



SSIS

Succinctly

by Rui Machado

SSIS Succinctly

By

Rui Machado

Foreword by Daniel Jebaraj



Copyright © 2014 by Syncfusion Inc.
2501 Aerial Center Parkway
Suite 200
Morrisville, NC 27560
USA
All rights reserved.

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: Pedro Perfeito

Copy Editor: Suzanne Kattau

Acquisitions Coordinator: Hillary Bowling, marketing coordinator, Syncfusion, Inc.

Proofreader: Graham High, content producer, Syncfusion, Inc.

Table of Contents

The Story behind the <i>Succinctly</i> Series of Books	8
About the Author	10
Who is This Book For?	11
Introduction	12
Chapter 1 Integration Services Architecture	13
Introduction	13
Explaining the Components	14
Runtime Engine	14
Integration Services Service	14
SSIS Designer	14
Log Provider	14
Connection Manager	15
SSIS Wizard	15
Packages	15
Tasks	15
Event Handlers	15
Containers	15
Control Flow	16
Data Flow Engine	16
Project and Package Deployment Models	16
Developer Environment	17
Introduction	17
SSIS Designer	18
Chapter 2 Packages	20

Introduction	20
Hello World Package—Import and Export Wizard.....	20
Custom Packages.....	26
Introduction	26
Adding a New Package to a Project	26
Executing the Packages	27
Package Explorer.....	29
Package Properties.....	30
Checkpoints	31
Execution	32
Forced Execution Value.....	32
Identification.....	32
Misc.....	32
Security	33
Transactions	33
Version.....	33
Chapter 3 Control Flow.....	34
Introduction	34
Tasks and Containers	35
Introduction	35
Favorite Tasks	36
Common	37
Containers.....	39
Other Tasks	40
Precedence Constraints.....	43
Introduction	43
Advanced Precedence Constraints	45

Connection Managers	47
Introduction	47
Add a Connection Manager at a Solution Level	47
Add a Connection Manager at a Package Level	48
Available Connection Managers	48
Creating a New OLE Database (SQL Server Connection)	49
Fix: Excel/Access Connection Manager Error When Running Package in 64-bit Environment	53
Event Handlers	55
Introduction	55
Using Event Handlers	55
Troubleshooting	56
Breakpoints	57
Using Breakpoints to Stop Before Executing a Task	58
Logging	59
Activate Logging in Your Packages	59
Creating a Simple Control Flow	62
Chapter 4 Data Flow.....	65
Introduction	65
Favorites	66
Common.....	67
Other Transforms.....	69
Other Sources	71
Other Destinations	72
Data Viewers.....	73
Introduction	73
Using Data Viewers	73
Creating a Simple Data Flow	74

Introduction	74
Creating the Data Flow	75
Chapter 5 Variables, Expressions, and Parameters	80
Introduction	80
Understanding the components	80
Creating New Variables	81
Creating New Expressions.....	81
Creating New Parameters.....	82
Handling Slowly Changing Dimensions	83
Introduction	83
Demo: Using the Slowly Changing Dimension Component	83
Change Data Capture	92
Introduction	92
Demo: Using CDC in an ETL Project.....	94
Enable CDC on the Database	95
Enable the SQL Agent in the Server of the CDC.....	95
Enable CDC in the Tables We Want to Monitor for Changes	97
Process the Changed Data in SSIS and Use the SSIS CDC Component to Make It Easier	101
Process Initial Load.....	101
Create the CDC Data Flow	104
Chapter 6 Deploying the Packages	109
Introduction	109
Project Deployment Model.....	110
Package Deployment Model	114

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!



About the Author

Rui Machado is a software developer and software engineer. Born in Portugal, he has a strong .NET background and a special interest in scripting languages. He earned a bachelor's degree in information technologies and a post-graduate degree in business intelligence. He is currently finishing his master's degree in informatics. He works in the business intelligence department of [ALERT Life Sciences Computing](#), a Portuguese software house for healthcare solutions.

A technology enthusiast with a passion for technologies and information systems, he created his first VB.NET application in 2007. Since then, he has worked with C#, C, SQL Server, Windows Phone, PL/SQL, Oracle Data Integrator, PowerShell, BizTalk, Integration Services, and Analysis Services.

Although most of his time is spent working with technology, he also enjoys skateboarding, surfing, and enjoying life with friends and family.

Who is This Book For?

This book is for all developers who want to enrich their knowledge about extract, transform, and load operations (ETL) using SQL Server Integration Services. Although the main targets are developers, every information systems professional will be able to understand how Integration Services works and how to use it to develop ETL processes. This is a book based in the visual environment (SQL Server Data Tools or SSDT) and is designed to be easy to use and understand.

Code Examples

All of the examples in this book were created in Visual Studio. The code examples are formatted in blocks as follows:

```
public int setPersonName(){  
    return 0;  
}
```

Notes

Throughout this book, there are notes that highlight particularly interesting information about a language feature, including potential pitfalls you will want to avoid.



Note: An interesting note about Integration Services.

Introduction

Extract, transform, and load (ETL) is more than just a trendy or fancy concept; it's mandatory knowledge for any developer. Companies need to integrate data more often than you think, either because of new content available for a product catalog, new data trade agreements between parties, technological migrations, or to create data warehouses, which is one of the most popular usage contexts. Because of this, the information systems market demands professionals with ETL skills—either using Microsoft products, Oracle tools, or other vendors' tools. So it's important that you understand how quickly data science is evolving as well as how important integration has become; it's now a stage in the process of delivering information to companies' management layer.

SQL Server Integration Services (SSIS) is Microsoft's business intelligence suite and ETL tool. If you are not considering it for your data integration projects, you are losing one of the most efficient, developer-friendly, and, most importantly, free tools (if you already have a SQL Server paid license, that is). It's not only a tool that allows you to move data from one database to another. It also allows you to clean and transform data in a drag-and-drop development environment that is organized in data flows. By using it, you can maintain your project in an easy and friendly environment.

The main goal for this product is to create integrated, clean, and consistent data to be used by data warehouses (and, optionally, by OLAP-based systems such as SQL Server Analysis Services) to create analysis cubes. However, developers have found it to be more than just an ETL tool; it has been used to consolidate (when you need to bring data from several similar databases into one), integrate (when you need to clean and move data), and move data (when your goal is just moving data without any transformation).

In this book, you will learn how to use SSIS to develop data integration, migration, and consolidation projects so that you, as a developer, can offer your company the ability to populate data warehouses with an analytical purpose. To help you create and maintain the best ETL projects, in this book I will guide you through the development and understanding of SSIS packages, including the output of SSIS developments, their control flow, their execution flow (for example, what will be executed and in what order), tasks, precedence constraints, and other components that this platform provides.

Chapter 1 Integration Services Architecture

Introduction

Microsoft's business intelligence suite is a very complex system of interconnected tools and platforms responsible for managing data, transforming it, delivering it to data warehouses, and visualizing it in many different perspectives so that a decision maker can have the information he or she needs. Each of the platforms are complex and demand a deep understanding of how they work.

This chapter will take you through an overview of the Integration Services architecture with all its components so that you understand how they interact with each other and what you, as a developer, should expect from these components. Let's take a look at the main components of the Integration Services architecture. The following figure shows the diagram used by Microsoft to describe the architecture of Integration Services.

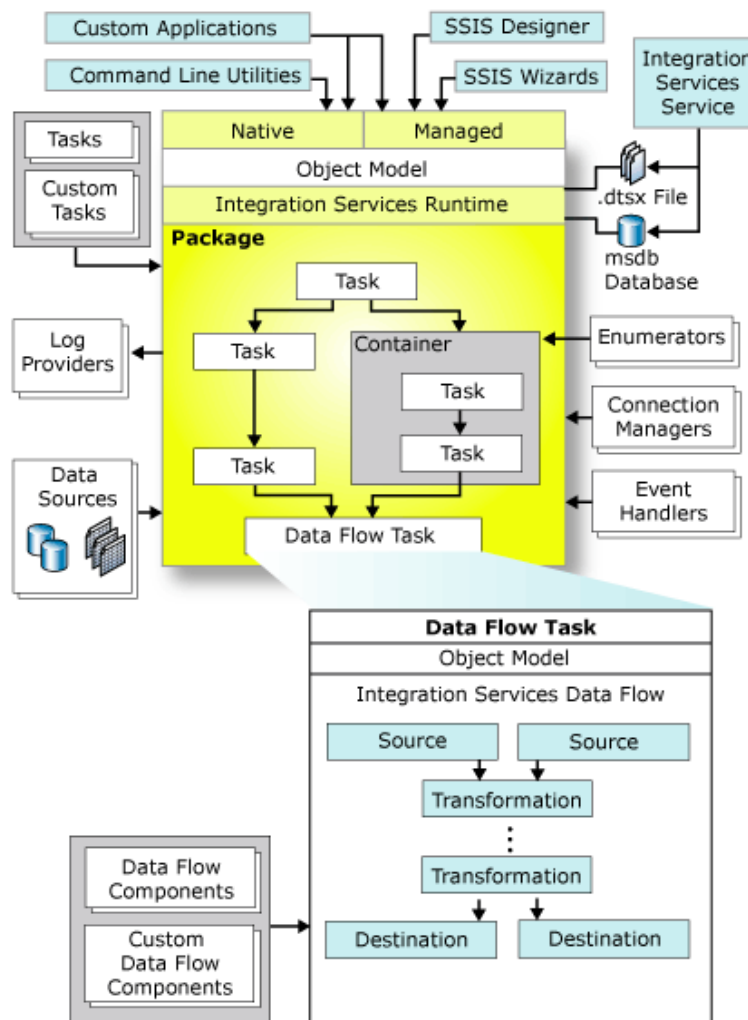


Figure 1: Integration Services Architecture

Explaining the Components

Now let's go through each one of the components of the architecture to give you an overview of what will be explained in depth in the following chapters.

Runtime Engine

The Runtime Engine is responsible for executing the packages that will work with the data according to our needs. As you can see in the previous figure, it's the heart of SSIS as it not only executes the packages, but also saves their layout and provides support for logging, breakpoints, configuration, connections, and transactions. This engine is activated every time we invoke a package execution, which can be done from the command line, PowerShell scripts, the SSIS Designer, the SQL Agent tool, or even from a custom application.

Integration Services Service

The Integration Services service is a Windows service for managing Integration Services packages and is available only in SQL Server Management Studio. It enables developers to import and export packages, view running packages, and view stored packages. It also lets you use SQL Server Management Studio to monitor running Integration Services packages and to manage the storage of packages.

SSIS Designer

The SSIS Designer is where you will be spending most of your time when developing your packages. SSIS Designer is a graphical tool that you can use to create and maintain Integration Services packages. SSIS Designer is available in SQL Server Data Tools (SSDT) as part of an Integration Services project. You can use it to do the following tasks:

- Develop the control flow in a package.
- Develop data flows in a package.
- Add event handlers to the package and package objects.
- View the package content.
- At run time, to view the execution progress of the package.

Log Provider

Managed by the SSIS engine, the log provider's responsibility is to create all the logs needed to monitor the package execution. Every time you run a package, the log provider will write logs with information about the package execution including duration, errors, warnings, and other related information.

Connection Manager

Managed by the SSIS engine, the connection manager's responsibility is to manage the connections defined either at project level or package level. This is available as a tool inside the SSIS Designer so that you can create the connections you need to extract and load data. It can manage connections to FTP servers, databases, files, and Web services. The SSIS connection manager allows you to easily connect to several types of systems, both Microsoft and non-Microsoft platforms.

SSIS Wizard

The SSIS wizard is the simplest method by which to create an Integration Services package that copies data from a source to a destination. When using this wizard, you can only use one type of task.

Packages

The packages are the heart of the engine; in other words, they are the main component of the SSIS engine and they are also subsystems as they are composed of several other artifacts. So, these packages will make the connections to the source and destination targets using the connection managers. They contain the tasks we need to run in order to complete the entire sequence of ETL steps. As you can see in the architecture diagram (Figure 1), these package tasks can be executed singularly, in sequence, grouped in containers, or performed in parallel. Packages are the output of our Integration Services project and have the .dtsx extension.

Tasks

Tasks are control flow elements that define units of work that are performed in a package control flow. An SSIS package is made up of one or more tasks. If the package contains more than one task, they are connected and sequenced in the control flow by precedence constraints. SSIS provides several tasks out of the box that correspond to the majority of problems; however, it also allows you to develop and use your own custom tasks.

Event Handlers

Managed by the SSIS Runtime Engine, event handlers' responsibility is to execute tasks only when a particular event is fired—for example, if an OnError event is raised when an error occurs. You can create custom event handlers for these events to extend package functionality and make packages easier to manage at run time.

Containers

Containers are objects in SSIS that provide structure to packages and services to tasks. They support repeating control flows in packages, and they group tasks and containers into meaningful units of work. Containers can include other containers in addition to tasks. There are four types of containers:

- Task Host Container (encapsulates a single task)
- For Loop Container

- Foreach Loop Container
- Sequence Container

Control Flow

The last four tasks described define the executables in the SSIS Runtime Engine. Packages, containers, tasks, and event handlers are what it will execute at run time. If you only look at the last three (containers, tasks, and event handlers), they are also called control flow tasks as they define the control flow of a package. In this way, packages will have only one control flow and many data flows as we will see later.

The control flow is the brain of a package; it orchestrates the order of execution for all its components. The components are controlled inside it by precedence constraints.

Data Flow Engine

Although we have been talking about the runtime engines, packages, and how you can develop control flow, we have yet to talk about how to define data movements and transformations. This is because it's not an SSIS Runtime Engine responsibility. Instead, it's the responsibility of the data flow engine, also known as the pipeline engine.

This engine is encapsulated in the data flow tasks which are part of the SSIS Runtime Engine and its task components. The data flow engine provides the in-memory buffers that move the data from source to destination. It calls the sources that extract the data from files and relational databases. The data flow engine also manages the transformations that modify data as well as the destinations that load data or make data available to other processes. Integration Services' data flow components are the sources, transformations, and destinations that are included in Integration Services. You can also include custom components in a data flow.

As I have said before, a package can only have one control flow, in which you can add as many data flow tasks as you want.

Project and Package Deployment Models

There is an important detail of SSIS of which you should be aware. SSIS 2012 gives you two types of package deployment. The first type is the legacy deployment model in which each package is treated as a single unit of deployment. The second type is the project deployment model which creates a deployment package containing everything (packages and parameters) needed for deployment in a single file. The use of parameters is restricted to the project deployment model. So, if you want to use parameters, you should be aware that you will need to use the project deployment model. Both models will be discussed and demonstrated in the chapter related to package deployment.

Developer Environment

Introduction

The Integration Services developer environment is created on top of Visual Studio. In SQL Server 2012, the environment comes under the name SQL Server Data Tools - Business Intelligence for Visual Studio 2012. Developing packages will feel familiar to .NET developers who use Visual Studio in their daily work; however, the look and feel of an Integration Services package will be very different as it is a much more visual development environment.

To create an Integration Services project, open the SQL Server Data Tools. After you open it, select **File > New > Project** and then, when the new project window appears, select **Business Intelligence** and under it, **Integration Services**. When you select it, the following screen will be displayed.

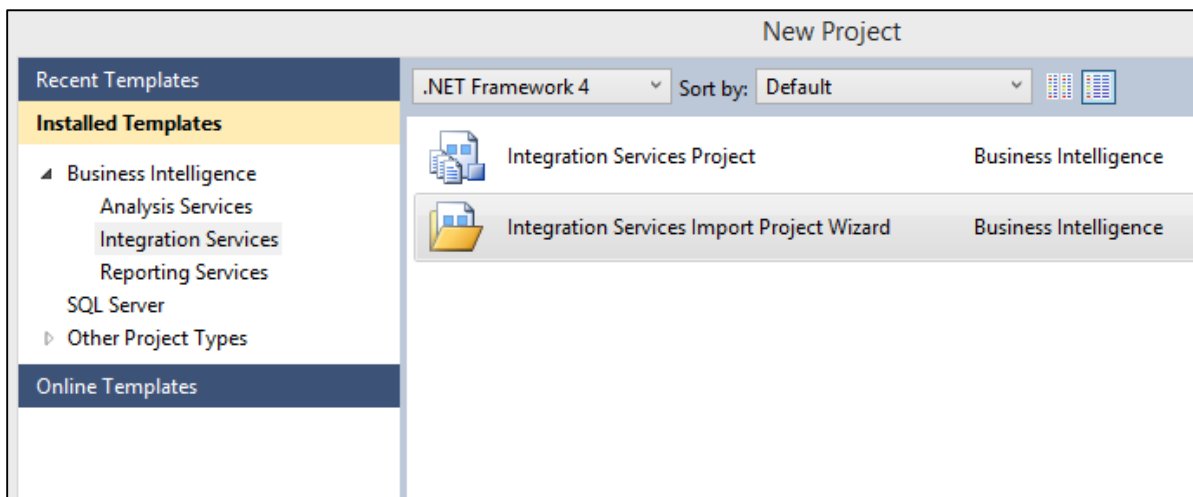


Figure 2: New Integration Services Project

As you can see, there are two distinct project types:

- Integration Services Project
- Integration Services Import Project Wizard

As their names suggest, the difference between them is that the first will create an empty project in which you can start building your SSIS packages. The second one will allow you to import an existing SSIS project and continue your development. For our examples, we will always be using the first project type, Integration Services Project.

