to interpret the second apostrophe in the BusinessName string as an escape character, but instead would process it as the closing text qualifier for the column. As a result, processing of the flat file returned an error because the next character in the row is not a column delimiter. This problem is now resolved in the current version of Integration Services with no additional configuration required for the Flat File source.

Transformations

Next we turn our attention to transformations.

Merge and Merge Join Transformations

Both the Merge transformation and the Merge Join transformation allow you to collect data from two inputs and produce a single output of combined results. In earlier versions of Integration Services, these transformations could result in excessive memory consumption by Integration Services when data arrives from each input at different rates of speed. The current version of Integration Services better accommodates this situation by introducing a mechanism for these two transformations to better manage memory pressure in this situation. This memory management mechanism operates automatically with no additional configuration of the transformation necessary.

> **Note** If you develop custom data flow components for use in the data flow and if these components accept multiple inputs, you can use new methods in the Microsoft.SqlServer.Dts.Pipeline namespace to provide similar memory pressure management to your custom components. You can learn more about implementing these methods at "Developing Data Flow Components with Multiple Inputs," located at *http://msdn.microsoft.com/en-us/library/ff877983(v=sql.110).aspx*.

DQS Cleansing Transformation

The DQS Cleansing transformation is a new data flow component that you use in conjunction with Data Quality Services (DQS). Its purpose is to help you improve the quality of data by using rules that are established for the applicable knowledge domain. You can create rules to test data for common misspellings in a text field or to ensure that the column length conforms to a standard specification.

To configure the transformation, you select a data quality field schema that contains the rules to apply and then select the input columns in the data flow to evaluate. In addition, you configure error handling. However, before you can use the DQS Cleansing transformation, you must first install and configure DQS on a server and create a knowledge base that stores information used to detect data anomalies and to correct invalid data, which deserves a dedicated chapter. We explain not only how DQS works and how to get started with DQS, but also how to use the DQS Cleansing transformation in Chapter 7, "Data Quality Services."

Column References

The pipeline architecture of the data flow requires precise mapping between input columns and output columns of each data flow component that is part of a Data Flow Task. The typical workflow during data flow development is to begin with one or more sources, and then proceed with the addition of new components in succession until the pipeline is complete. As you plug each subsequent component into the pipeline, the package designer configures the new component's input columns to match the data type properties and other properties of the associated output columns from the preceding component. This collection of columns and related property data is also known as *metadata*.

If you later break the path between components to add another transformation to pipeline, the metadata in some parts of the pipeline could change because the added component can add columns, remove columns, or change column properties (such as convert a data type). In previous versions of Integration Services, an error would display in the data flow designer whenever metadata became invalid. On opening a downstream component, the Restore Invalid Column References Editor displayed to help you correct the column mapping, but the steps to perform in this editor were not always intuitive. In addition, because of each data flow component's dependency on access to metadata, it was often not possible to edit the component without first attaching it to an existing component in the pipeline.

Components without Column References

Now Integration Services makes it easier to work with disconnected components. If you attempt to edit a transformation or destination that is not connected to a preceding component, a warning message box displays: "This component has no available input columns. Do you want to continue editing the available properties of this component?"

After you click Yes, the component's editor displays and you can configure the component as needed. However, the lack of input columns means that you will not be able to fully configure the component using the basic editor. If the component has an advanced editor, you can manually add input columns and then complete the component configuration. However, it is usually easier to use the interface to establish the metadata than to create it manually.

Resolve References Editor

The current version of Integration Services also makes it easier to manage the pipeline metadata if you need to add or remove components to an existing data flow. The data flow designer will display an error indicator next to any path that contains unmapped columns. If you right-click the path between components, you can select Resolve References to open a new editor that allows you to map the output columns to input columns by using a graphical interface, as shown in Figure 6-8.

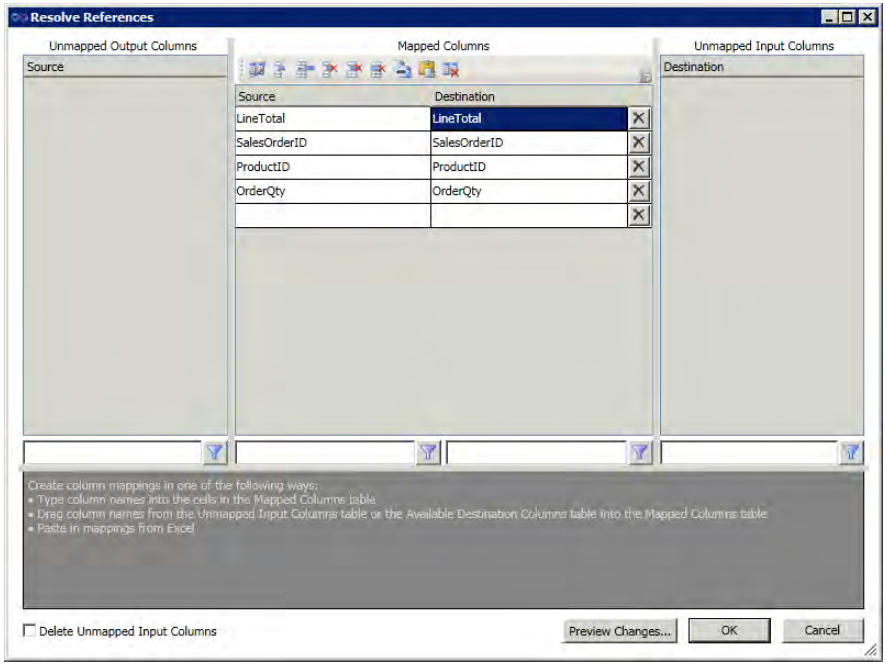


FIGURE 6-8 Resolve References editor for mapping output to input columns.

In the Resolve References editor, you can drag a column from the Unmapped Output Columns list and add it to the Source list in the Mapped Columns area. Similarly, you can drag a column from the Unmapped Input Columns to the Destination list to link together the output and input columns. Another option is to simply type or paste in the names of the columns to map.

Tip When you have a long list of columns in any of the four groups in the editor, you can type a string in the filter box below the list to view only those columns matching the criteria that you specify. For example, if your input columns are based on data extracted from the Sales.SalesOrderDetail table in the AdventureWork2008R2 database, you can type **unit** in the filter box to view only the UnitPrice and UnitPriceDiscount columns.

You can also manually delete a mapping by clicking the Delete Row button to the right of each mapping. After you have completed the mapping process, you can quickly delete any remaining unmapped input columns by selecting the Delete Unmapped Input Columns checkbox at the bottom of the editor. By eliminating unmapped input columns, you reduce the component's memory requirements during package execution.

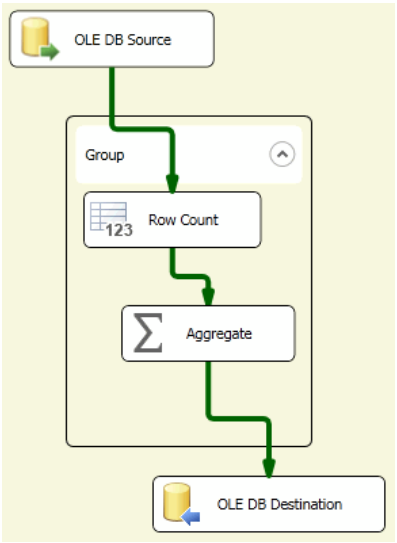
Collapsible Grouping

Sometimes the data flow contains too many components to see at one time in the package designer, depending on your screen size and resolution. Now you can consolidate data flow components into groups and expand or collapse the groups. A group in the data flow is similar in concept to a sequence

container in the control flow, although you cannot use the group to configure a common property for all components that it contains nor can you use it to set boundaries for a transaction or to set scope for a variable.

To create a group, follow these steps:

1. On the data flow design surface, use your mouse to draw a box around the components that you want to combine as a group. If you prefer, you can click each component while pressing the Ctrl key.
2. Right-click one of the selected components, and select Group. A group containing the components displays in the package designer, as shown below.