Now, select **Integration Services Project** and set its Name, Location, and Solution Name. As I have already mentioned, this is very similar to any other Visual Studio-based project. Once you click **OK**, the SSIS Designer will open and this is where all your developments are going to be made. Welcome to the Integration Services world.

SSIS Designer

The SSIS Designer is where you can create all your packages with its control flow and all its data flows. You can use SSIS Designer to drag and drop the components you want to use and then configure all of them. Before creating our first package, let's take a look at the Designer.

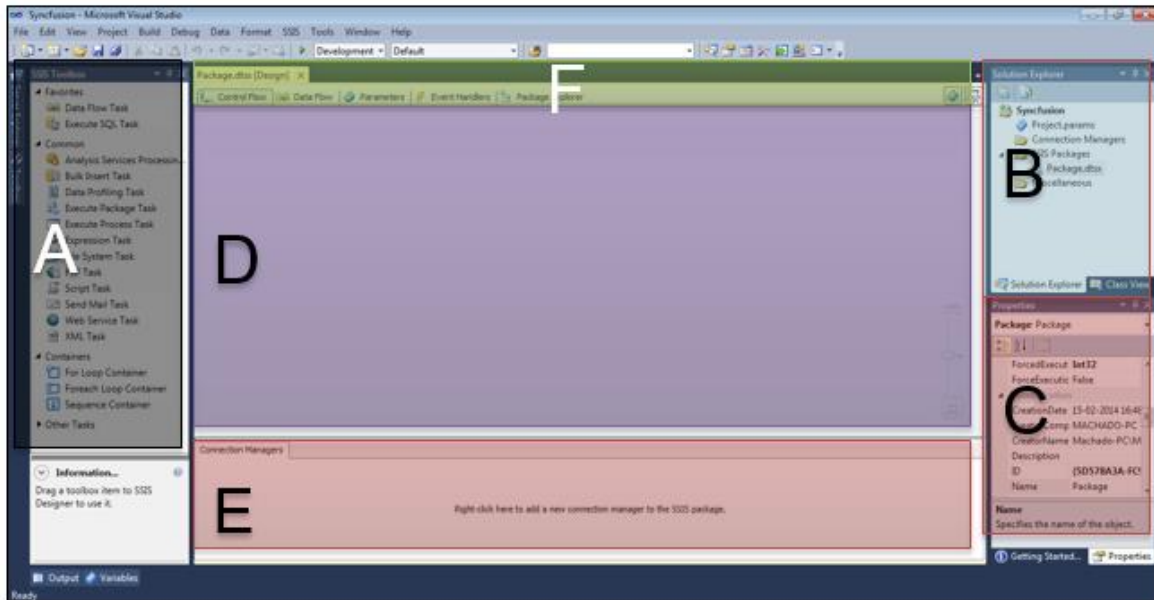


Figure 3: SSIS Designer

As you can see in the previous figure, I have divided the SSIS Designer into six distinct areas. These six areas represent the tools and options you are going to use while developing your packages. The following list describes the six areas:

- A. **The SSIS Toolbox**—This is where you can find available tools to use. The tools displayed will depend on whether you are in the Control Flow tab or Data Flow tab. When you're in the Control Flow tab, you will have tasks. When you're in the Data Flow tab, you will have transformation options.
- B. **The Solution Explorer**—This is where you can find all the project artifacts including project connection managers, packages, and project parameters.
- C. **Properties window**—Again, like in any other Visual Studio-based project, this window displays all the available properties for the currently selected object.
- D. **The design window**—This is where you develop your flows. You start by dragging an object from the SSIS Toolbox and then dropping it into this design window. After adding the item to this designer, you configure it and connect it to the next component you add. This window displays all the package control flows and all its data flows.
- E. **The package connection manager**—This is where you create and see connection managers restricted to this package which cannot be used by any other package.
- F. **The package tabbed-toolbar**—This allows you to navigate between the control flow, the currently selected data flow (even so, you can change the currently selected data flow inside the data flow design window), the package parameters, the package event handlers, the package explorer, and the progress tab to see the package execution log information.

As Microsoft advises, it's important to keep in mind that the SSIS Designer has no dependency on the Integration Services service, the service that manages and monitors packages. Plus, it is not required that the service be running to create or modify packages in SSIS Designer. However, if you stop the service while SSIS Designer is open, you can no longer open the dialog boxes that SSIS Designer provides, and you may receive the error message "RPC server is unavailable".

To reset the SSIS Designer and continue working with the package, you must close the designer, exit SQL Server Data Tools, and then reopen SQL Server Data Tools, the Integration Services project, and the package.

Chapter 2 Packages

Introduction

Packages are the output of an Integration Services project; they are the executables you are going to run in order to process an ETL task. The best definition of a package defines it as a collection of connections, control flow elements, data flow elements, event handlers, variables, parameters, and settings that you assemble by either using the graphical design tools that SQL Server Integration Services provides or by building programmatically.

These tasks will be pulled together using the SSIS Designer; it is a very easy process of connecting them using arrows. However, internally, the Integration Services engine will use precedence constraints to not only connect the tasks together but to also manage the order in which they execute. The engine does this based on what happens in each task or based on rules that are defined by the package developer.

This is a basic definition of a package. It is important to understand that package development is not just about adding tasks. When you finish a package, you can add advanced features such as logging and variables to extend the package functionality. When you finish the package developments, they need to be configured by setting package-level properties that implement security, enable restarting of packages from checkpoints, or incorporate transactions in package workflows.

The package output file will have the .dtsx file extension which holds inside it an XML structure with all the operations needed to execute in order to meet the developer's needs. Similar to other .NET projects, the file-based code is marked up using the development environment. It can then be saved and deployed to a SQL Server so that you can schedule its execution. However, this is not the only way to execute these packages. You can, for example, use PowerShell scripts to get the same results or even use SSIS DB-stored procedures.

Hello World Package—Import and Export Wizard

Although we will be using the SSIS Designer to create custom packages with all control flow tasks and data flow transformations, SSIS comes with an Import and Export Wizard that assists in creating simple “from source to target” packages. This wizard allows you to configure a source, a destination, and only a few more options to copy data. An important note about this wizard is that the target table will be created when you run this package; however, as you will see in the following steps, you will also be able to configure a “drop” and “create” (re-create) operation.



Note: You can open the Import and Export Wizard by using the Run window and typing “DtsWizard”.

Although the wizard can be opened from outside the SSIS Designer, we will use it inside the SSIS Designer to create this first “Hello world” package. To do so, right-click the **SSIS Package**

in the Solution Explorer and choose the option **SSIS Import and Export Wizard**, as shown in the following figure.

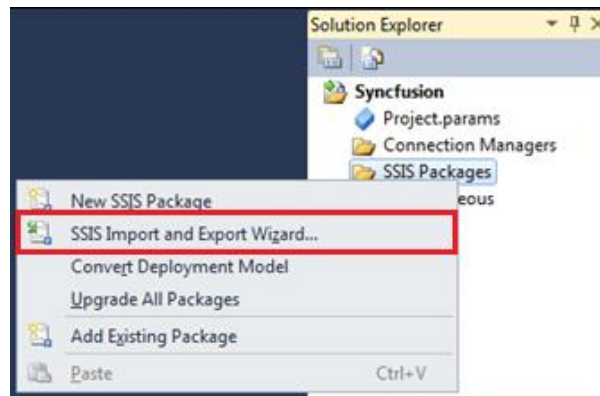


Figure 4: Opening the Import and Export Wizard

Once you click it, SSIS will display the first screen of the Import and Export Wizard. This is just a simple information screen; you have to click **Next** to start the configuration of this package.

In the following screen, you need to configure the source database from which you want to retrieve data. You just need to set the data source of the package (which, in my example, will be SQL Server), the server name (which, in my example, will be “localhost” but you can use “.” or “(local)” to identify localhost), the authentication type (which, in my example, will be Windows Authentication), and the source database. The following figure shows my example’s configuration.

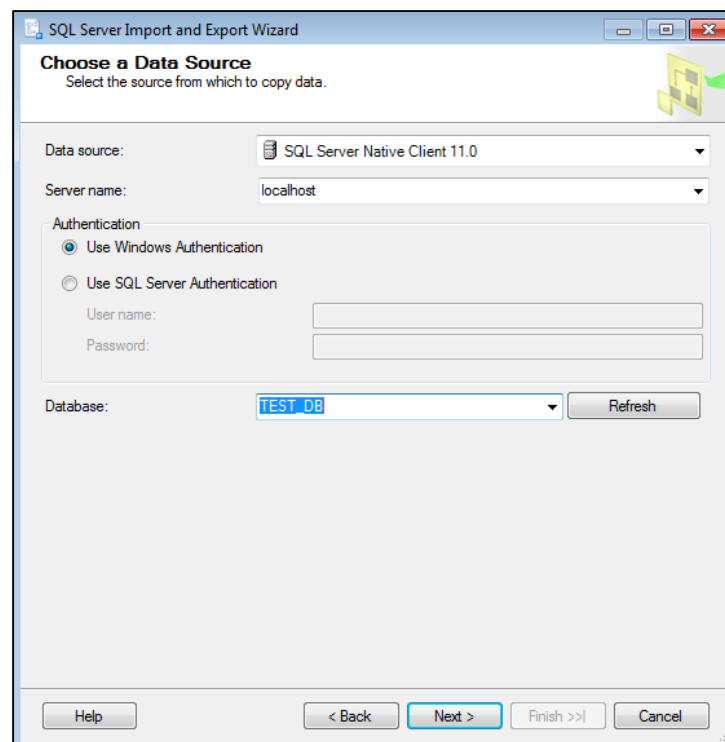


Figure 5: Configuring the Source

Once you click **Next**, you will be prompted with a similar screen to configure the target database. You should make the same configurations you did for the source and click **Next**.

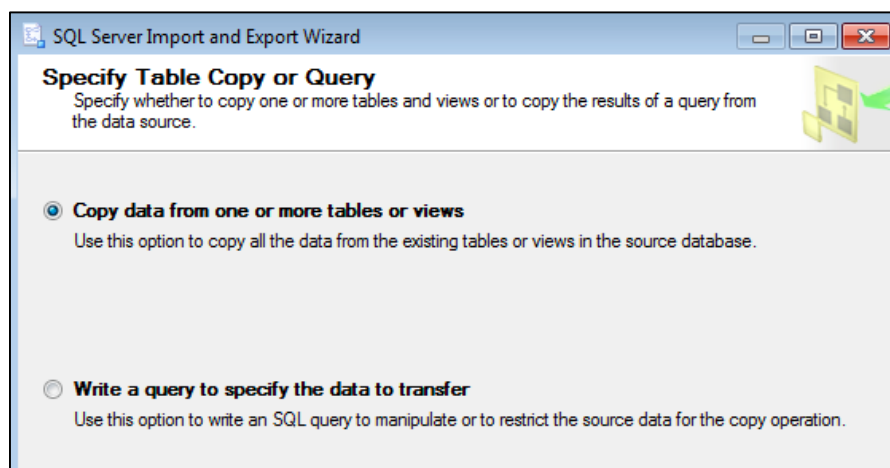


Figure 6: Choosing how to retrieve data from the source

Now comes one of the most important steps. This step is when you decide how you want to retrieve the data from the source, by either doing so directly from a table or view, or by writing your own query. The choice between one and the other is not linear. However, based on my experience, I can advise you to keep the retrieve logic in a view that is stored in your database, and inside your packages, to retrieve the data from them.

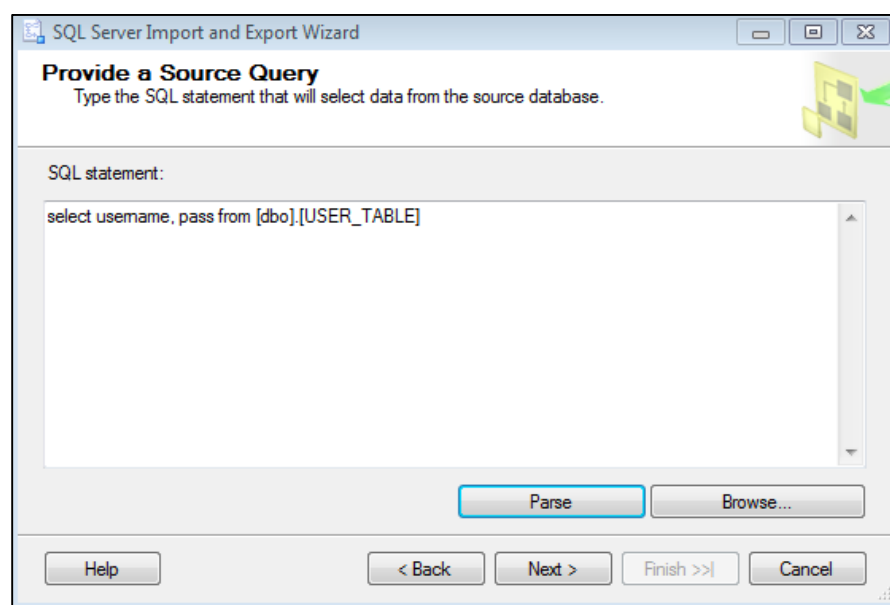


Figure 7: Query for Retrieving Data

This way, if your target database changes, you can dynamically change the view, and, for example, change the format of your columns. You can even apply transformations (using SQL functions) to them without having to change, save, and redeploy your packages.

In our example, we will choose the second option: writing a simple query that will retrieve just two columns from a source table. Once again, this is not a best practice because, if some target columns changed, I would need to open the package, change the query, save, and redeploy it. However, for this simple “Hello World”, it won’t be a problem.

Once you complete this step, click **Next** and a new screen will be displayed. This screen allows you to select the tables and views you want to copy to the target database. Because we have created a custom SQL query, you will only have one option (which is a “Query” object as you can see in the following figure).

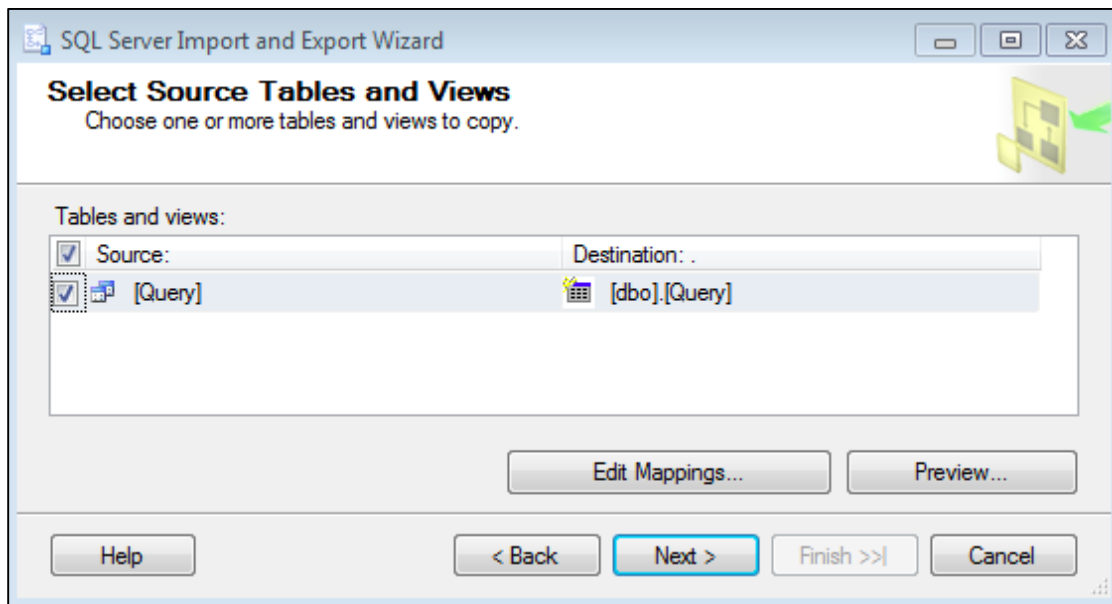


Figure 8: Choosing tables and views to copy

Once you finish selecting the tables and views you want to copy, click **Next**. However, before doing so, it is worthwhile to explore some of the interesting options in this screen. As shown in the previous figure, there are two important buttons: Preview, which allows you to see the data being copied, and Edit Mappings, which we will discuss in further detail later. If you click the **Edit Mappings** button, you will see the following screen.