3. Click the arrow to the right of the group label to collapse the group.

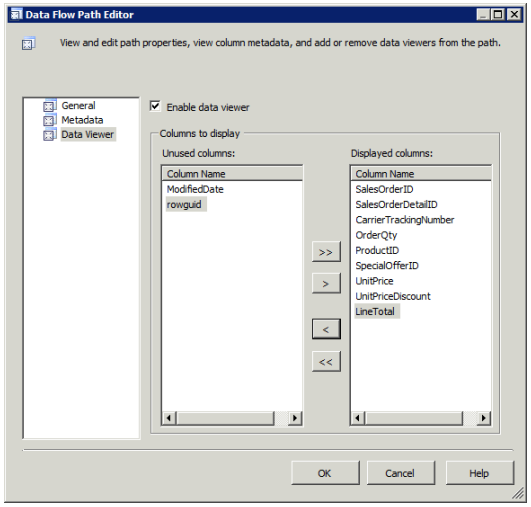
Data Viewer

The only data viewer option available now in Integration Services is the grid view. The histogram, scatter plot, and chart views have been removed.

To use the data viewer, follow these steps:

1. Right-click the path and select Enable Data Viewer. All columns in the pipeline are automatically included.
2. If instead you want to display a subset of columns, right-click the new Data Viewer icon (a magnifying glass) on the data flow design surface, and select Edit.
3. In the Data Flow Path Editor, select Data Viewer in the list on the left.
4. Move columns from the Displayed Columns list to the Unused Columns list as applicable

(shown below), and click OK.



Flexible Package Design

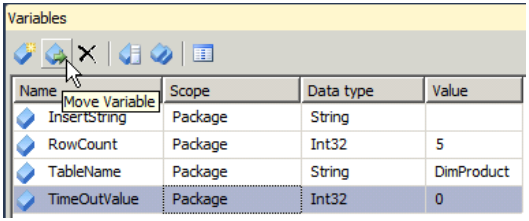
During the initial development stages of a package, you might find it easiest to work with hard-coded values in properties and expressions to ensure that your logic is correct. However, for maximum flexibility, you should use variables. In this section, we review the enhancements for variables and expressions—the cornerstones of flexible package design.

Variables

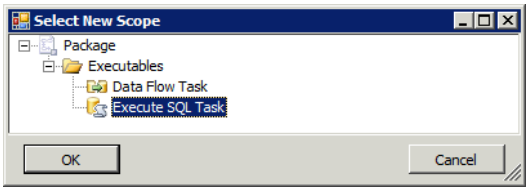
A common problem for developers when adding a variable to a package has been the scope assignment. If you had inadvertently selected a task in the control flow designer and then added a new variable in the Variables window, the variable was created within the scope of that task and could not be changed. You were required to delete the variable, clear the task selection on the design surface, and then add the variable again within the scope of the package.

Integration Services now creates new variables with scope set to the package by default. To change the variable scope, follow these steps:

1. In the Variables window, select the variable to change, then click the Move Variable button in the Variables toolbar (the second button from the left), as shown here.



2. In the Select New Scope dialog box, select the executable to have scope—the package, an event handler, container, or task—as shown here, and click OK.



Expressions

The expression enhancements in this release address a problem with expression size limitations and introduce new functions in the SQL Server Integration Services Expression Language.

Expression Result Length

Prior to the current version of Integration Services, if an expression result had a data type of DT_WSTR or DT_STR, any characters above a 4000-character limit would be truncated. Furthermore, if an expression contained an intermediate step that evaluated a result exceeding this 4000-character limit, the intermediate result would similarly be truncated. This limitation is now removed.

New Functions

The SQL Server Integration Services Expression Language now has three new functions that are useful for string manipulation:

- **LEFT** You can now more easily return the leftmost portion of a string rather than use the SUBSTRING function.

```
LEFT(character_expression,number)
```

- **TOKEN** This function allows you to return a substring by using delimiters to separate a string into tokens and then specifying which occurrence to return.

```
TOKEN(character_expression, delimiter_string, occurrence)
```

- **TOKENCOUNT** This function uses delimiters to separate a string into tokens and then returns the count of tokens found within the string.

```
TOKENCOUNT(character_expression, delimiter_string)
```

Deployment Models

Up to now in this chapter, we have explored the changes to the package development process in BIDS, which have been substantial. Another major change to Integration Services is the concept of deployment models.

Supported Deployment Models

The latest version of Integration Services supports two deployment models:

- **Legacy deployment model** The legacy deployment model, as its name suggests, is the deployment model used in previous versions of Integration Services in which the unit of deployment is an individual package stored as a DTSX file. A package can be deployed to the file system or to the MSDB database in a SQL Server database instance. Although packages can be deployed as a group and dependencies can exist between packages, there is no unifying object in Integration Services that identifies a set of related packages deployed using the legacy model. To modify properties of package tasks at run-time, which is important when running a package in different environments such as development or production, you use configurations saved as DTSCONFIG files on the file system. You use either the DTExec or the DTExecUI utilities to execute a package on the Integration Services server, providing arguments on the command-line or in the graphical interface when you want to override package property values at run-time manually or by using configurations.
- **Project deployment model** With this deployment model type, the unit of deployment is a project, stored as an ISPAC file, which in turn is a collection of packages and parameters. You deploy the project to the Integration Services Catalog, which we describe in a separate section of this chapter. Instead of configurations, you use parameters (as described later in the "Parameters" section) to assign values to package properties at run-time. Before executing a package, you must create an execution object in the catalog and optionally assign parameter values or environment references to the execution object. When ready, you start the execution object by using a graphical interface in SQL Server Management Studio by executing a stored procedure or by running managed code.

In addition to the characteristics described above, there are additional differences between the legacy deployment model and the project deployment model. Table 6-1 compares these differences.

TABLE 6-1 Deployment Model Comparison

Characteristic	Legacy Deployment Model	Project Deployment Model
Unit of deployment	Package	Project
Deployment location	File system or MSDB database	Integration Services catalog
Run-time property value assignment	Configurations	Parameters
Environment-specific values for use in property values	Configurations	Environment variables
Package validation	Just before execution using: DTExec Managed code	Independent of execution using: SQL Server Management Studio interface Stored procedure Managed code
Package execution	DTExec DTExecUI	SQL Server Management Studio interface Stored procedure Managed code
Logging	Configure log provider or implement custom logging	No configuration required
Scheduling	SQL Server Agent job	SQL Server Agent job
CLR integration	Not required	Required

When you create a new project in BIDS, the project is by default established as a project deployment model. You can use the Convert To Legacy Deployment Model command on the Project menu (or from the context menu when you right-click the project in Solution Explorer) to switch to the legacy deployment model. The conversion works only if your project is compatible with the legacy deployment model. For example, it cannot use features that are exclusive to the project deployment model, such as parameters. After conversion, Solution Explore displays an additional label after the project name to indicate the project is now configured as a legacy deployment model, as shown in Figure 6-9. Notice all that the Parameters node is removed from the project while the Data Sources folder is added to the project.

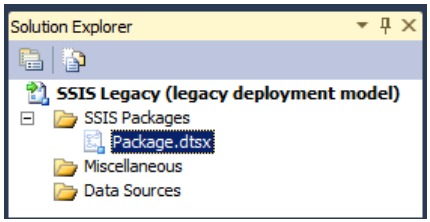


FIGURE 6-9 Legacy deployment model.

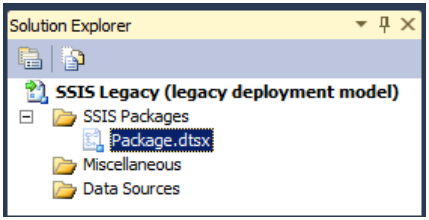


FIGURE 6-9 Legacy deployment model.

Tip You can reverse the process by using the Project menu, or the project's context menu in Solution Explorer, to convert a legacy deployment model project to a project deployment model.

Project Deployment Model Features

In this section, we provide an overview of the project deployment model features to help you understand how you use these features in combination to manage deployed projects. Later in this chapter, we explain each of these features in more detail and provide links to additional information available online.