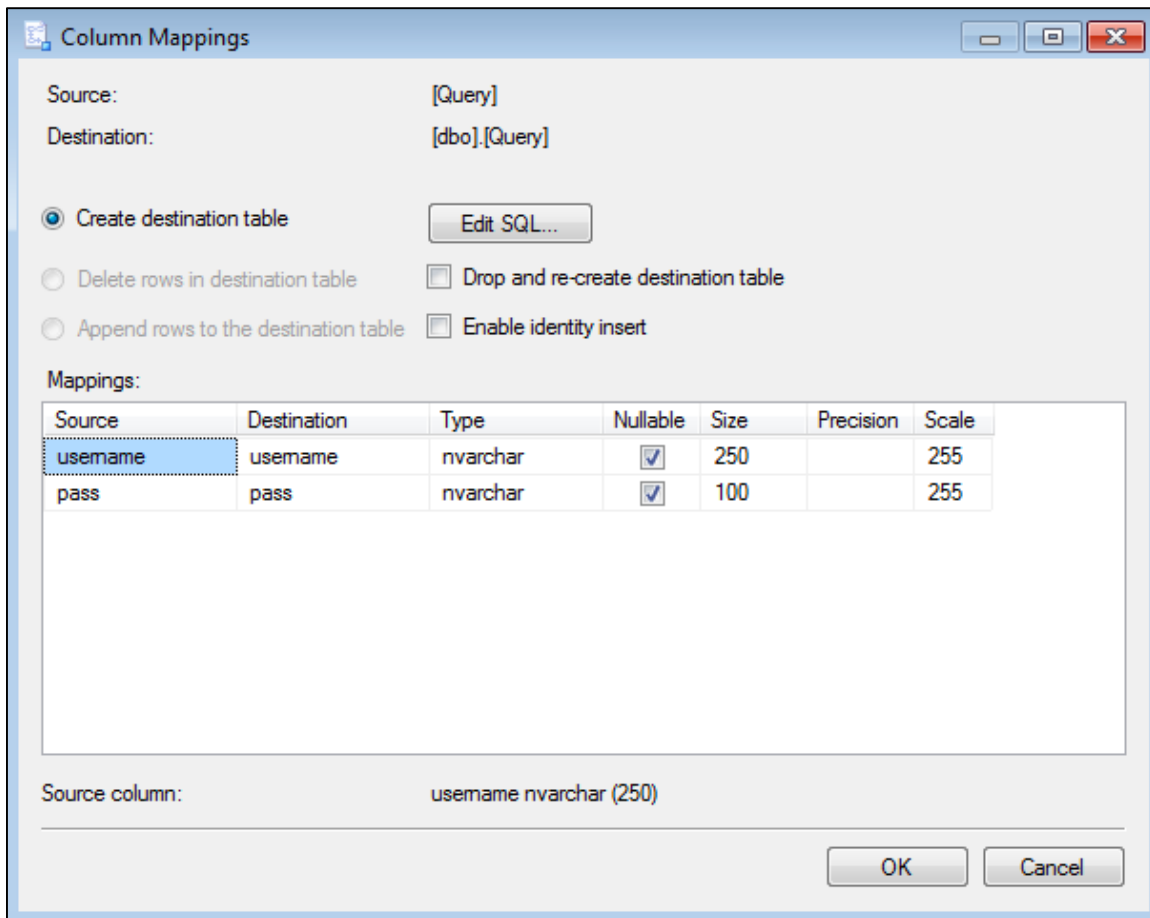


Figure 9: Column Mappings

If you have chosen, like I did, to retrieve data using a SQL query, the wizard will only allow you to create a new table in the destination database as it assumes there isn't any data in the the Query columns. However, you have two available check boxes. The first check box is "Drop and re-create destination table" which will re-create the destination table if it already exists. This is a dangerous option as it will drop the table whether or not you know it exists. The second check box is "Enable identity insert" which allows the values to be inserted in the identity field. This way, the exact identity values are moved from the source database to the destination table.

You can also edit the destination table column names and types by editing the mappings grid (also available in this screen). Once you set up all the settings, click **OK**. In the Select Source Tables and Views window, click **Next**.

This page is the last step in the Import and Export Wizard. You will be able to review all your configurations. If everything is correct, click **Next** to generate the package.

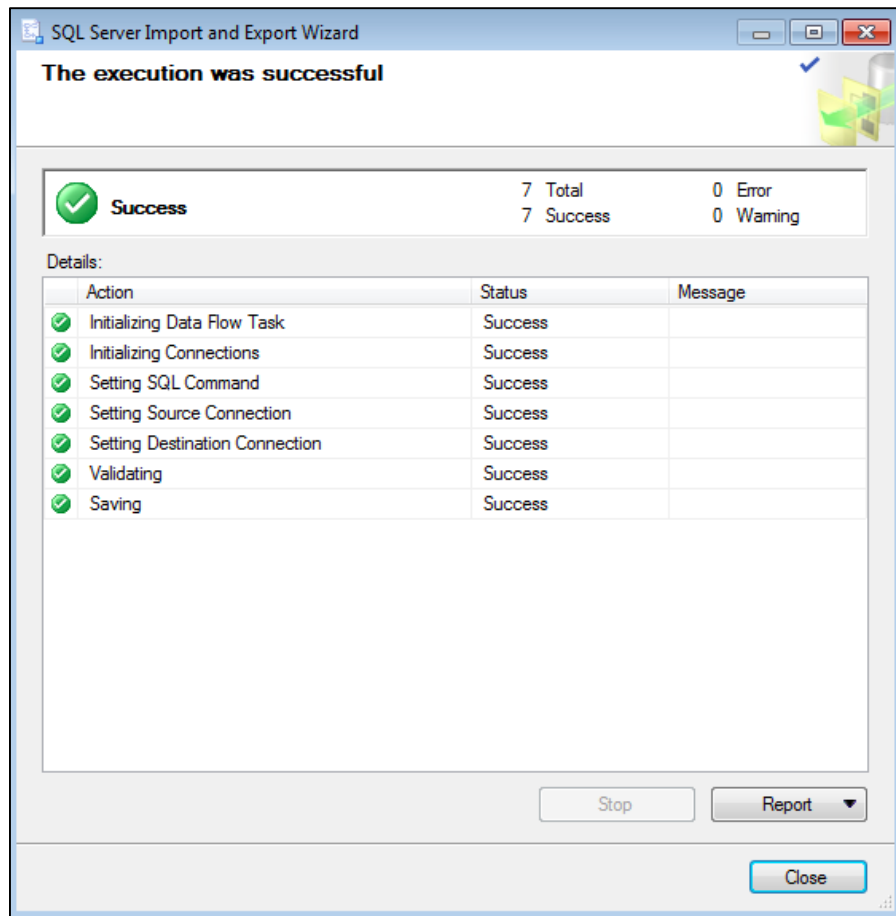


Figure 10: Generating the Package

Now the package will become available in the Solution Explorer. You can try to run it by right-clicking the package and selecting **Execute Package**. And that's it! You have just created your first package. Next, we will learn how to create custom packages with more logics and transformations.

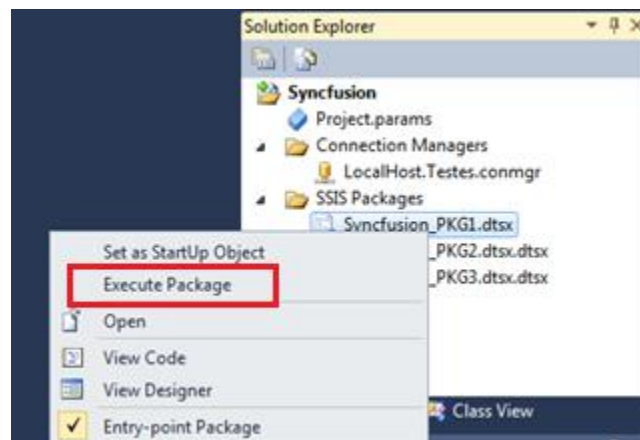


Figure 11: Execute Package

Custom Packages

Introduction

Although the Import and Export Wizard is a very powerful tool for simple ETL jobs, in most cases, when using Integration Services, you are going to need more complex structures with execution logics that involve more than just a single data flow. You may need to execute SQL queries to create support tables, send emails if something went wrong in the process, make several data transformations, and even process multiple data sources to create a single insert. By using the Import and Export Wizard, you are using just a small percentage of Integration Services' capabilities.

The following chapters will show you how SSIS can easily become your best friend in extract, transform, and load operations using some out-of-the-box, reusable components. These components will save you a lot of time and allow you to maintain and evolve your projects. It's important to understand that most of the development logics provided by SSIS could be made using SQL queries and programming languages such as C# or PowerShell (with some considerable effort). However, the beauty of SSIS is, without developing a single line of code, you can create fantastic data-oriented solutions without losing time thinking about how to implement an algorithm. You just need to focus on the most important thing: creating a stable solution that's easy to maintain and understand, using a very friendly design environment.

Adding a New Package to a Project

To add a new package to our Integration Services project, right-click the **SSIS Packages** folder in the Solution Explorer, and then select **New > SSIS Package**. This will create a blank package that is ready for you to develop.

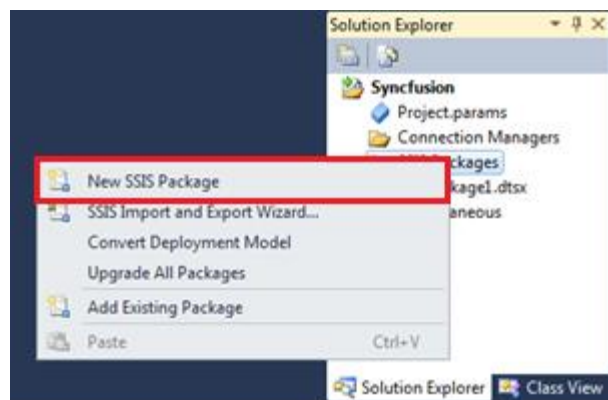


Figure 12: New SSIS Package

This will automatically add a new package to your project that is visible in the Solution Explorer and has a default name. To change the package name after it has been added to the project, right-click the package and select **Rename**.

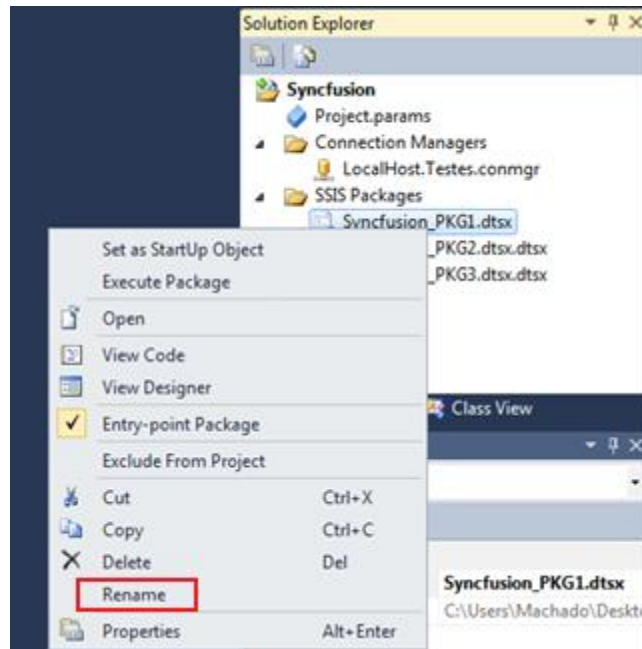


Figure 13: Rename Package

After adding your new package to the project, you will need to configure your control flow. Although it's not mandatory to start defining all of the control flow tasks of your package, you can only start developing the transformation logics after adding a data flow task to it. This way, the control flow is the first component you will look at after adding a new package to a project. To open the newly added packages, just double-click them in the Solution Explorer.

Executing the Packages

Executing the packages is very easy to do in Integration Services. You have several options. The first and more visible option is to click the **Play** button icon on the toolbar as shown in the following figure.

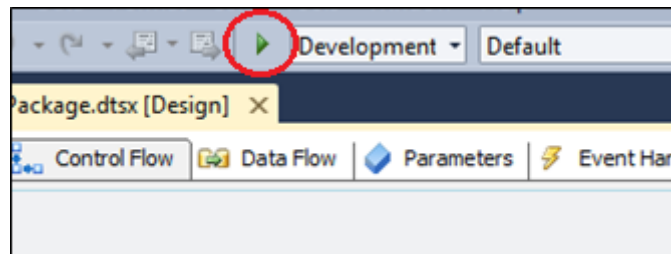


Figure 14: Execute Project

The second option is to use the Solution Explorer. You can right-click the package you want to execute and then select **Execute Package** as shown in the following figure. This is the most often used technique because, when you click **Play**, the SQL Server Data Tools will create a new build of the entire project and run every single project inside it in debug mode. After execution, it leaves the execution window open so that you can evaluate the execution steps. By selecting **Execute Package**, you will only run the packages you want.

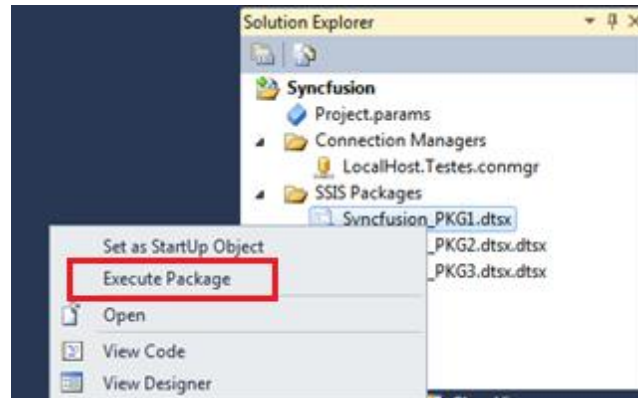


Figure 15: Execute Package

Once the execution ends, you need to stop the debug mode to go back to the development environment. Do this by clicking **Stop Debug** in the toolbar.

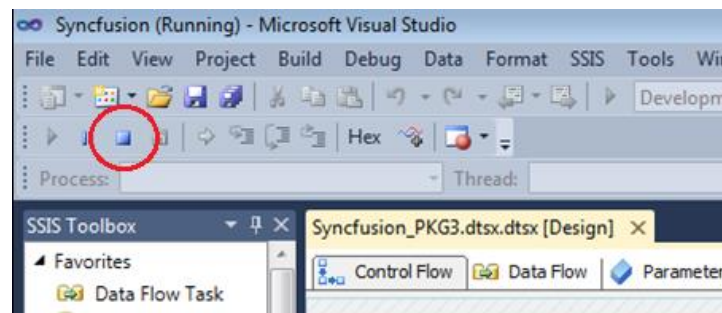


Figure 16: Stop Debug

One last component of the package execution options is a window labeled Execution Progress, which allows you to see the log results of the package execution, including errors if something went wrong, or just the execution time and steps if everything executed correctly. You can access this window by clicking the **Progress** tab that is shown after running a package. The following figure shows the location of the Progress tab.

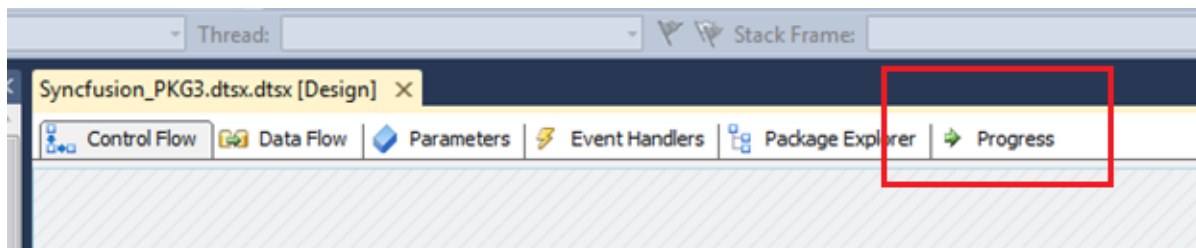


Figure 17: Progress tab to view execution results

This window will show you a hierarchical structure showing all the objects inside the package and all of its execution results. You can see the start date, end date, error messages, and other log messages which can be used to evaluate if everything is working as it should. The following figure shows an example of an execution result in which an Execute SQL task was used to create a table, but an error was displayed stating that this particular table already exists in the target database.

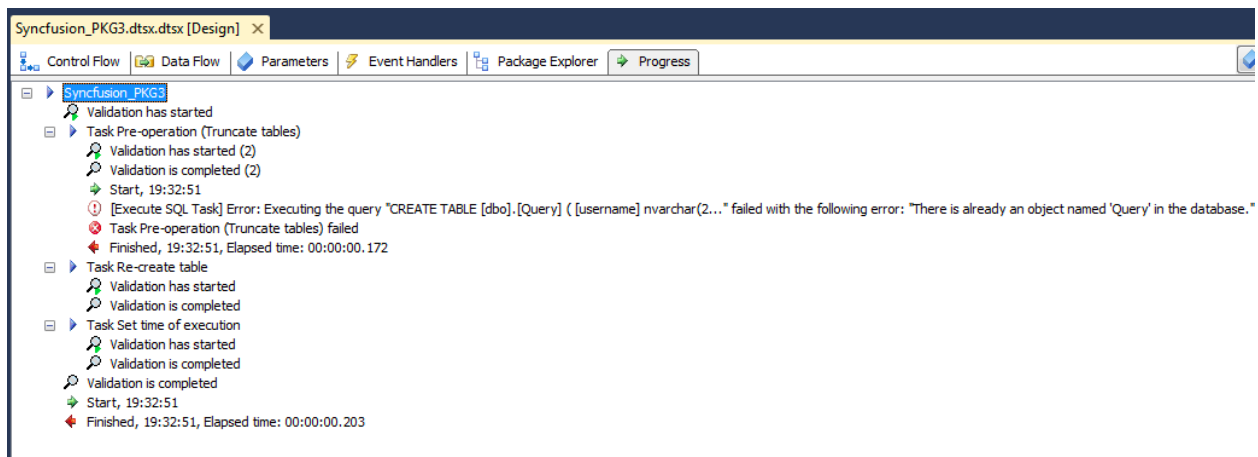


Figure 18: Execution Progress

You might face some difficulties when trying to read a full message. If you do, you can copy and paste the message to an external text editor and evaluate the message there in full detail.

Package Explorer

The Package Explorer is a very useful tool to get an overview of all your package's objects. With its hierarchical view, you are able to navigate between all of your design panes with all its tasks, connections, containers, event handlers, variables, and transformations. It basically summarizes your developments and allows you to easily navigate between the properties of all your objects and change them.

To access the Package Explorer, open a package, click the **Package Explorer** tab, and expand the objects you want to edit. Or just follow the structure as a single view.

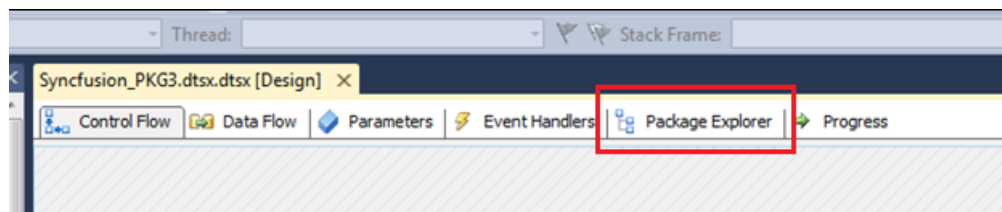


Figure 19: Package Explorer Tab

Once you open the Package Explorer, you will see the hierarchical view that shows the various objects you have added to control flows and data flows.

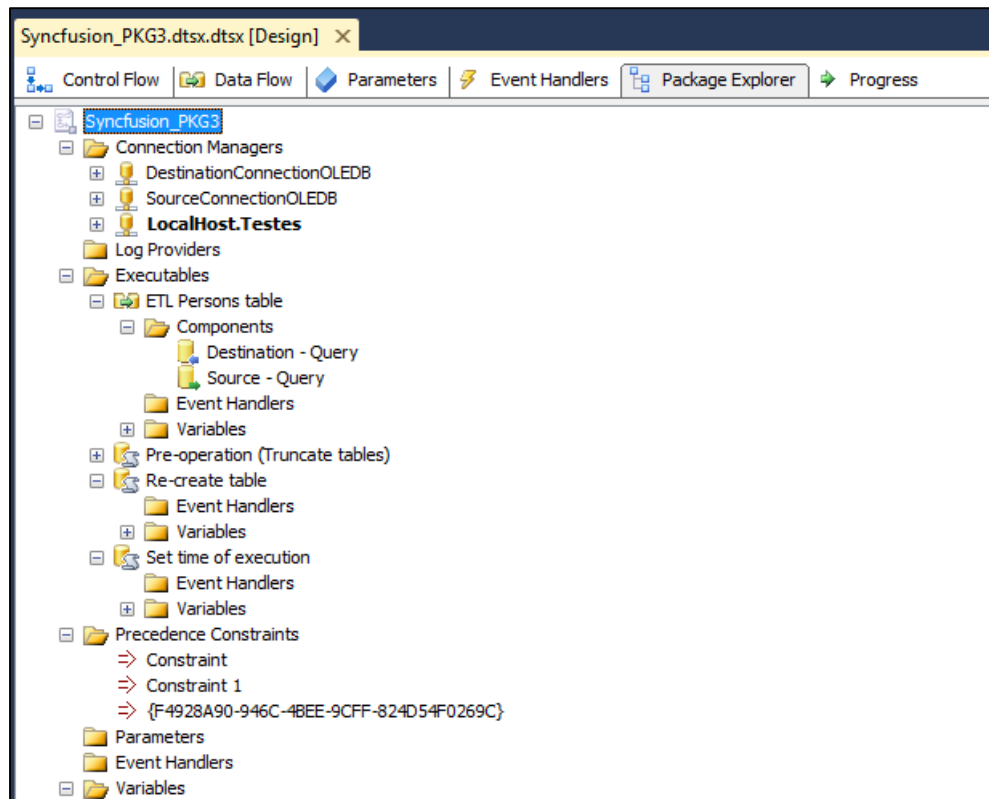


Figure 20: Package Explorer

Package Properties

Like any other object that is clicked in a Visual Studio-based solution, when you open a package in the Integration Services design environment, because that object gets the focus of the IDE, the Properties window will reflect that particular object properties. In this case, these are packages. These properties allow you to change basic attributes such as the package name and version, but they also allow you to add important attributes such as passwords to execute the package.

In this section, I will guide you in the analysis of every property of packages in Integration Services so that you can understand its important features. The first thing is to clarify where you can find these properties. So, start by opening a package and then look at the Properties window. The following figure shows the package properties.

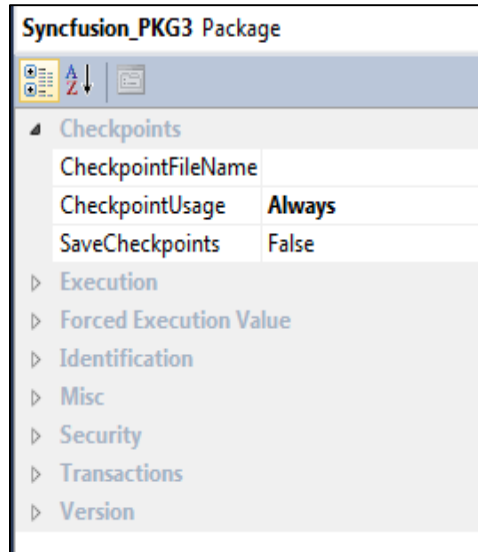


Figure 21: Package Properties

As you can see in the previous figure, package properties are organized into eight groups:

- Checkpoints
- Execution
- Forced Execution Value
- Identification
- Misc
- Security
- Transactions
- Version

Before explaining each group of properties, there is an important topic I cannot avoid discussing and that is checkpoints. When activated, checkpoints force SSIS to maintain a record of the control flow executables that have successfully run. In addition, SSIS records the current values of user-defined variables. This current context will be stored in an XML file which is defined in the package property window. After that, if a package fails during execution, SSIS will reference the checkpoint file when you try to rerun the package. It will first do so by retrieving the current variable values as they existed prior to package failure and, based on the last successful executable that ran, start running the package where it left off. That is, it will continue executing from the point of failure, assuming we've addressed the issue that caused the failure in the first place.

My goal in this section is more than teaching you how to apply these properties and how they can be used in your scenarios—my goal is to teach you about their existence. This way, when a particular problem appears, you will know its source. The following sections will help you understand which properties exist and for what reason.

Checkpoints

- CheckpointFileName—File name of the checkpoint context values. This file is deleted after a successful run of the package.

- CheckpointUsage—Tells when a checkpoint file should be used (Always, never, or if the file exists).
- SaveCheckpoints—Allows you to enable and disable checkpoints.

Execution

- DelayValidation—Indicates whether package validation is delayed.
- Disable—Indicates whether the package is disabled.
- DisableEventHandlers—Specifies whether the package event handlers run.
- FailPackageOnFailure—Specifies whether the package fails if an error occurs in a package task.
- FailParentOnError—Specifies whether the parent container fails if an error occurs in a child container.
- MaxConcurrentExecutables—The number of executable files that the package can run concurrently (-1 indicates no limit).
- MaximumErrorCount—The maximum number of errors that can occur before a package stops running.
- PackagePriorityClass—The Win32 thread priority class of the package thread.

Forced Execution Value

- ForcedExecutionValue—If ForceExecutionValue is set to True, this is a value that specifies the optional execution value that the package returns.
- ForcedExecutionValueType—The data type of ForcedExecutionValue.
- ForceExecutionValue—A Boolean value that specifies whether or not the optional execution value of the container should be forced to contain a particular value.

Identification

- CreationDate—Date the package was created.
- CreatorComputerName—Computer on which the package was developed.
- CreatorName—Name of the package developer.
- Description—Description of the package functionality.
- ID—The package GUID which is assigned when the package is created. This property is read-only.
- Name—Package name.
- PackageType—The package type.

Misc

- Configurations—Package configurations (only available for Package Deployment Approach).
- Expressions—Create expressions for the package. Allows you to set properties based on attributes or parameters.
- ForceExecutionResult—The execution result of the package. The values are None, Success, Failure, and Completion.
- LocaleId—A Microsoft Win32 locale.

- **LoggingMode**—A value that specifies the logging behavior of the package. The values are Disabled, Enabled, and UseParentSetting.
- **OfflineMode**—Indicates whether the package is in offline mode.
- **SuppressConfigurationWarnings**—Indicates whether the warnings generated by configurations are suppressed.
- **UpdateObjects**—Indicates whether the package is updated to use newer versions of the objects it contains, if newer versions are available.

Security

- **PackagePassword**—The password for package protection levels (EncryptSensitiveWithPassword and EncryptAllWithPassword) that require passwords.
- **ProtectionLevel**—The protection level of the package. This property is used to specify how sensitive information is saved within the package and also whether or not to encrypt the package or the sensitive portions of the package.

Transactions

Packages use transactions to bind the database actions that tasks perform into atomic units. By doing this, they maintain data integrity. In other words, they allow developers to group tasks together to be executed as a single transaction.

- **IsolationLevel**—Sets the way locking works between transactions.
- **TransactionOption**—The transactional participation of the package. The values are NotSupported, Supported, and Required.

Version

- **VersionBuild**—The version number of the build of the package.
- **VersionComments**—Comments about the version of the package.
- **VersionGUID**—The GUID of the version of the package. This property is read-only.
- **VersionMajor**—The latest major version of the package.
- **VersionMinor**—The latest minor version of the package.

And that's it. While some properties seem useful only when reading their definitions, others will only show their applicability when your needs demand it. Integration Services is an aggregation of many components and tools; some projects might require you to use all of them, while others just demand a small set.

Now it's time to start digging into the main components of this platform and start developing our packages. The first thing we need to learn about is the control flow. We will be learning all the control flow tasks, how you can connect them, and even about how these tasks connect to external systems.

Chapter 3 Control Flow

Introduction

This chapter will explain how control flows are used inside Integration Services packages. The topics discussed will be based on their main components: tasks, connection managers, event handlers, and containers.

In the [architecture chapter](#), I explained that packages can only contain one control flow, so let's learn how to develop one. Start by adding a new package to the project and then opening the Control Flow tab in SSIS Designer. When the Control Flow tab is active, the SSIS Toolbox lists the tasks and containers that you can add to it. The following figure shows the location of the Control Flow tab inside the package designer.

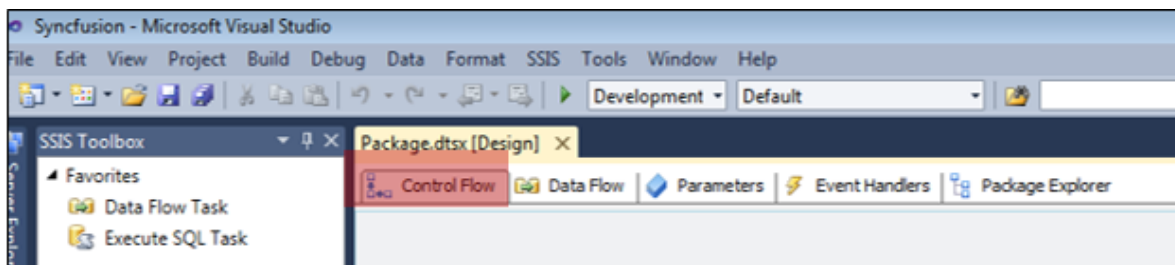


Figure 22: Control Flow Tab

Building the control flow is easy in the Integration Services Designer; you just need to access the Control Flow tab and then drag and drop all the tasks you need from the SSIS Toolbox into the Control Flow design pane. Once you have a task added to the design window, you can double-click it to specify its behavior within its properties. As I have already mentioned, the control flow is the orchestrator of your package execution. So, while dealing with this flow, you should be thinking about what you want your package to do—the “how” will be defined inside each task.

After you configure the task, you can link it to other tasks by using Precedence Constraints. Once you click on a task, you'll notice a green arrow pointing down from it. If you drag and drop this arrow into another task, you will create what is called an OnSuccess constraint. Very simply, these precedence constraints link the individual executables together and determine how the workflow moves from one executable to the next. This topic will be explained in deeper detail in the section about [Precedence Constraints](#).

The following figure shows three tasks connected using these precedence constraints; for example, in a scenario in which you want to send different emails if the execution of a SQL task succeeds or if it fails. In this scenario, the precedence constraints used differ (OnSucess and OnFail).

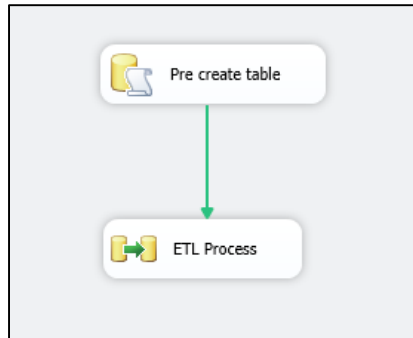


Figure 23: Linked Tasks

Once you have all the tasks connected using these precedence constraints and configured the tasks using the appropriate wizards, you have the control flow of your package defined and the workflow of its execution completed. It might look simple; however, each task has its own properties and configuration. You may spend a significant amount of time developing all the tasks needed for the control flow execution.

Learning about the existing tasks is very important so you can know which one you'll need to use to respond to a particular problem. Because of this, the next section will show you all the tasks that are available out of the box with Integration Services 2012. It will take you through some demonstrations about how to configure the most often used tasks. (The most often used tasks might be different from developer to developer; those included in the next section were chosen based on my experience).

Tasks and Containers

Introduction

Tasks are the atomic or individual work components inside the control flow. They allow you to execute a particular job inside your packages. The total number of tasks inside your package control flow defines the orchestration of its execution. Comparing these tasks to programming languages, you can see them as functions, a particular code block that changes or gets the state of your object. However, in Integration Services Designer, you don't need to code anything; you just drag and drop tasks into the design surface and configure them using their configuration wizards.

Integration Services provides several built-in tasks in its Toolbox. To access them, select the **Control Flow** tab inside the package designer, and choose from the available ones. You will find four distinct groups: Favorites, Common, Containers, and Other Tasks. Expanding these groups will allow you to see and use their tasks.

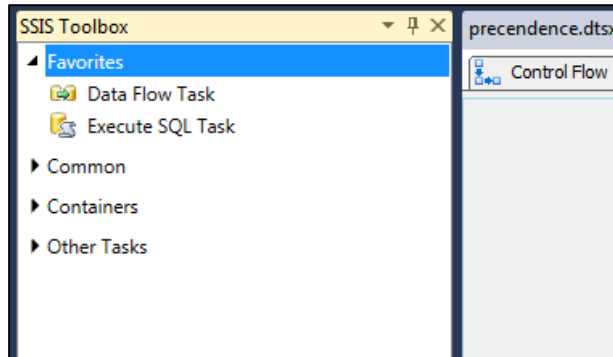


Figure 24: SSIS Toolbox Tasks

Choosing the correct task to use will not be easy if you don't know which tasks exist and what they do. In this chapter, I will explain all the tasks available in the SSIS Toolbox. After that, I will demonstrate how you can use some of these tasks.

Favorite Tasks

This is your personal, customizable group of the control flow tasks toolbox. By default, it contains two tasks which are two of the most often used ones. The first is the data flow task which processes the source to the destination and retrieves, transforms, and inserts it into databases or files. The second is the execute SQL task which allows you to run DML (manipulation), DDL (definition), and DCL (control) commands against a relational database.

You can move your task group by right-clicking it and choosing one of the available move options as shown in the following figure.

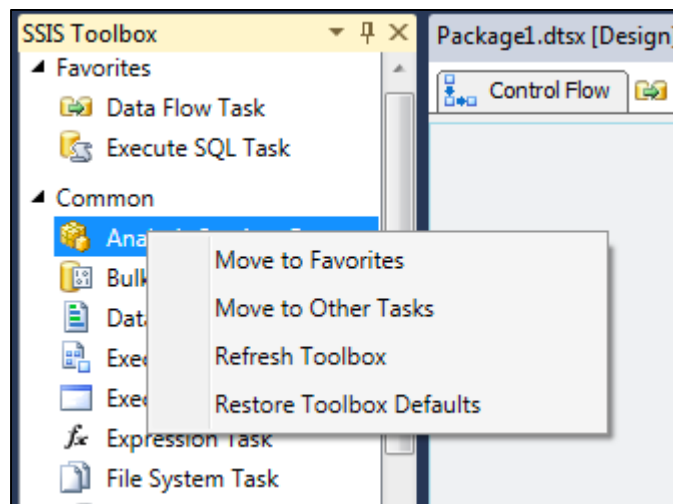




Figure 25: Moving a Task Group

The following table lists the default tasks available in the Favorites group.




Table 1: Favorite Tasks








Icon	Name	Description
	Data Flow Task	This task allows you to move data between sources and destinations while transforming and cleaning. This is one of the most important tasks in Integration Services because it is responsible for the ETL processing. This can be the only task inside a package control flow.
	Execute SQL Task	This task will run SQL data manipulation or data definition scripts, or allow you to execute stored procedures against a relational database. They could be creating a new table, truncating an existing one, or any other operation you need.



Common

The Common tasks group includes the most commonly used tasks in Integration Services projects. The following table explains each task.

Table 2: Common Tasks

Icon	Name	Description
	Analysis Services Processing Task	This task allows you to process SQL Server Analysis Services objects such as cubes, dimensions, and mining models.
	Bulk Insert Task	This task allows you to load data into a table by using the BULK INSERT SQL command. This command is often used to insert large amounts of data more quickly because it does not allow data to be transformed while being inserted into a database; it's inserted directly.
	Data Profiling Task	This task allows you to examine data before or after managing it. This task will profile and identify data quality problems such as unexpected NULL values.