Although you can continue to work with the legacy deployment model if you prefer, the primary advantage of using the new project deployment model is the improvement in package management across in multiple environments. For example, a package is commonly developed on one server, then tested on a separate server, and eventually implemented on a production server. With the legacy deployment model, there are a variety of techniques that you can use to provide connection strings for the correct environment at run-time, each of which requires you to create at least one configuration file and optionally maintain SQL Server tables or environment variables. Although this approach is flexible, it can also be confusing and prone to error. The project deployment model continues to separate run-time values from the packages, but uses object collections in the Integration Services catalog to store these values and to define relationships between packages and these object collections known as parameters, environments, and environment variables.

- **Catalog** The catalog is a dedicated database that stores packages and related configuration information accessed at package run-time. You can manage package configuration and execution by using the catalog's stored procedures and views or by using the graphical interface in SQL Server Management Studio.
- **Parameters** As Table 6-1 shows, the project deployment model relies on parameters to change task properties during package execution. Parameters can be created within a project scope or within a package scope. When you create parameters within a project scope, you use apply a common set of parameter values across all the packages contained in the project. You can then use parameters in expressions or tasks, much the same way that you use variables.
- **Environments** Each environment is a container of variables that you associate with a package at run-time. You can create multiple environments to use with a single package, but the package can only use variables from one environment during execution. For example, you can cre-

ate environments for development, test, and production, and then execute a package using one of the applicable environments.

- **Environment variables** An environment variable contains a literal value that Integration Services assigns to a parameter during package execution. After deploying a project, you can associate a parameter with an environment variable. The value of the environment variable resolves during package execution.

Project Deployment Workflow

The project deployment workflow includes not only the process of converting design-time objects in BIDS into database objects stored in the Integration Services catalog, but also the process of retrieving database objects from the catalog to update a package design or to use an existing package as a template for a new package. To add a project to the catalog or to retrieve a project from the catalog, you use a project deployment file which has an ISPAC file extension. There are four stages of the project deployment workflow in which the ISPAC file plays a role: build, deploy, import, and convert. In this section, we review each of these stages.

Build

When you use the project deployment model for packages, you use BIDS to develop one or more packages as part of an Integration Services project. In preparation for deployment to the catalog, which serves as a centralized repository for packages and related objects, you build the Integrations Services project in BIDS to produce an ISPAC file, as shown in Figure 6-10. The ISPAC file is the project deployment file that contains project information, all packages in the Integration Services project, and parameters.

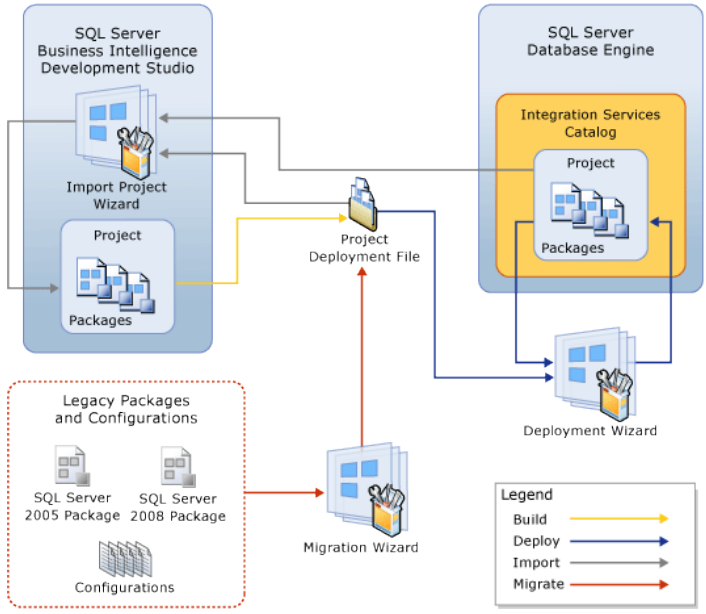


FIGURE 6-10 Project deployment workflow.

Before performing the build, there are two additional tasks that might be necessary:

- **Identify entry-point package** If one of the packages in the project is the package that triggers the execution of the other packages in the project, directly or indirectly, you should flag that package as entry-point package. You can do this by right-clicking the package in Solution Explorer and selecting Entry-point Package. An administrator uses this flag to identify the package to start when a package contains multiple projects.
- **Create project and package parameters** You use project-level or package-level parameters to provide values for use in tasks or expressions at run-time, which you learn more about how to do later in this chapter in the "Parameters" section. In BIDS, you assign parameter values to use as a default. You also mark a parameter as required, which prevents a package from executing until you assign a value to the variable.

During the development process in BIDS, you commonly execute a task or an entire package within BIDS to test results before deploying the project. BIDS creates an ISPAC file to hold the information required to execute the package and stores it in the bin folder for the Integration Services project. When you finish development and want to prepare the ISPAC file for deployment, use the Build menu or press F5.

Deploy

The deployment process uses the ISPAC file to create database objects in the catalog for the project, packages, and parameters, as shown in Figure 6-11. To do this, you use the Integration Services Deployment Wizard which prompts you for the project to deploy and the project to create or update as

part of the deployment. You can also provide literal values or specify environment variables as default parameter values for the current project version. These parameter values that you provide in the wizard are stored in the catalog as server defaults for the project, and override the default parameter values stored in the package.

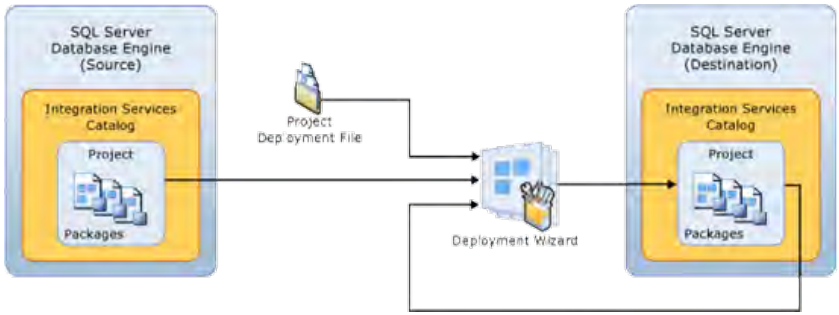


FIGURE 6-11 Deployment of the ISPAC file to the catalog.

You can launch the wizard from within BIDS by right-clicking the project in Solution Explorer, and selecting Deploy. However, if you have an ISPAC file saved to the file system, you can double-click the file to launch the wizard.

Import

When you want to update a package that has already been deployed or to use it as basis for a new package, you can import a project into BIDs from the catalog or from an ISPAC file, as shown in Figure 6-12. To import a project, you use the Integration Services Import Project Wizard which is available in the template list when you create a new project in BIDS.

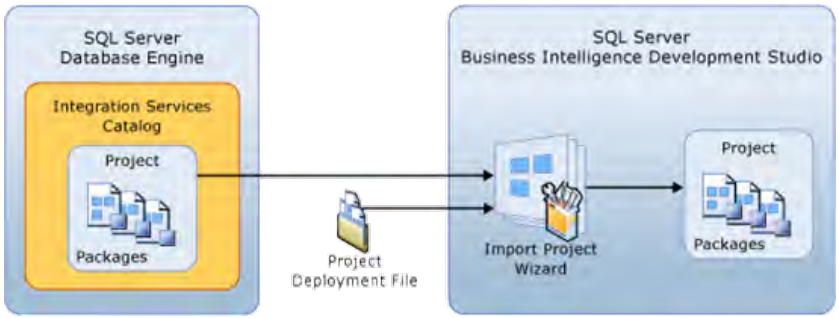


FIGURE 6-12 Import a project from the catalog or an ISPAC file.

Convert

If you have legacy packages and configuration files, you can convert them to the latest version of Integration Services, as shown in Figure 6-13. The Integration Services Project Conversion Wizard is available in both BIDS and in SQL Server Management Studio. Another option is to use the Integration Services Package Upgrade Wizard available on the Tools page of the SQL Server Installation Center.

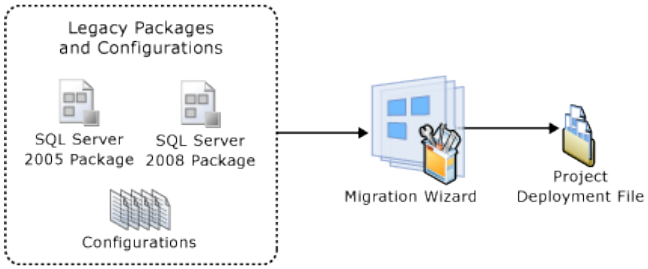


FIGURE 6-13 Convert existing DTSX files and configurations to an ISPAC file.

Note You can use the Conversion Wizard to migrate packages created using SQL Server 2005 Integration Services and later. If you use SQL Server Management Studio, the original DTSX files are not modified, but used only as a source to produce the ISPAC file containing the upgraded packages.

In BIDS, open a legacy project, right-click the project in Solution Explorer, and select Convert To Project Deployment Model. The wizard upgrades the DTPROJ file for the project and the DTSX files for the packages.

The behavior of the wizard is different in SQL Server Management Studio. There you right-click the Projects node of the Integration Services catalog in Object Explorer, and select Import Packages. The wizard prompts you for a destination location and produces an ISPAC file for the new project and the upgraded packages.

Regardless of which method you use to convert packages, there are some common steps that occur as packages are upgraded:

- **Update Execute Package tasks** If a legacy package contains an Execute Package task, the wizard changes the external reference to a DTSX file to a project reference to a package contained within the same project. The child package must be in the same legacy project that you are converting and must be selected for conversion in the wizard.
- **Create parameters** If a legacy package uses a configuration, you can choose to convert the configuration to parameters. You can add configurations belonging to other projects to include them in the conversion process. Additionally, you can choose to remove configurations from the upgraded packages. The wizard uses the configurations to prompt you for properties to convert to parameters and also requires you to specify project scope or package scope for each parameter.
- **Configure parameters** The Conversion Wizard allows you to specify a server value for each parameter and whether to require the parameter at run-time.

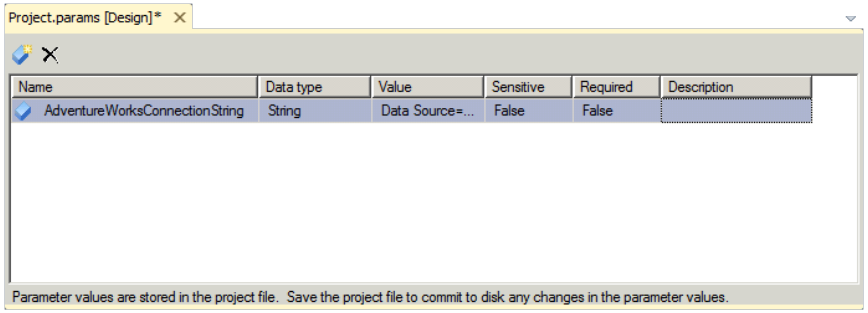
Parameters

As we explain in the previous section, parameters are the replacement for configurations in legacy packages, but only when you use the project deployment model. The purpose of configurations was to provide a way to change values in a package at run-time without requiring you to open the package and make the change directly. You can establish project-level parameters to assign a value to one or more properties across multiple packages, or you can have a package-level parameter when you need to assign a value to properties within a single package.

Project Parameters

A project parameter shares its values with all packages within the same project. To create a project parameter in BIDS, follow these steps:

1. In Solution Explorer, double-click Project.params.
2. Click the Add Parameter button on the toolbar in the Project.params window.
3. Type a name for the parameter in the Name text box, select a data type, and specify a value for the parameter as shown here. The parameter value that you supply here is known as the design default value.



> **Note** The parameter value is a design-time value that can be overwritten during or after deployment to the catalog.

4. Save the file.

Optionally, you can configure the following properties for each parameter:

- **Sensitive** By default, this property is to False. If you change it to True, the parameter value is encrypted when you deploy the project to the catalog. If anyone attempts to view the parameter value in SQL Server Management Studio or by accessing Transact-SQL views, the parameter value will display as NULL. This setting is important when you use a parameter to set a connection string property and the value contains specific credentials.

- **Required** By default, this property is also set to False. When the value is True, you must configure a parameter value during or after deployment before you can execute the package. The Integration Services engine will ignore the parameter default value that you specify on this screen when the Required property is True and deploy the package to the catalog.
- **Description** This property is optional, but allows you to provide documentation to an administrator responsible for managing packages deployed to the catalog.

Package Parameters

Package parameters apply only to the package in which they are created and cannot be shared with other packages. The center tab in the package designer allows you to access the Parameters window for your package. The interface for working with package parameters is identical to the project parameters interface.

Parameter Usage

After creating project or package parameters, you are ready to implement the parameters in your package much like you implement variables. That is, anywhere you can use variables in expressions for tasks, data flow components, or connection managers, you can also use parameters.

As one example, you can reference a parameter in expressions, as shown in Figure 6-14. Notice the parameter appears in the Variables and Parameters list in the top left pane of the Expression Builder. You can drag the parameter to the Expression text box and use it alone or as part of a more complex expression. When you click the Evaluate Expression button, you can see the expression result based on the design default value for the parameter.

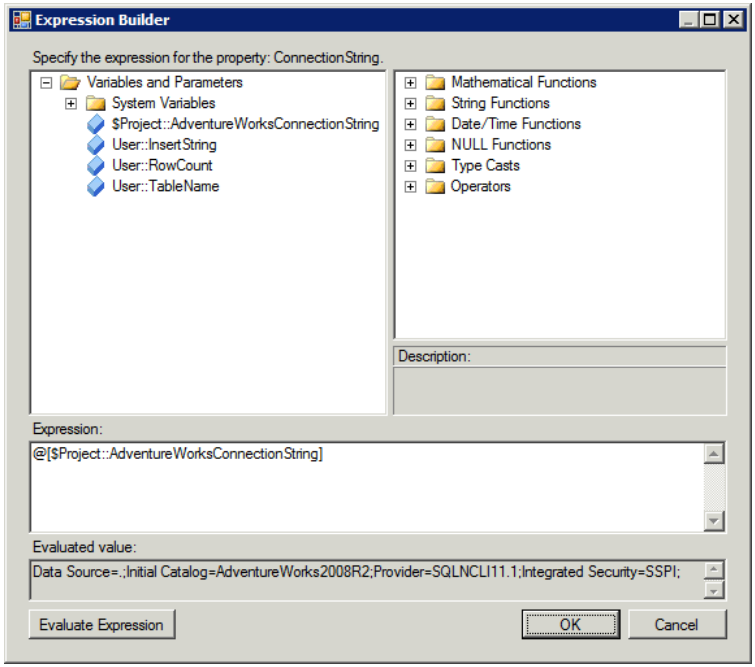


FIGURE 6-14 Parameter usage in an expression.

Note This expression uses a project parameter which has a prefix of $Project. To create an expression that uses a package parameter, the parameter prefix is $Package.

You can also directly set a task property by right-clicking the task and selecting Parameterize on the context menu. The Parameterize dialog box displays as shown in Figure 6-15. You select a property, and then choose whether to create a new parameter or use an existing parameter. If you create a new parameter, you specify values for each of the properties that you access in the Parameters window. Additionally, you must specify whether to create the parameter within package scope or project scope.

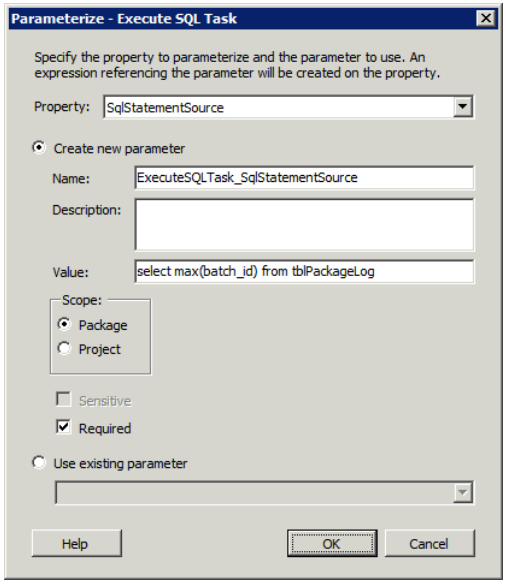


FIGURE 6-15 Parameterize task dialog box.