Icon	Name	Description
	Execute Package Task	This task allows you to execute a package from within another package. This way, you can share the logics of your package between packages, creating a modular structure that is easier to maintain.
	Execute Process Task	This task will execute a program external to the package. You can call applications such as Notepad or the Windows Calculator, batch files, command-line utilities, or third-party programs.
	Expression Task	This task allows you to set variables to an expression at run time, such as the current number of records in a SQL Server database table.
	File System Task	This task allows you to interact with your file system. You can perform operations such as create folders, rename or delete directories, and even handle performing the same operations against files. You can use it to create a log folder, for instance.
	FTP Task	This task allows you to manage directories and send or receive files from an FTP server. You can, for instance, load a file from a remote FTP server into your data flow, or transfer a flat file to a remote FTP server.
	Script Task	This task allows you to perform functions that are not provided by the standard SSIS tasks. You can, for instance, invoke an external API as a part of the control flow. Imagine, for instance, invoking a social network API and retrieving data from it, adding it to your data flow.
	Send Mail Task	This task allows you to send emails using the SMTP protocol. You can use it to notify, via email, the success or failure of an ETL process.

Icon	Name	Description
	Web Service Task	This task allows you to run a method on a Web service and assign the return to a variable or a file. You can, for instance, get the daily weather temperature or wind speed and then assign the return value to a variable.
	XML Task	This task allows you to work with XML data. Common operations made with this task are retrieving XML documents, using XPath expressions, and merging multiple documents. You can also validate, compare, and save updated documents to files or variables.

Containers




Containers are not tasks; they are used to group tasks together logically into unique units of work. In this way, you can treat several tasks as one in terms of flow logics. There are several advantages to using containers to group related work unit tasks. One of the advantages is having the ability to define the scope of variables and event handlers at a containers level or, even better, to be able to iterate a redefined enumeration until a condition is verified. There are four types of containers:

- Task Host Container
- For Loop Container
- Foreach Loop Container
- Sequence Container

You can find the containers in the SSIS Toolbox, inside the Containers group. The following table explains each of the four containers.

Table 3: Containers



Shape	Name	Description
N/A	Task Host Container	This is not a usable container. This encapsulates a task every time you add one to the control flow. This means that every time you add a task to the control flow, you are creating a new container.










Shape	Name	Description
	For Loop Container	This container allows you to iterate through a series of tasks until a condition is met. In this case, you can use container variables to define the expression being evaluated. You can, for example, use it to load a subset of files in a directory or perform a data flow for each day between dates. In this case, you could define a condition based on the day.
	Foreach Loop Container	This container allows you to iterate through a series of files or records in a finite data set and then execute tasks inside the container for each of the elements. You can use it for the same purpose as the For Loop container; the difference is that this one will not use a condition to stop the iteration but, instead, the last element of the data set. This way, you don't need any expression, just a data set.
	Sequence Container	This container allows you to logically organize area-related tasks. This can be used, for instance, to execute in parallel sequences of tasks. This is a useful tool if you have big control flows with lots of tasks because these containers have a collapse and expand mechanism to better organize your flows.









Other Tasks





The Other Tasks group includes infrequently used tasks in Integration Services projects. However, they are as important as, or even more important than, the common ones. If you master these tasks, you will be able to do pretty much anything against a SQL database without writing a line of code. The following table explains each of the tasks.

Table 4: Other Tasks

Shape	Name	Description
	Analysis Services Execute DDL Task	This task allows you to execute DDL operations in SQL Server Analysis Services, such as creating, dropping, or altering a cube.
	Back Up Database Task	This task allows you to specify the source databases, destination files or tapes, and overwrite options to run a backup operation over a database.

Shape	Name	Description
	CDC Control Task	This task allows you to maintain and interact with the change data capture (CDC) feature of SQL Server. You can use it to control the life cycle of change data processing packages.
	Check Database Integrity Task	This task allows you to perform internal consistency checks of the data and index pages within a database.
	Data Mining Query Task	This task allows you to run predictive queries against Analysis Services data-mining models.
	Execute SQL Server Agent Job Task	This task allows you to select SQL Server Agent jobs to run as part of the maintenance plan.
	Execute T-SQL Statement Task	This task allows you to run any Transact-SQL query against your SQL Server database.
	History Cleanup Task	This task allows you to delete historical data related to backup and restore, SQL Server Agent, and maintenance plan operations. The user interface allows you to specify the type and age of the data to be deleted.
	Maintenance Cleanup Task	This task allows you to remove files left over from executing a maintenance plan.
	Message Queue Task	This task allows you to send or receive messages from a Microsoft Message Queue (MSMQ). An important thing to note is that the queue must be on the same computer on which the package is running.
	Notify Operator Task	This task allows you to send an email message to any SQL Server Agent operator.

Shape	Name	Description
	Rebuild Index Task	This task allows you to rebuild indexes to reorganize metadata and index pages. This improves performance of index scans and seeks. This task also optimizes the distribution of data and free space on the index pages, allowing faster future growth.
	Reorganize Index Task	This task allows you to defragment and compact clustered and non-clustered indexes on tables and views.
	Shrink Database Task	This task allows you to reduce the disk space consumed by the database and log files by removing empty data and log pages.
	Transfer Database Task	This task allows you to transfer a SQL Server database between two instances of SQL Server. It can also be used to copy a database within the same server.
	Transfer Error Messages Task	This task allows you to transfer SQL Server user-defined error messages between instances of SQL Server. This task can be configured to transfer all error messages or only specified ones. User-defined messages have an identifier equal to or greater than 50000.
	Transfer Jobs Task	This task allows you to transfer SQL Agent jobs between instances of SQL Server. You can transfer all jobs or just selected ones.
	Transfer Master Stored Procedures Task	This task allows you to transfer one or more user-defined store procedures between master databases on instances of SQL Server.
	Transfer Logins Task	This task allows you to transfer one or more logins between instances of SQL Server. You can transfer all logins or just some selected ones.

Shape	Name	Description
	Transfer SQL Server Objects Task	This task allows you to transfer one or more types of objects between instances of SQL Server.
	Update Statistics Task	This task allows you to update the query optimizer to have the most recent information of the distribution of data values in the tables. This will optimize your query execution.
	WMI Data Reader Task	This task allows you to run WQL queries against the Windows Management Instrumentation. This enables you to read the event log, get a list of applications that are installed, or determine hardware that is installed.
	WMI Event Watcher Task	This task allows you to make SSIS wait for and respond to certain WMI events that occur in the operating system.

Precedence Constraints

Introduction

Precedence constraints are used by the Integration Services control flow component to define the order of execution of each task and container, manage the workflow of a package, and handle error conditions. There are three main types of precedence constraints, which can be defined by their colors:

- **Blue**—Defines the precedence constraint for a task or container completion; can be success or failure.
- **Green**—Defines the precedence constraint for a task or container success.
- **Red**—Defines the precedence constraint for a task or container failure.

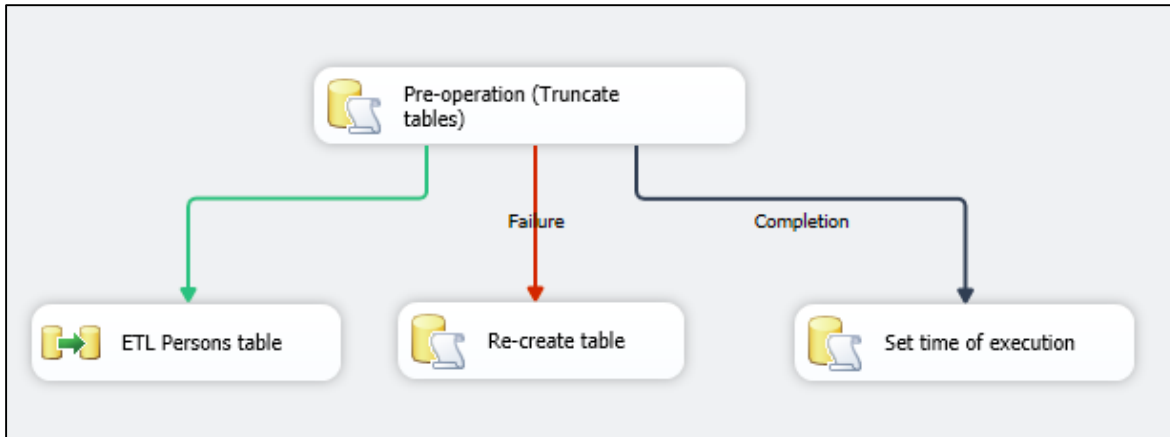


Figure 26: Available, basic precedence constraints

The behavior of these constraints is easy to understand. In the success case, the task or container linked to the current one being executed will only run if the current execution is successful. When dealing with the failure constraint, the behavior is the opposite: the next task or container will run only if the current execution fails. When using the completion constraint, the next task will be executed regardless of whether the current execution fails, which means that no matter what, this will always be executed.

It's important to keep in mind that you can set up all three of these for the same task or container. Let me give you an example. You can define that after the execution of a data flow task you are always going to execute a SQL script, send an email in case of failure, and run four more data flow tasks in case of success.

While linking two tasks or containers, the default precedence constraint is a green arrow designating success because it is the most commonly used. However, you can change its behavior just by right-clicking the arrow and choosing a different evaluation. The previous figure shows what linking tasks using all precedence constraints looks like. The following figure shows the menu for changing the type of precedence constraint.

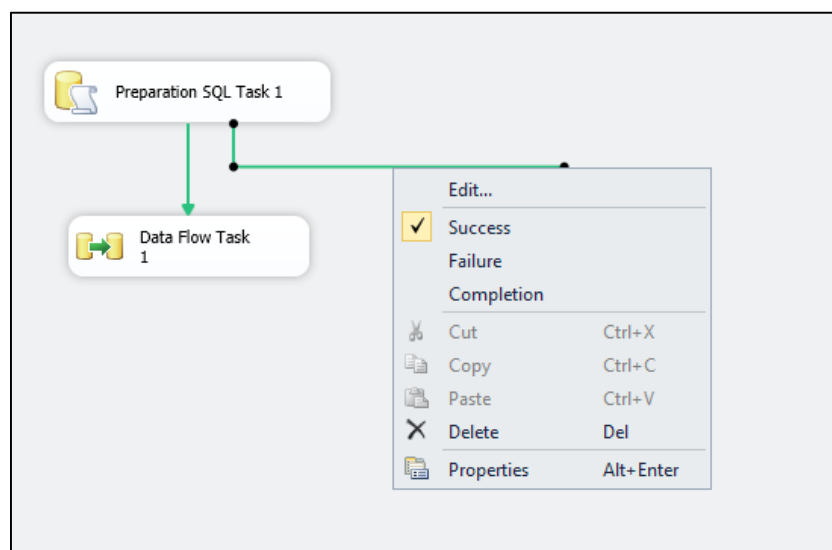


Figure 27: Changing the precedence constraint type

Advanced Precedence Constraints

Although evaluating the success or failure of a task or containers might be a good mechanism to control the flow of your package, there are some scenarios in which you might need more control of the flow execution. Such scenarios include running only on a particular date or time, or even based on a particular number of records in the source table. To meet such requirements, you can edit the conditions that are being evaluated in a precedence constraint.

To edit the conditions inside a precedence constraint, start by linking the tasks or containers by using the three options available (green, red, or blue). Next, double-click the arrow linking them to open the Precedence Constraint Editor. The following figure shows what this editor looks like.

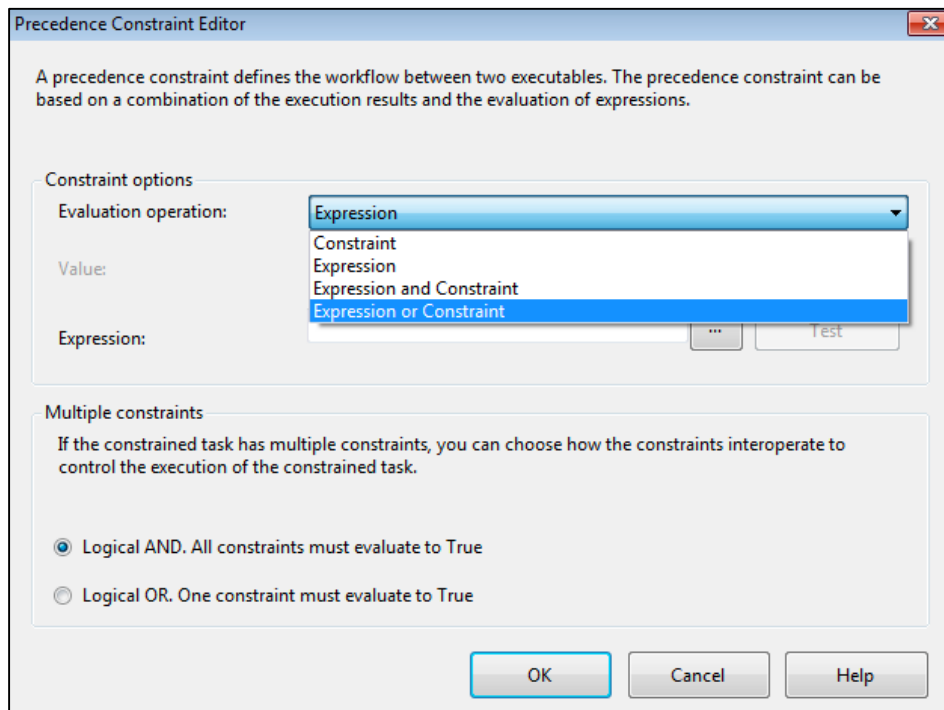


Figure 28: Precedence Constraint Editor

By using the Precedence Constraint Editor, you are able to evaluate more than just the success or failure of a task or container execution. Without getting into too much detail, let's take a brief look the four available options.

As you can see in the previous figure, the first option, Constraint, is the basic one. It is used when you need one of the three basic available constraints (Success, Failure, or Completion). However, the flexibility of precedence constraints comes with the other options. The full list of options is explained below.

- **Constraint**—Uses only the execution result of the precedence executable to determine whether or not the constrained executable runs. The execution result of the precedence executable can be Success, Failure, or Completion. This is the default operation.
- **Expression**—Evaluate an expression to determine whether the constrained executable runs. If the expression evaluates to true, the constrained executable runs.

- Expression and Constraint—An expression and a constraint that combines the requirements of the execution results of the precedence executable with the returned results from evaluating the expression.
- Expression or Constraint—An expression or a constraint that uses either the execution results of the precedence executable or the returned results from evaluating the expression.

There is also an interesting concept called multiple constraints, which is commonly used when a task has multiple inputs with different precedence constraints for each other. Grouping your constraints enables you to implement complex control flow in packages; however, you need to tell SSIS how to manage the interoperability of all precedence constraints.

To do so, you have two options: grouping them using the AND logical operator in which all constraints must be evaluated to true, or by using the OR logical operator in which one of the precedence constraints must be evaluated to true.

If you choose any evaluation operation other than Constraint, Integration Services will allow you to open the Expression Builder in the Precedence Constraint Editor so that you can develop the expression being evaluated. The following figure shows the Expression Builder and an example expression to evaluate. In this example, I use a preset variable to check if there are records to process.

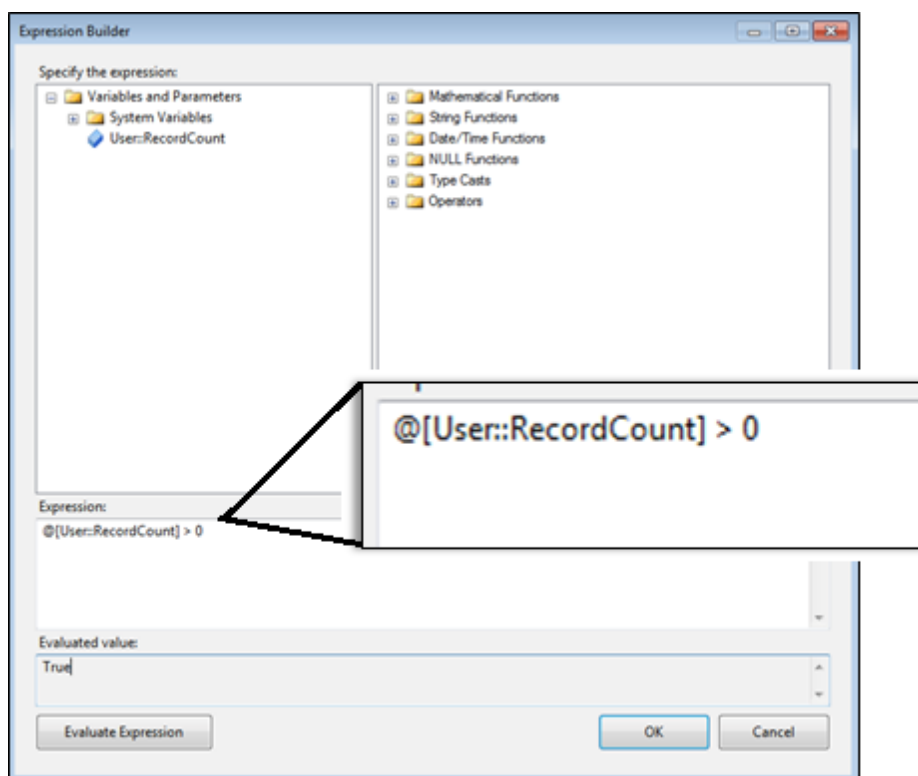


Figure 29: Expression Builder

Connection Managers

Introduction

Connection managers are components that allow you to connect to third-party platforms in order to retrieve, insert, or perform operations on their data. Integration Services allows you to specify two types of connection managers: one is at a solution level and the other is at a package level. If you define a connection at a package level, only the control flow and data flow of that specific package will be able to use it. On the other hand, if you define it at a solution level, then all control flows and data flows of all packages in that solution will be able to use them. The following figure shows the Add SSIS Connection Manager window.