Post-Deployment Parameter Values

The design default values that you set for each parameter in BIDS are typically used only to supply a value for testing within the BIDS environment. You can replace these values during deployment by specifying server default values when you use the Deployment Wizard or by configuring execution values when creating an execution object for deployed projects.

Figure 6-16 illustrates the stage at which you create each type of parameter value. If a parameter has no execution value, the Integration Services engine will use the server default value when executing the package. Similarly, if there is no server default value, package execution uses the design default value. However, if a parameter is marked as required, then you must provide either a server default value or an execution value.

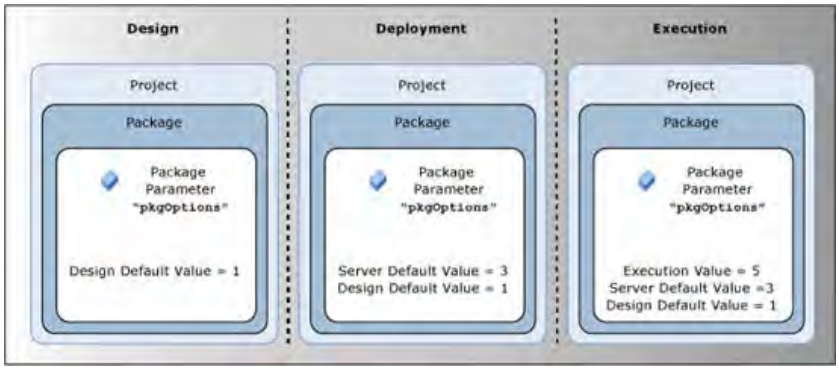


FIGURE 6-16 Parameter values by stage.

Note A package will fail when the Integration Services engine cannot resolve a parameter value. For this reason, it is recommended to validate projects and packages as described in the "Validation" section of this chapter.

Server Default Values

Server default values can be literal values or environment variable references (explained later in this chapter), which in turn are literal values. To configure server defaults in SQL Server Management Studio, you right-click the project or package in the Integration Services node in Object Explorer, select Configure, and change the Value property of the parameter, as shown in Figure 6-17. This server default value will persist even if you make changes to the design default value in BIDS and redeploy the project.

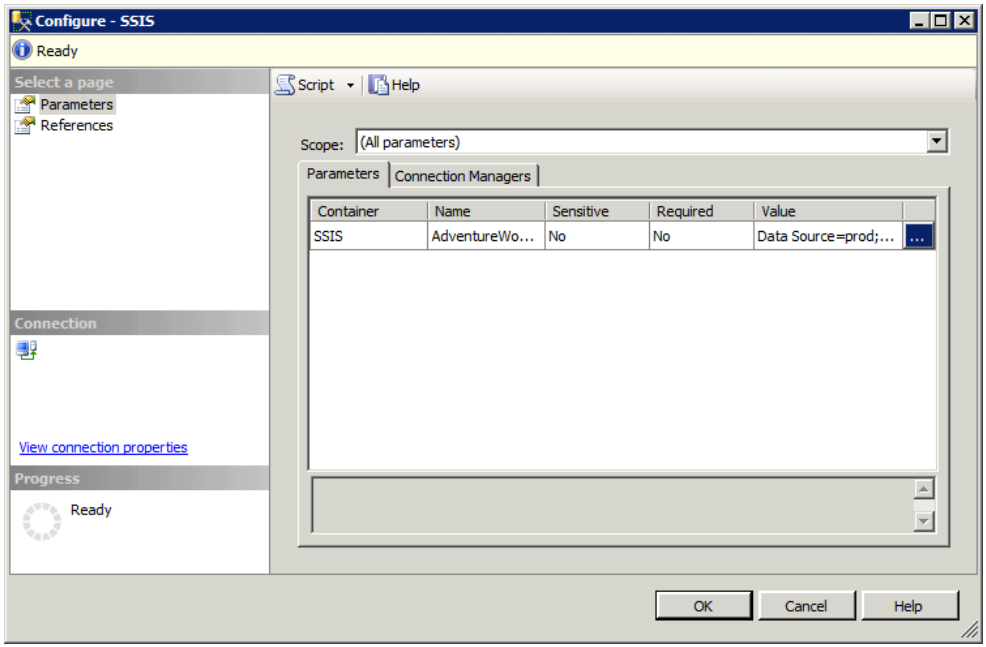


FIGURE 6-17 Server default value configuration.

Execution Parameter Values

The execution parameter value applies only to a specific execution of a package and overrides all other values. You must explicitly set the execution parameter value by using the catalog.set_execution_parameter_value stored procedure. There is no interface available in SQL Server Management Studio to set an execution parameter value.

```
set_execution_parameter_value [ @execution_id = execution_id
        , [ @object_type = ] object_type
        , [ @parameter_name = ] parameter_name
        , [ @parameter_value = ] parameter_value
```

To use this stored procedure, you must supply the following arguments:

- **execution_id** You must obtain the execution_id for the instance of the execution. You can use the catalog.executions view to locate the applicable execution_id.
- **object_type** The object type specifies whether you are setting a project parameter or a package parameter. Use a value of 20 for a project parameter and a value of 30 for a package parameter.
- **parameter_name** The name of the parameter must match the parameter stored in the catalog.
- **parameter_value** Here you provide the value to use as the execution parameter value.

Integration Services Catalog

The Integration Services catalog is a new feature to support the centralization of storage and administration of packages and related configuration information. Each SQL Server instance can host only one catalog. When you deploy a project using the project deployment model, the project and its components are added to the catalog and optionally placed in a folder that you specify in the Deployment Wizard. Each folder (or the root level if you choose not to use folders) organizes its contents into two groups—projects and environments, as shown in Figure 6-18.

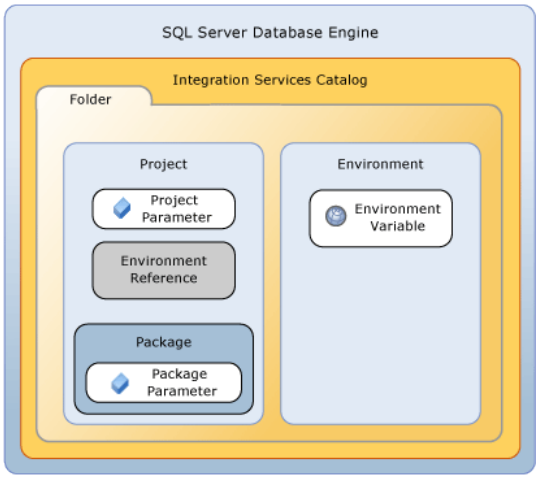
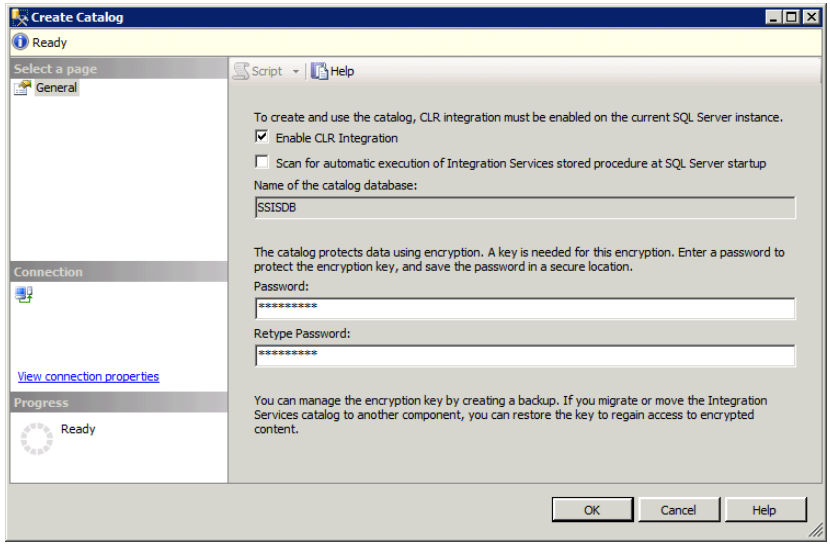


FIGURE 6-18 Catalog database objects.