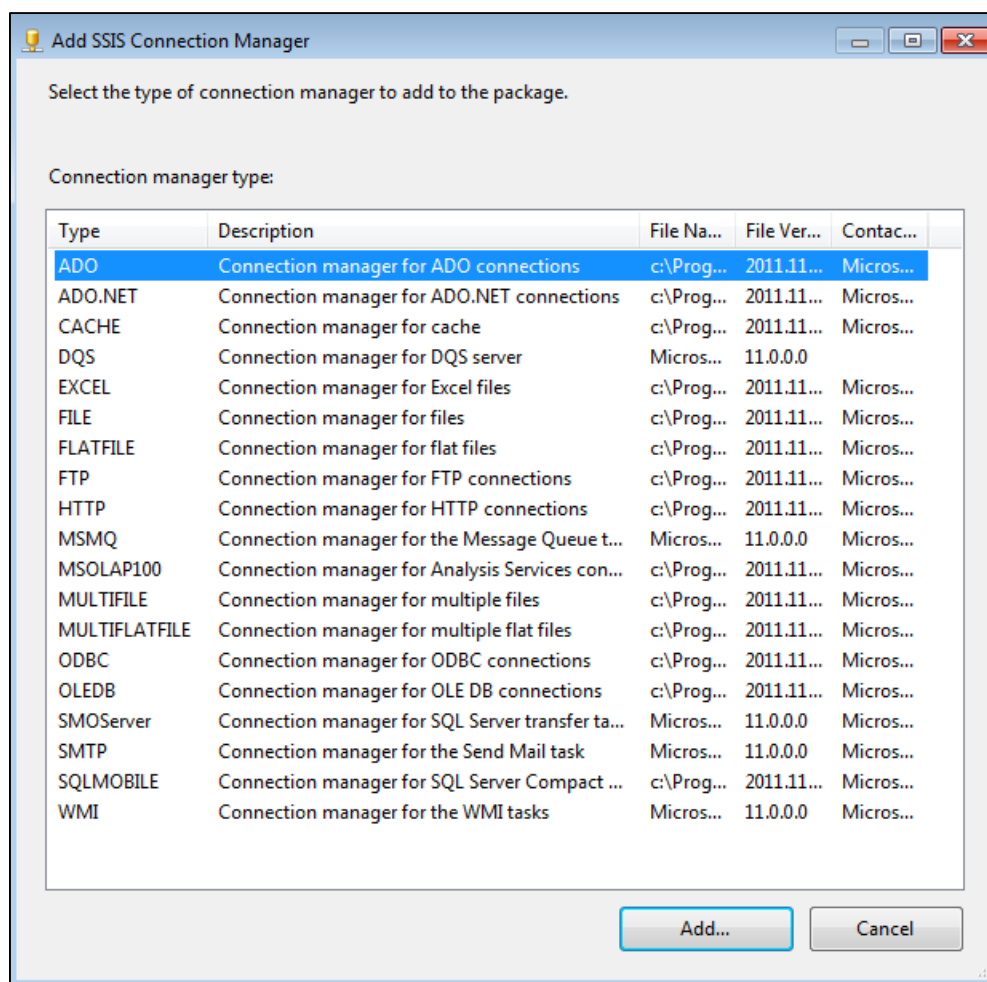


Figure 30: Available Connection Managers

Add a Connection Manager at a Solution Level

To configure a connection manager at a solution level, you should configure it in the Solution Explorer. Right-click the **Connection Managers** section, and select **New Connection Manager**. The following figure shows what this option looks like.

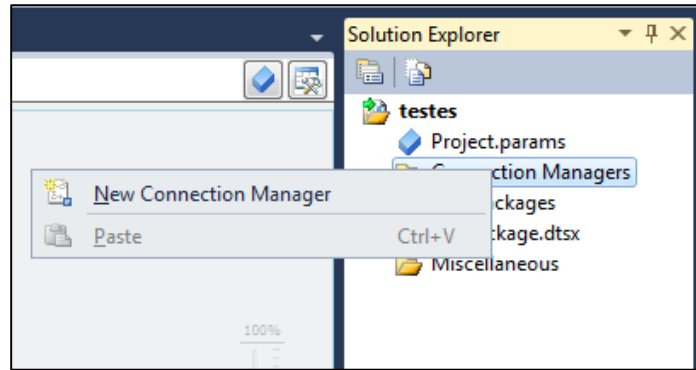


Figure 31: Creating a solution-level connection

Add a Connection Manager at a Package Level

To configure a connection manager at a package level, you should use the Connection Manager pane on the bottom of your package. To create a new connection, right-click inside the **Connection Manager** pane. For this connection manager, you have several options. To use the default connection manager, select **New Connection**. If you prefer to use any of the shortcuts for connection types such as OLE Database, Flat File, ADO.NET, Analysis Services, or file, you can select them directly from here. The following figure shows the menu that is displayed after right-clicking in the Connection Manager pane.

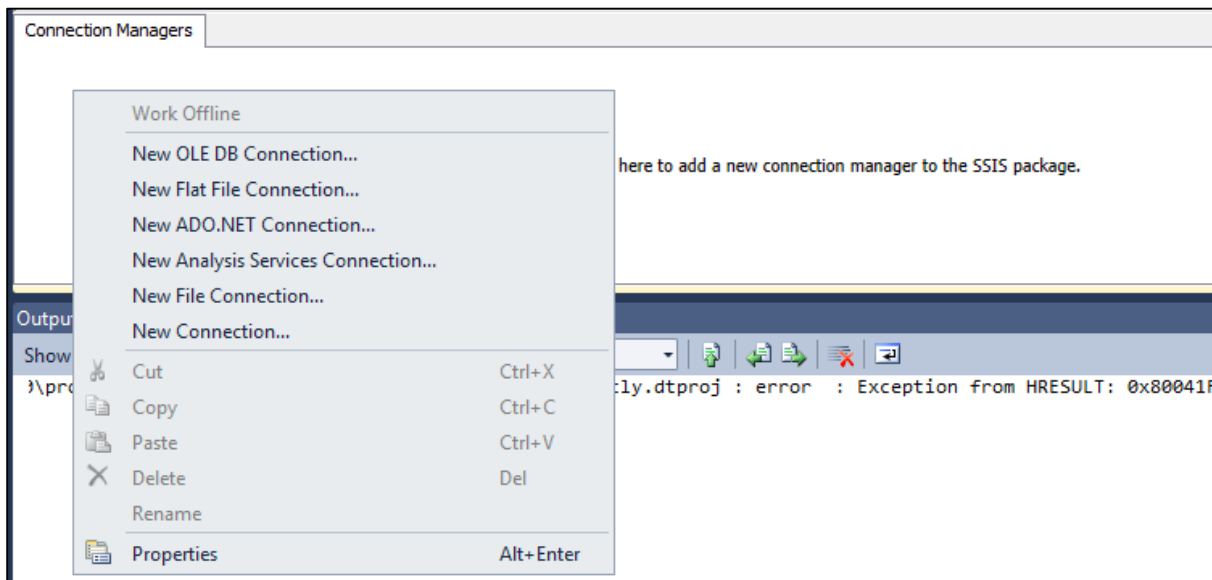


Figure 32: Creating a package-level connection

Available Connection Managers

There are several available connection managers in Integration Services for connecting to Excel files, SQL Server databases, FTP clients, and others. In this chapter, I will explain how to use some of the most commonly used connection types and how to configure them so that you can create a successful connection to your source or destination targets. The full list of available connection types is as follows:

- ADO
- ADO.NET
- Cache
- DQS Server
- Excel
- File
- Flat File
- FTP
- HTTP
- MSMQ (Message Queues)
- MSOLAP100 (Analysis Services)
- MULTIFILES (Multiple Files)
- MULTIFLATFILES (Multiple Flat Files)
- ODBC
- **OLE Database (SQL Server)**
- SMO
- SMTP
- SQL Mobile
- WMI

The OLE Database (SQL Server) emphasized in bold is the one that I will explain how to use. All the connection managers work similarly as they are wizards created to assist you in configuring your connections faster and with more security, thus avoiding errors.

Creating a New OLE Database (SQL Server Connection)

In this demo, we will create a new connection to a SQL Server 2012 database using the OLE Database (OLE DB) connection manager. To start the operation, follow the instructions provided previously to add a new connection manager at the project level.

After the connection manager wizard opens, start by selecting the **OLEDB** type and clicking the **Add** button as shown in the following figure.

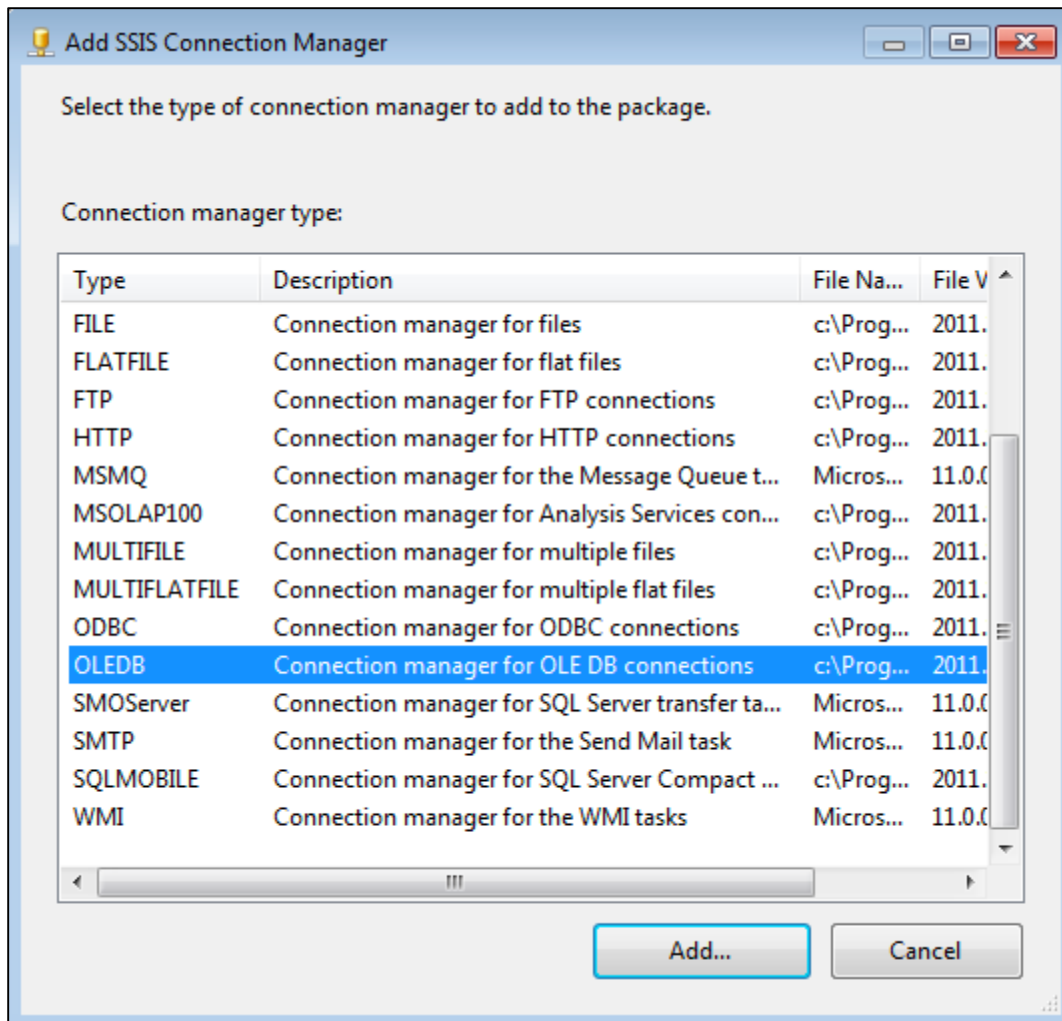


Figure 33: Selecting an OLE DB connection manager

The wizard for the OLE DB connection manager will open. At this time, you will need to know your server name as well as the database you want to access. Remember also that the username you will be using to connect to the database should now be configured in the database.

Integration Services stores the previous connections configured in previous projects. This way, when you open the wizard, if you have already configured any OLE DB connection type you will see it in the wizard as shown in the following figure. If the connection you need has already been configured, you can select it directly. For our example, let's click **New** to configure a new connection from scratch.

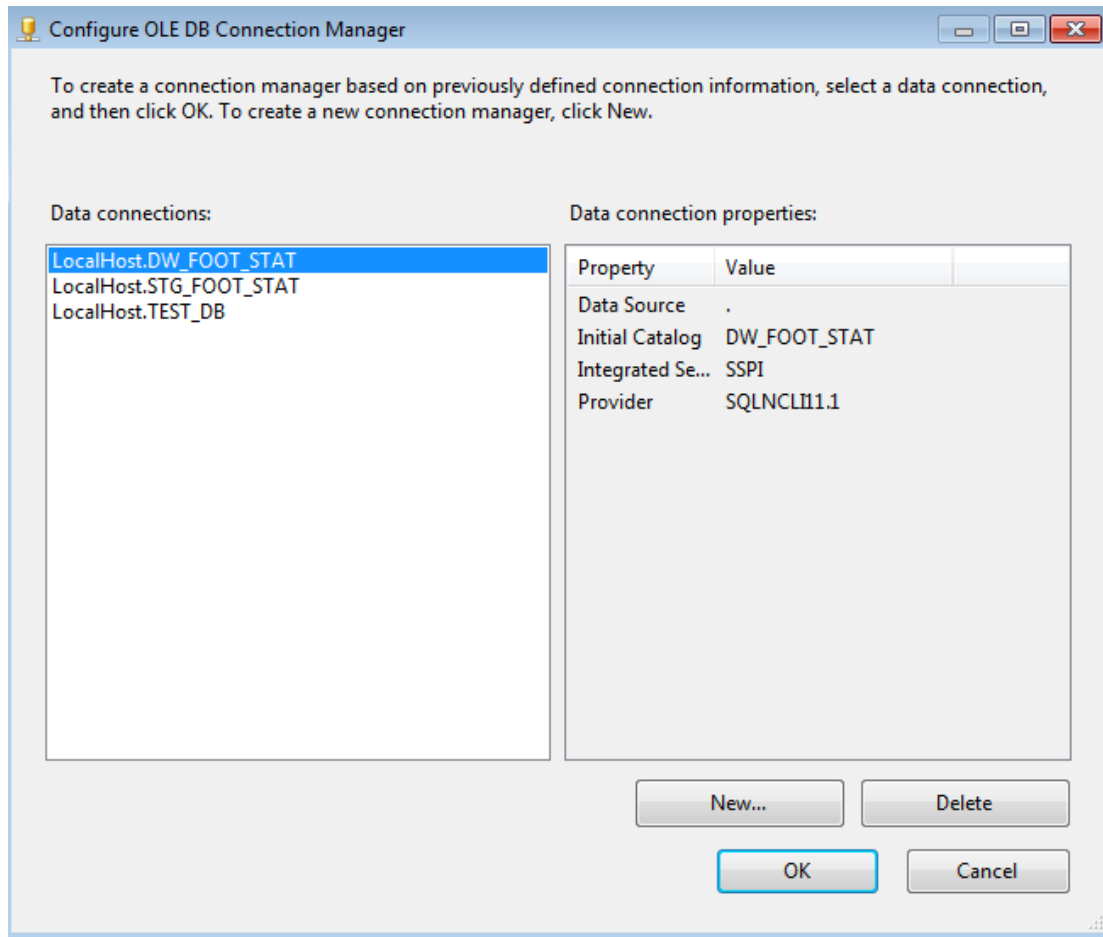


Figure 34: Previous Connections

Now let's start configuring the connection to our SQL Server database. Once the new connection wizard starts, you need to provide all the properties I have mentioned before to establish a connection. As you can see in the following figure, you can test the connection by clicking **Test Connection** at the bottom left after providing all the required fields.

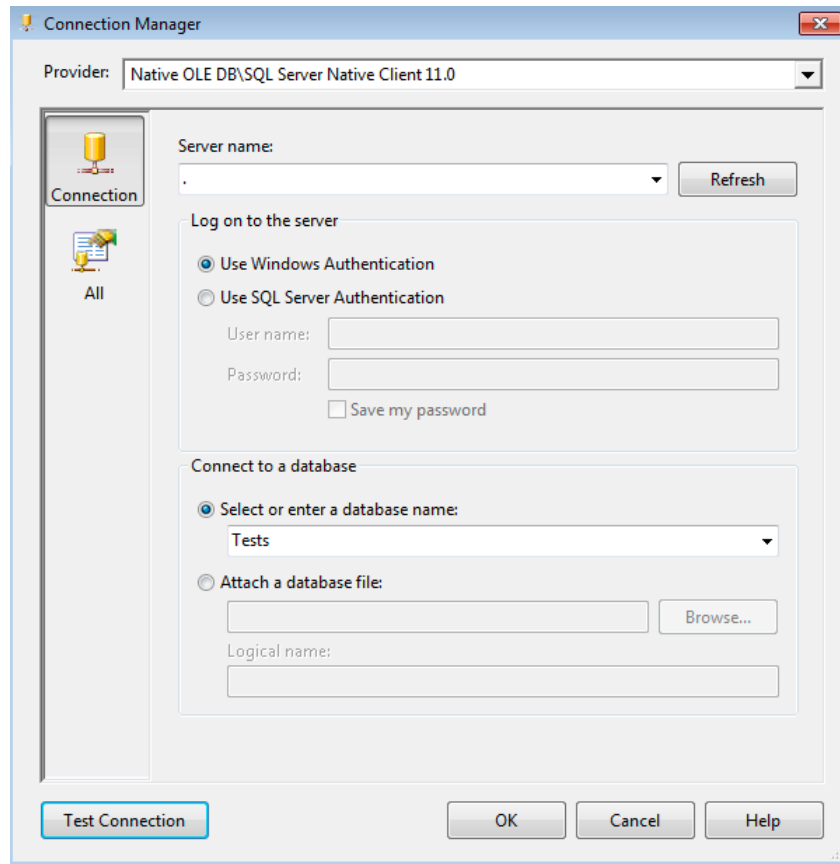


Figure 35: Configuring the Connection

After you create a successful connection, click **OK** in the current screen, and then click **OK** in the Connection Manager screen to complete the configuration. You will now be able to see your new connection in the Connection Managers folder inside the Solution Explorer.