Catalog Creation

Installation of Integration Services on a server does not automatically create the catalog. To do this, follow these steps:

1. In SQL Server Management Studio, connect to the SQL Server instance, right-click the Integration Services folder in Object Explorer, and select Create Catalog.
2. In the Create Catalog dialog box, select the Enable CLR Integration checkbox. This feature is required to manage Integration Services functionality.
3. Optionally, you can select the Scan For Automatic Execution Of Integration Services Stored Procedure At SQL Server Startup checkbox. This stored procedure performs a cleanup operation when the service restarts and adjusts the status of packages that were executing when the service stopped.
4. Notice that the catalog database name cannot be changed from SSISDB, as shown below so the final step is to provide a strong password, and then click OK. The password creates a database master key that Integration Services uses to encrypt sensitive data stored in the catalog.



After you create the catalog, you will see it appear twice as the SSISDB database in Object Explorer. It displays under both the Databases node as well as the Integration Services node. In the Databases node, you can interact with it as you would any other database, using the interface to explore database objects. You use the Integration Services node to perform administrative tasks.

> Note In most cases, there are two options available for performing administrative tasks with the catalog. You can use the graphical interface by opening the applicable dialog box for a selected catalog object, or you can use Transact-SQL views and stored procedures to view and modify object properties. For more information about the Tranact-SQL application programming interface (API), see *http://msdn.microsoft.com/en-us/library/ff878003(v=SQL.110).aspx*.

Catalog Properties

The catalog has several configurable properties. To access these properties, right-click SSISDB under the Integration Services node, and select Properties. The Catalog Properties dialog box, as shown in Figure 6-19, displays several properties.

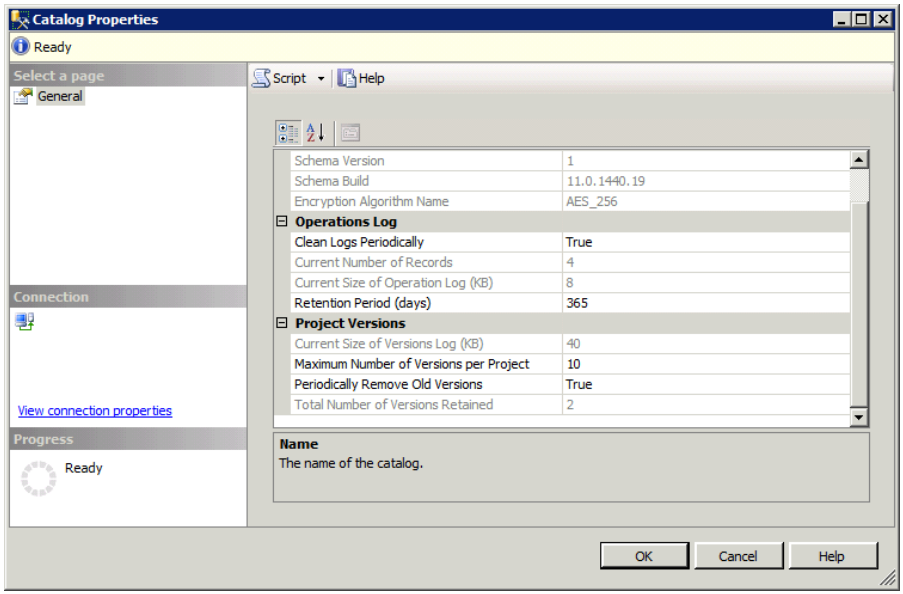


FIGURE 6-19 Catalog Properties dialog box.

Encryption

Notice in Figure 6-19 that the default encryption algorithm is AES_256. If you put the SSISDB database in single-user mode, you can choose one of the other encryption algorithms available:

- DES
- TRIPLE_DES
- TRIPLE_DES_3KEY
- DESX
- AES_128
- AES_192

Integration Services uses encryption to protect sensitive parameter values. When anyone uses the SQL Server Management Studio interface or the Transact-SQL API to query the catalog, the parameter value will display only a NULL value.

Operations

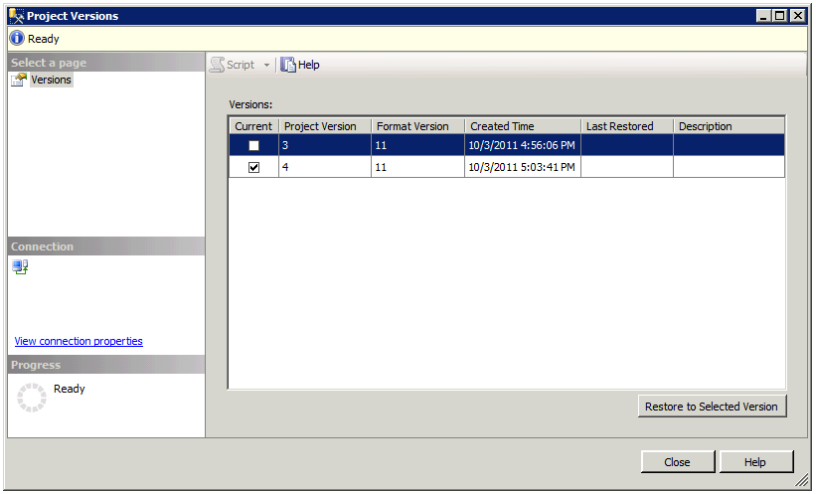
Operations include activities such as package execution, project deployment, and project validation, to name a few. Integration Services stores information about these operations in tables in the catalog. You can use the Transact-SQL API to monitor operations, or you can right-click the SSISDB database on the Integration Services node in Object Explorer and select Active Operations. The Active Operations dialog box displays the operation identifier, its type, name, the operation start time, and the caller of the operation. You can select an operation and click the Stop button to end the operation.

Periodically, older data should be purged from these tables to keep the catalog from growing unnecessarily large. By configuring the catalog properties, you can control the frequency of the SQL Server Agent job that purges the stale data by specifying how many days of data to retain. If you prefer, you can disable the job.

Project Versioning

Each time you redeploy a project with the same name to the same folder, the previous version remains in the catalog until ten versions are retained. If necessary, you can restore a previous version by following these steps:

1. In Object Explorer, locate the project under the SSISDB node.
2. Right-click the project and select Versions.
3. In the Project Versions dialog box, shown here, select the version to restore and click the Restore To Selected Version button.



4. Click Yes to confirm, click OK to close the information message box. Notice the selected version is now flagged as the current version, and that the other version remains available as an option for restoring.

You can modify the maximum number of versions to retain by updating the applicable catalog

property. If you increase this number above the default value of ten, you should continually monitor the size of the catalog database to ensure that it does not grow too large. To manage the size of the catalog, you can also decide whether to remove older versions periodically with a SQL Server agent job.

Environment Objects

After you deploy projects to the catalog, you can create environments to work in tandem with parameters to change parameter values at execution time. An environment is a collection of environment variables. Each environment variable contains a value to assign to a parameter. To connect an environment to a project, you use an environment reference. Figure 6-20 illustrates the relationship between parameters, environments, environment variables, and environment references.

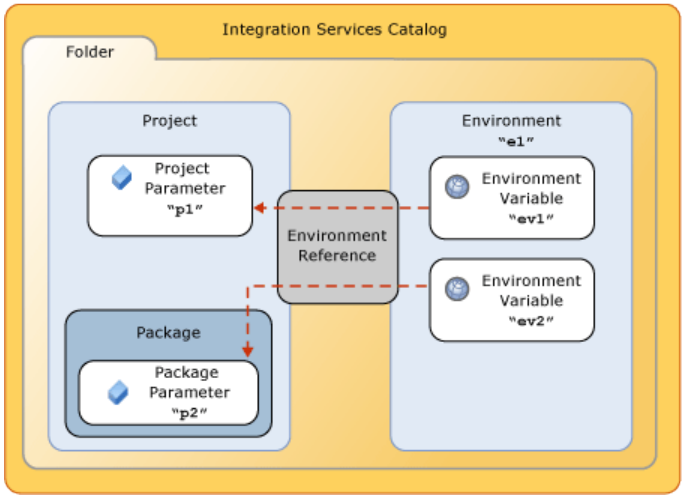


FIGURE 6-20 Environment objects in the catalog.