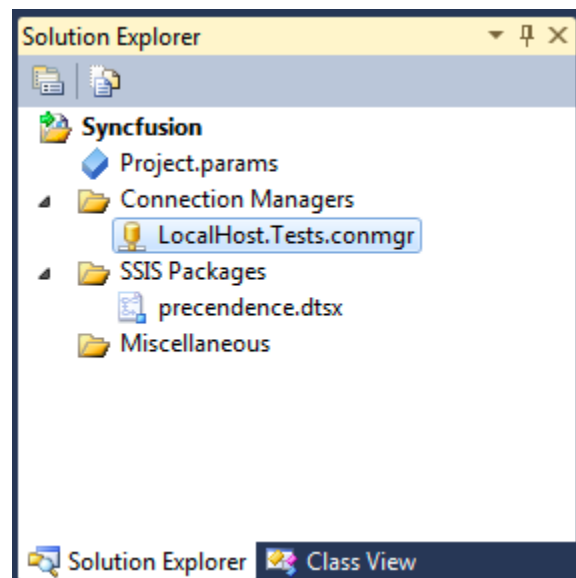


Figure 36: New Connection Created

Fix: Excel/Access Connection Manager Error When Running Package in 64-bit Environment

Although I haven't made a demo of the Excel or Access connection manager, I want to make a special note related to the use of these two connection managers as they are not compatible with 64-bit environments, but there is a way to fix this issue.

The problem only occurs in 64-bit environments because the Excel and Access connection manager isn't compatible with them. This can be a problem in integration projects; many have an initial loading stage in which we get data from several Excel files and integrate them in a SQL Server database.

For example, imagine that you have installed SQL Server 2012 with business intelligence features in a 64-bit environment; however, you want to use an Excel source inside your data flow.

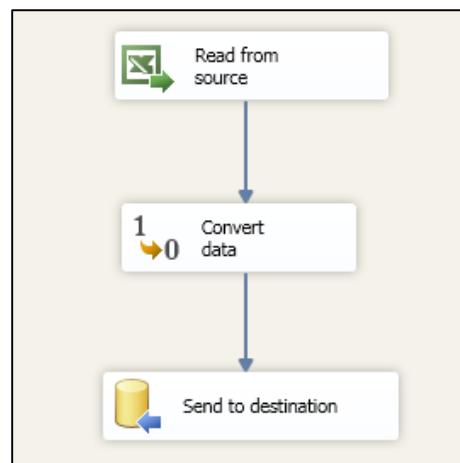


Figure 37: Simple Package

When you compile and run a package like this, SSIS will try to execute it in a 64-bit version by default. You will get the following error:

```
[Read from source [16]] Error: SSIS Error Code
DTS_E_CANNOTACQUIRECONNECTIONFROMCONNECTIONMANAGER. The AcquireConnection
method call to the connection manager "YOUR CONNECTION NAME" failed with
error code 0xC0209303. There may be error messages posted before this with
more information on why the AcquireConnection method call failed.
```

This error occurs because the SSIS engine will try to run the package in a 64-bit version, and the Excel Connection Manager is only compatible with 32-bit packages. The solution for this problem is very easy. Right-click your project inside the Solution Explorer and open the Properties of your package as shown in the following figure.

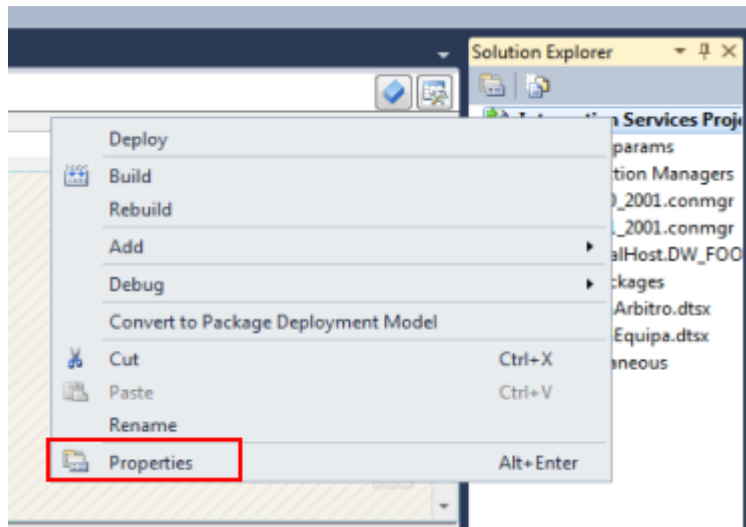


Figure 38: Opening the Properties window

Next, expand the **Configuration Properties** and open the **Debugging** options. Once they are open, change the **Run64BitRuntime** property to **False** to force the project package to run in 32 bits. The Configuration Properties should now look like the following figure.

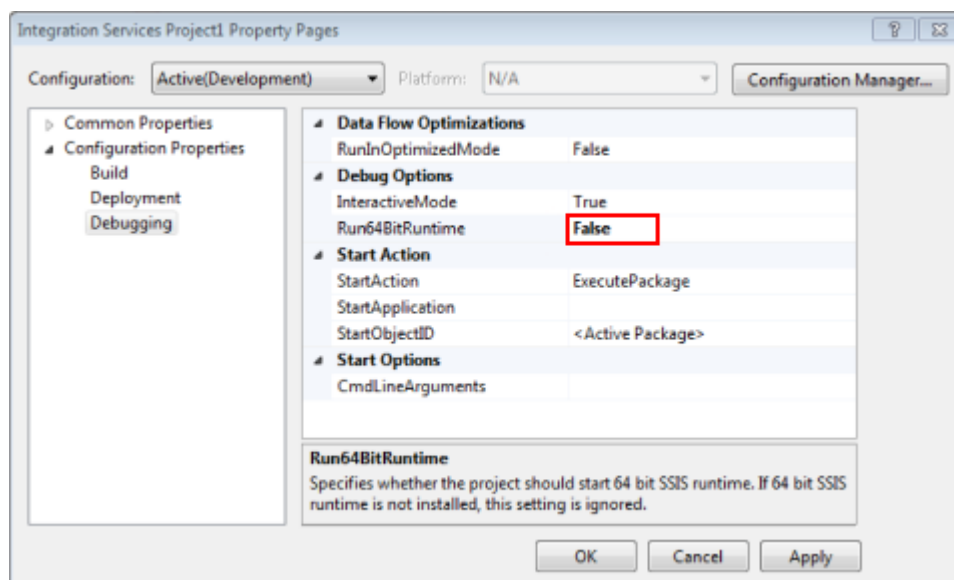


Figure 39: Changing properties to run 32-bit packages

Although this solution won't mess with any other connection manager inside your project, you must be aware that all your packages inside this project will now run in 32 bits.

Event Handlers

Introduction

Event handlers allow you to capture and handle the events that executing an object with Integration Services raises. After defining the event you want to capture, SSIS allows you to develop data flows to be executed when the event is detected. It's very important to be aware of the events you are able to capture in order to keep your packages' execution from failing and your ETL process from stopping.

To better manage the event handlers configuration inside a package, SSIS provides a specific tab inside the package designer. To access it, select the **Event Handler** tab as shown in the following figure.

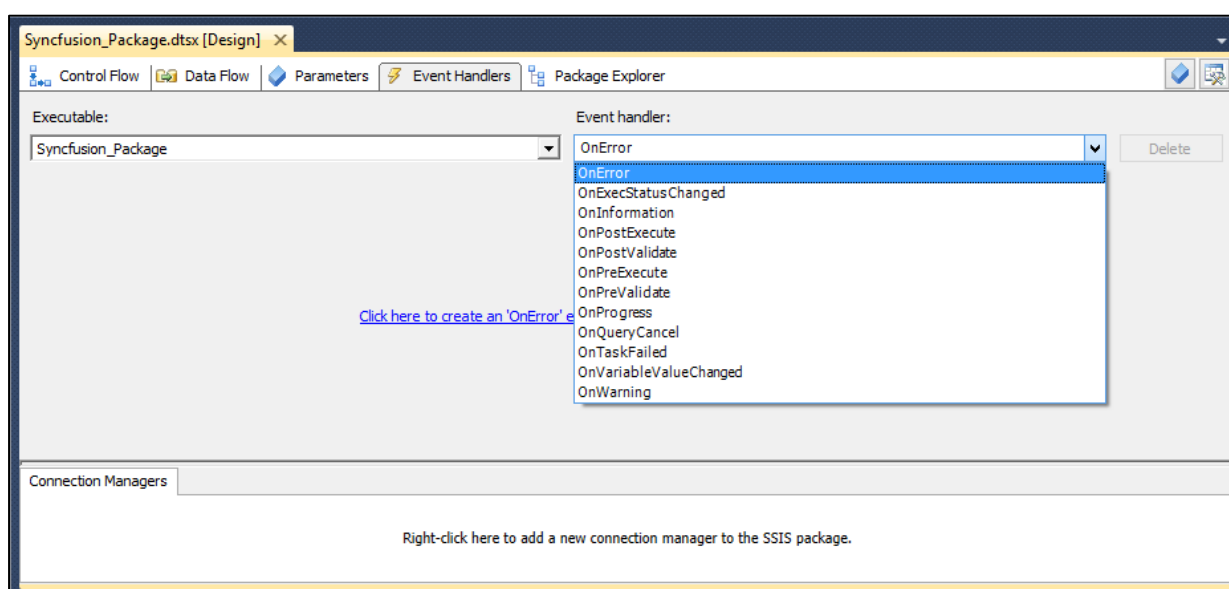


Figure 40: Event Handlers

As you can see in the previous figure, this Event Handlers tab has two drop-down menus in it. The first dropdown menu, Executable, allows you to select the task or container in the package for which the event handler is associated (you can also define it at the current package level). The second drop-down menu, Event Handler, allows you to select which event you want to capture.

Using Event Handlers

In this section, I will show how you can use event handlers while developing packages with Integration Services. The first step is to open the package in which you want the event handler to be configured. Once you open it, select the **Event Handlers** tab inside the package designer.

Inside the event handler window, select the executable you want the event handler to be added to in the left combo box, and select the event you want to add in the right combo box as shown in the following figure.

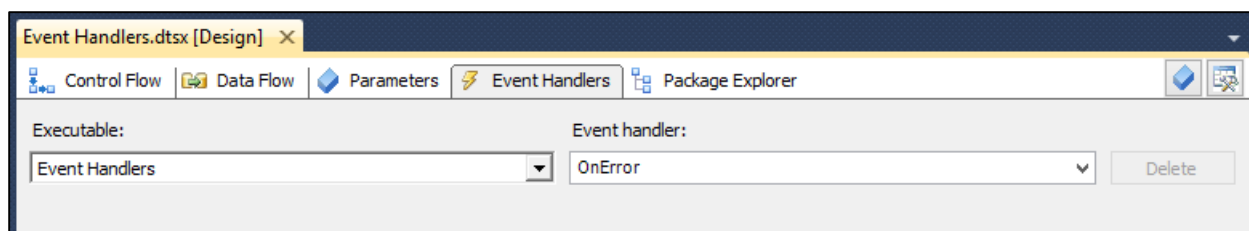


Figure 41: Creating an event handler

After you map the executable and the event handler, select the hyperlink in the bottom of the window as shown in the following figure. This will open a new event handler design window.



Figure 42: Opening the event handler designer

Inside the event handler designer you can design a normal control flow to orchestrate the execution flow when this error is caught. In this example, I send an email to our ETL developer and save the error into an XML file. After you make your developments, save the package and test it.

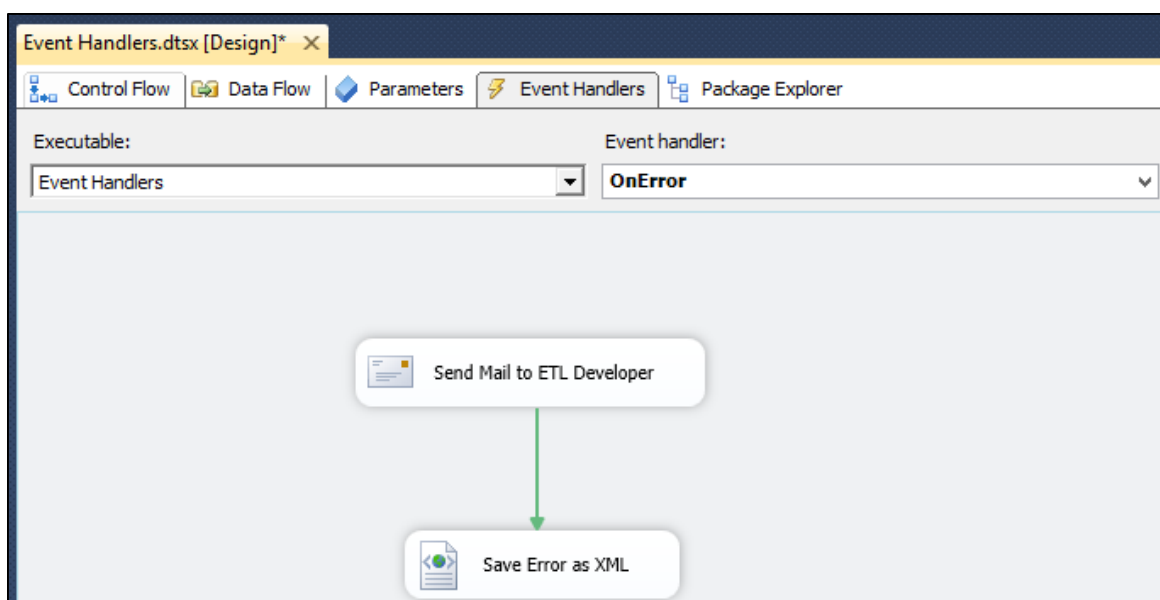


Figure 43: OnError Event Handler

Troubleshooting

Troubleshooting is important in every technology. You do it while working with programming languages as well as when working with databases. Integration Services isn't an exception. The first step to handle errors inside your packages' execution is to correctly use precedence constraints and event handlers.

However, these two mechanisms aren't enough to identify and solve problems in your packages' execution. Sometimes you need to place breakpoints or log the errors to predefined databases in order to evaluate the value of a variable, record values, and take action needed to fix it if the values recorded aren't the expected ones.

Breakpoints

Let's start by analyzing breakpoints. SSIS brings you its own breakpoints so that you can stop the package's execution at a particular step and evaluate the state of the variables and other components. They work like they do in any other programming language; if you have worked with Visual Studio for programming in any language you are familiar with the breakpoints mechanism. The main difference is that in those cases, you define them by code lines, while in SSIS you define them in tasks, containers, or packages. As you can see, they aren't available inside data flows.

To define breakpoints, you just need to right-click the component you want and select **Edit Breakpoints**. This will open the **Set Breakpoints** window where you will develop the breakpoints.

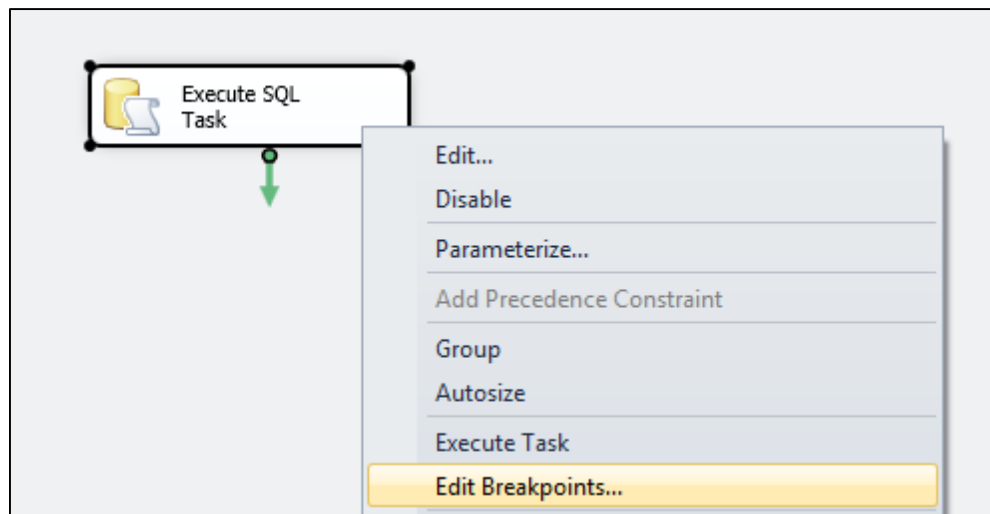


Figure 44: Edit Breakpoints

Once again, if you are familiar with breakpoints in programming language IDEs, in SSIS the concept is different. You don't click in the line you wish to stop the execution of, but rather, you enable break conditions which are based on events. The following figure shows what the breakpoint editor looks like.