Environments

One convention that you might use is to create one environment for each server that you will use for package execution. For example, you might have one environment for development, one for testing, and one for production. To create a new environment using the SQL Server Management Studio interface, follow these steps:

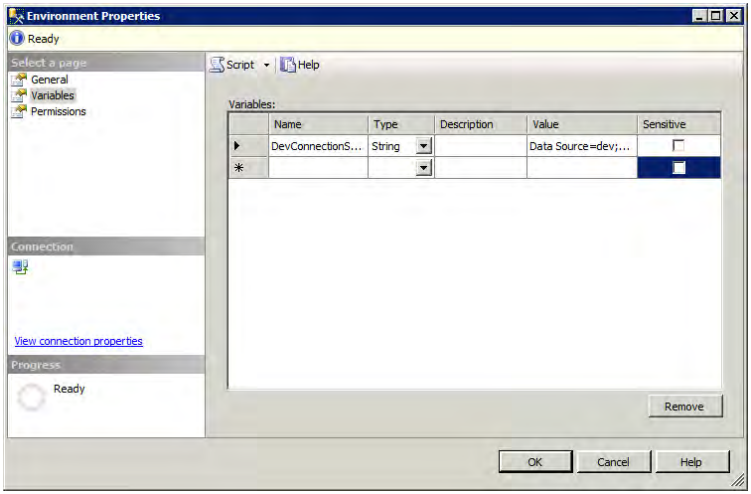
1. In Object Explorer, expand the SSISDB node, and locate the Environments folder that corresponds to the Projects folder containing your project.
2. Right-click the Environments folder and select Create Environment.
3. In the Create Environment dialog box, type a name, optionally type a description, and click OK.

Environment Variables

For each environment, you can create a collection of environment variables. The properties that you

configure for an environment variable are the same ones that you configure for a parameter, which is understandable when you consider that you use the environment variable to replace the parameter value at run-time. To create an environment variable, follow these steps:

1. In Object Explorer, locate the environment under the SSISDB node.
2. Right-click the environment and select Properties to open the Environment Properties dialog box.
3. Click Variables to display the list of existing environment variables, if any, as shown below.



4. On an empty row, type a name for the environment variable in the Name text box, select a data Type, type a Description (optional), type a Value for the environment variable, select the Sensitive checkbox if you want the value to be encrypted in the catalog. Continue adding environment variables on this page and click OK when finished.
5. Repeat this process by adding the same set of environment variables to other environments that you intend to use with the same project.

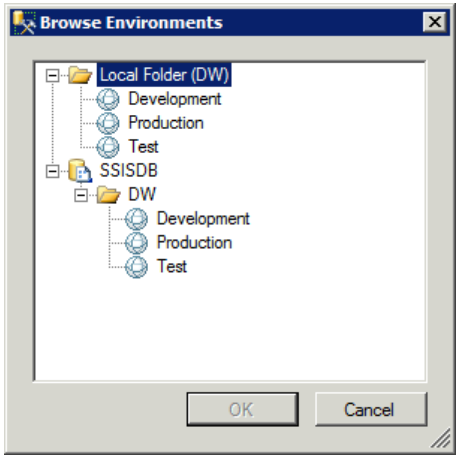
Environment References

To connect environment variable to a parameter, you create an environment reference. There are two types of environment references—relative and absolute. When you create a relative environment reference, the parent folder for the environment folder must also be the parent folder for the project folder. If you later move the package to another without also moving the environment, the package execution will fail. An alternative is to use an absolute reference which maintains the relationship between the environment and the project without requiring them to have the same parent folder.

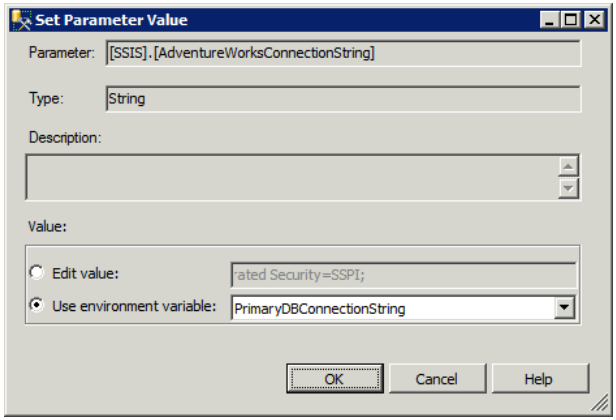
The environment reference is a property of the project. To create an environment reference, follow these steps:

1. In Object Explorer, locate the project under the SSISDB node.

2. Right-click the project and select Configure to open the Configure <Project> dialog box.
3. Click References to display the list of existing environment references, if any.
4. Click the Add button and select an environment in the Browse Environments dialog box. Use the Local Folder node for a relative environment reference or use the SSISDB node for an absolute environment reference.



5. Click OK twice to create the reference. Repeat steps 4 and 5 to add reference for all other applicable environments.
6. In the Configure <Project> dialog box, click Parameters to switch to the parameters page.
7. Click the ellipsis button to the right of the Value text box to display the Set Parameter Value dialog box, select the Use Environment Variable option, and select the applicable variable in the drop-down list, as shown below.



8. Click OK twice.

You can create multiple references for a project, but only one environment will be active during package execution. At that time, Integration Services will evaluate the environment variable based on the environment associated with the current execution instance as explained in the next section.

Administration

After the development and deployment processes are complete, it's time to become familiar with the administration tasks that enable operations on the server to keep running.

Validation

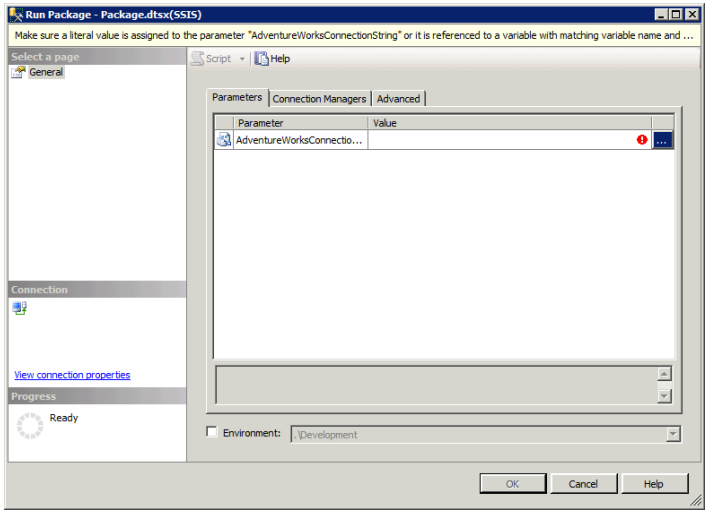
Before executing packages, you can use validation to verify that projects and packages are likely to run successfully, especially if you have configured parameters to use environment variables. The validation process ensures that server default values exist for required parameters, that environment references are valid, and that data types for parameters are consistent between project and package configurations and their corresponding environment variables, to name a few of the validation checks.

To perform the validation, right-click the project or package in the catalog, click Validate, and select the environments to include in the validation: all, none, or a specific environment. Validation occurs asynchronously, so the Validation dialog box closes while the validation processes. You can open the Integration Services Dashboard report to check the results of validation. Your other options are to right-click the SSISDB node in Object Explorer and select Active Operations or to use of the Transact-SQL API to monitor an executing package.

Package Execution

After deploying a project to the catalog and optionally configuring parameters and environment references, you are ready to prepare your packages for execution. This step requires you to create a SQL Server object called an execution. An *execution* is a unique combination of a package and its corresponding parameter values, whether the values are server defaults or environment references. To configure and start an execution instance, follow these steps:

1. In Object Explorer, locate the entry-point package under the SSISDB node.
2. Right-click the project and select Run to open the Run Package dialog box, shown below.



3. Here you have two choices. You can either click the ellipsis button to the right of the Value and specify a literal execution value for the parameter, or you can select the Environment checkbox at the bottom of the dialog box and select an environment in the corresponding drop-down list.

You can continue configuring the execution instance by updating properties on the Connections Manager tab and by overriding property values and configuring logging on the Advanced tab. For more information about the options available in this dialog box, see *http://msdn.microsoft.com/en-us/library/hh231080(v=SQL.110).aspx*.

When you click OK to close the Run Package dialog box, the package execution begins. Because package execution occurs asynchronously, the dialog box does not need to stay open during execution. You can use the Integration Services Dashboard report to monitor the execution status, or right-click the SSISDB node and select Active Operations. Another option is the use of the Transact-SQL API to monitor an executing package.

More often, you will schedule package execution by creating a Transact-SQL script that starts execution and save the script to a file that you can then schedule using a SQL Server agent job. You add a job step using the Operating System (CmdExec) step type and then configure the step to use the sqlcmd.exe utility and pass the package execution script to the utility as an argument. You run the job using the SQL Server Agent service account or a proxy account. Whichever account you use, it must have permissions to create and start executions.

Logging and Troubleshooting Tools

Now that Integration Services centralizes package storage and executions on the server and has access to information generated by operations, server-based logging is supported and operations reports are available in SQL Server Management Studio to help you monitor activity on the server and troubleshoot problems when they occur.

Package Execution Logs

In legacy Integration Services packages, there are two options you can use to obtain logs during package execution. One option is to configure log providers within each package and associate log providers with executables within the package. The other option is to use a combination of Execute SQL statements or script components to implement a custom logging solution. Either way, the steps necessary to enable logging are tedious in legacy packages.

With no configuration required, Integration Services stores package execution data in the [catalog].[excecutions] table. The most important columns in this table include the start and end times of package execution as well as the status. However, the logging mechanism also captures information related to the Integration Services environment such as physical memory, the page file size, and available CPUs. Other tables provide access to parameter values used during execution, the duration of each executable within a package, and messages generated during package execution. You can easily write ad hoc queries to explore package logs or build your own custom reports using Reporting Services for ongoing monitoring of the log files.

Note For a thorough walkthrough of the various tables in which package execution log data is stored, see "SSIS Logging in Denali," a blog post by Jamie Thomson at *http://sqlblog.com/blogs/jamie_thomson/archive/2011/07/16/ssis-logging-in-denali.aspx*.

Data Taps

A data tap is similar in concept to a data viewer, except that it captures data at a specified point in the pipeline during package execution outside of BIDS. The captured data is stored in a CSV file which you can review after package execution completes. No changes to your package are necessary to use this feature.

Reports

Before you build custom reports from the package execution log tables, review the built-in reports now available in SQL Server Management Studio for Integration Services. These reports provide information on package execution results for the past 24 hours (as shown in Figure 6-24), performance, and error messages from failed package executions. Hyperlinks in each report allow you to drill through from summary to detailed information to help you diagnose package execution problems.

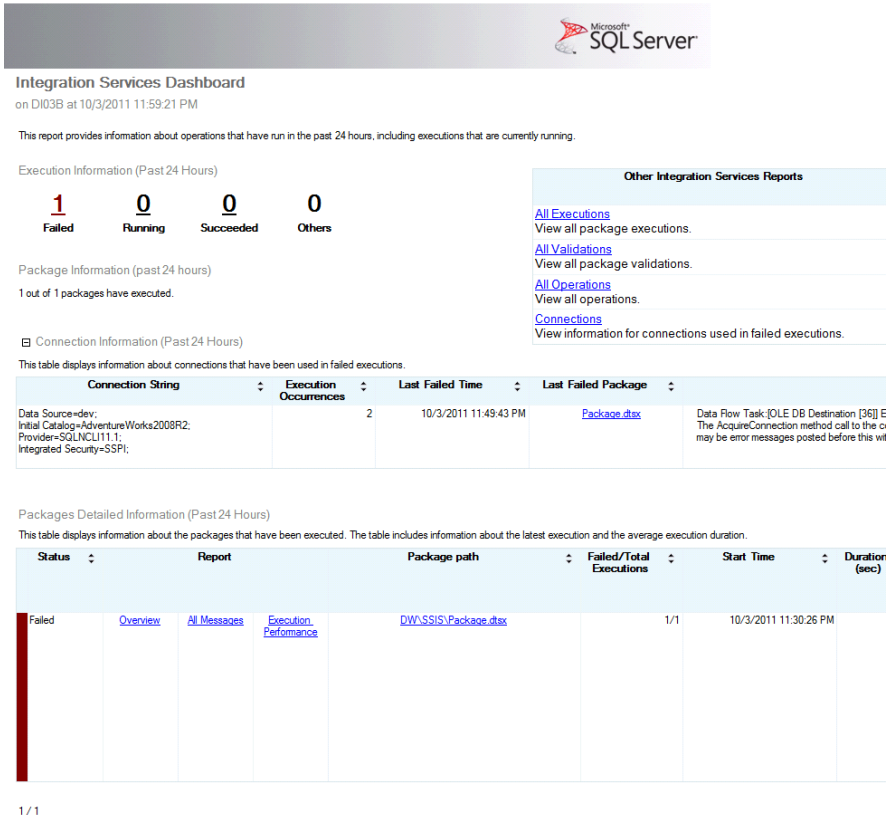


FIGURE 6-24 Integration Services Operations Dashboad.

To view the reports, you right-click the SSISDB node in Object explorer, point to Reports, point to Standard Reports, and then choose from the following list of reports:

- Integration Services Dashboard
- All Executions
- All Validations
- All Operations
- Connections

Security

Packages and related objects are stored securely in the catalog using encryption. Only members of the new SQL Server database role ssis_admin or members of the existing sysadmin role have permissions to all objects in the catalog. Members of these roles can perform operations such as creating the catalog, creating folders in the catalog, and executing stored procedures, to name a few.

Members of the administrative roles delegate administrative permissions to users who need to manage a specific folder. Delegation is useful when you do not want to give these users access to the higher privileged roles. To give a user folder-level access, you grant the MANAGE_OBJECT_PERMISSIONS permission to the user.

For general permissions management, open the Properties dialog box for a folder (or any other securable object) and go to the Permissions page. On that page, you can select a security principal by name and then set explicit Grant or Deny permissions as appropriate. You can use this method to secure folders, projects, environments, and operations.

Package File Format

Although legacy packages stored as DTSX files are formatted as XML, their structure is not compatible with differencing tools and source control systems that you might use to compare packages. In the current version of Integration Services, the package file format is pretty-printed, with properties formatted as attributes rather than as elements. Moreover, attributes are listed alphabetically and attributes configured with default values have been eliminated. Collectively, these changes not only help you more easily locate information in the file, but you can more easily compare packages with automated tools and more reliably merge packages that have no conflicting changes.

Another significant change to the package file format is the replacement of the meaningless numeric lineage identifiers with a refid attribute with a text value that represents the path to the referenced object. For example, a refid for the first input column of an Aggregate transformation in a data flow task called Data Flow Task in a package called Package looks like this:

```
Package\Data Flow Task\Aggregate.Inputs[Aggregate Input 1].Columns[LineTotal]
```

Last, annotations are no longer stored as binary streams. Instead, they appear in the XML file as clear text. With better access to annotations in the file, the more likely that annotations can be programmatically extracted from a package for documentation purposes.

Summary

As you have seen through this chapter, Integration Services has benefited from many changes, large and small. Changes in the developer experience will enable faster development times and allow new developers to come up to speed more quickly. Perhaps the biggest change is the introduction of the

project deployment model to simplify changes to package values at run-time. A side benefit of the project deployment model is the catalog which captures information about packages, validations, and execution results in tables that you can query through views or by using built-in reports. This access to information gives you greater visibility into Integration Services than was possible in previous versions without extensive customization. If you're not ready to take advantage of these features, you can continue to manage packages using the legacy deployment model, but you will be missing out on a lot of helpful management features.