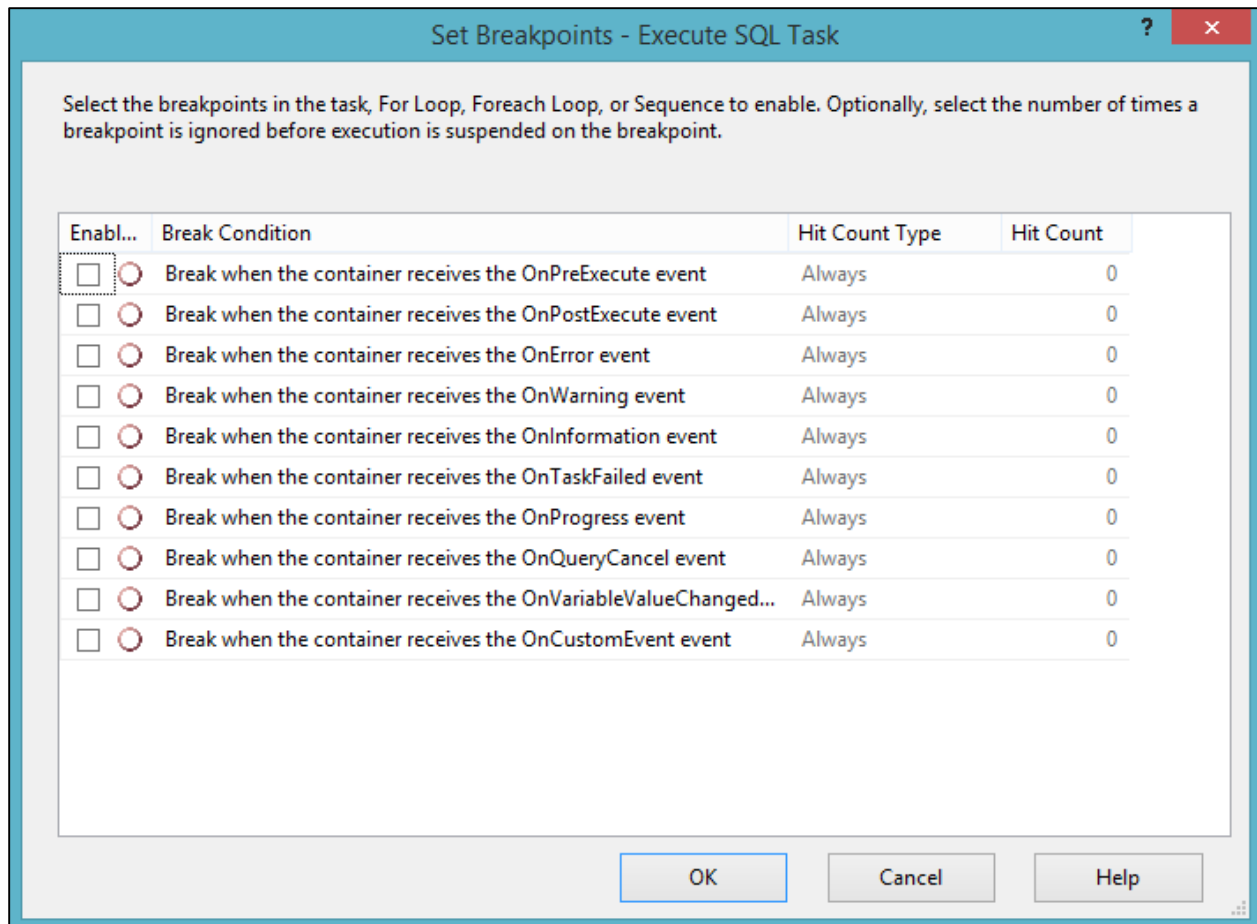


Figure 45: Breakpoints Editor

Using Breakpoints to Stop Before Executing a Task

In this example, I will show you how to use breakpoints to stop a package's execution before it executes a specific task. In this example, we will stop before the Execute SQL Task. The first step is opening the breakpoint editor and selecting the **OnPreExecute** event as shown in the following figure.

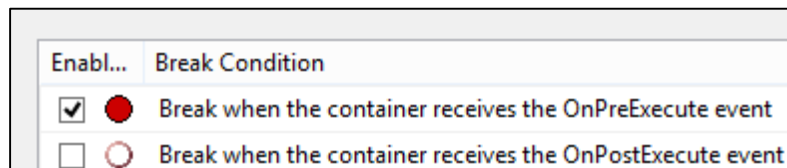


Figure 46: OnPreExecute Event

After adding the breakpoints, you want to save and close the breakpoint editor. Once you do, all executables to which you have added the breakpoints will be marked with a red dot as the following figure shows.

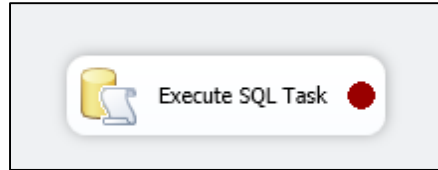


Figure 47: Task with Breakpoint

With these breakpoints defined, when you run your executables, the engine will stop at these breakpoints; you can then inspect variables and parameters at this stage of the execution.

Logging

Once again, as in any programming language or information system platform, logging is essential to understanding errors, execution times, and many more pieces of information without having to debug the solution. In SSIS, the logging mechanism is very easy to use and understand. By simply selecting a few check boxes and configuring some components, you can start logging your ETL process executions in XML or in a SQL Server table. This is because it has a built-in set of features that captures details about package execution.

You have several options for output objects when logging your information. You can generate text files, XML files, log to SQL Server databases, and even generate a Windows Event. To show you how simple it is to activate logging, in the next section I will create a demo that will create XML files with your package execution logs.

Activate Logging in Your Packages

An important thing to note is that SSIS logging activation is not needed if the project will be stored in an SSISDB catalog. To activate the logging mechanism in your packages, click **SSIS** in the toolbar and select **Logging**. This will open the log configuration editor.

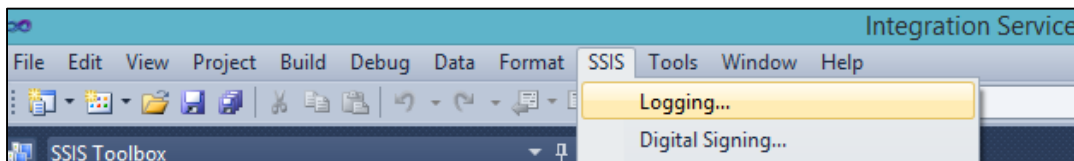


Figure 48: Opening the Logging menu

After the editor opens, you need to specify in the **Containers** pane on the left which package you want the logging to be activated in. In this example, I enabled logging for the event handlers of the three tasks I have in the control flow. After this, we need to choose which provider we want to store the logs in. In this example, I am saving the logs into a file. Selecting the provider is as simple as choosing one from the **Provider type** combo box and clicking **Add**.

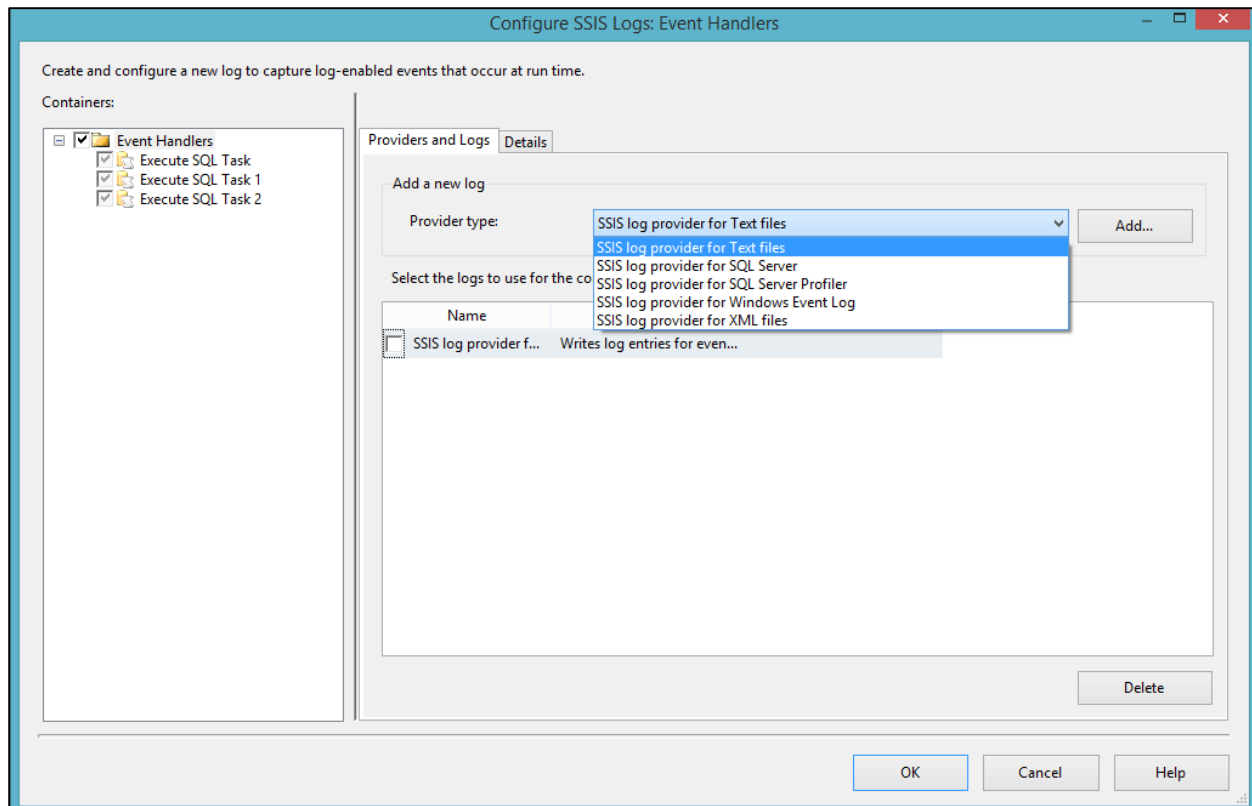


Figure 49: Logging Configuration Editor

The next step to configure the logging in your selected executables is to define the provider into which you want the logs to be stored. In this case, we have chosen a text provider, and we need to configure a new file to connect to, which tells the provider where it should store the logs. Don't forget to enable the provider by using the check box on the left. This allows you to enable and disable your providers easily.

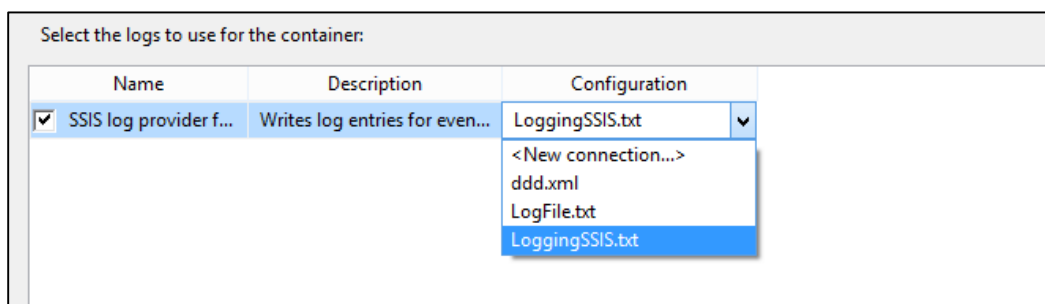


Figure 50: Configuring a Provider

As I've mentioned before, we need to define the file that will store the logs. The following figure shows the file connection properties window where we will define it.

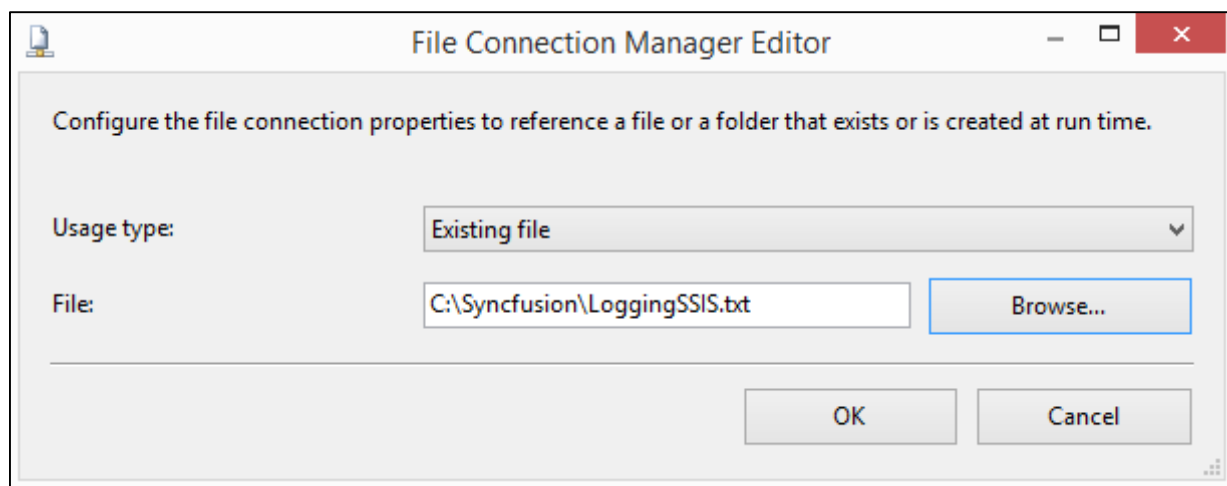


Figure 51: File Connection Manager Editor

Last but not least, we need to enable the events we want to associate this log configuration with. To do so, click in the **Details** tab and enable the ones you want. In this case, I've enabled all of them.

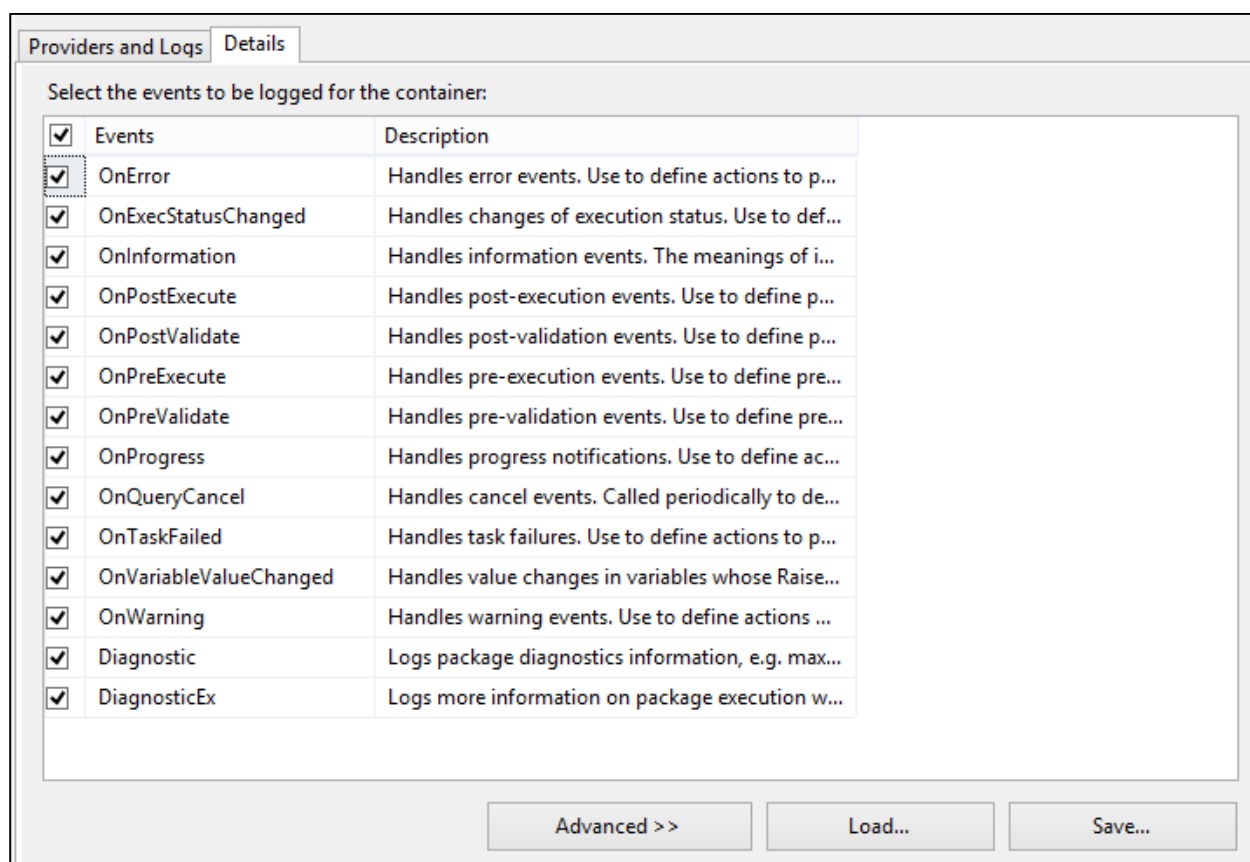


Figure 52: Enabling Event Logging

After saving and closing the windows, you can now test this log mechanism. To do this, run the package and open the logging file. If everything goes all right, your file should look like the following figure.

```
#Fields: event,computer,operator,source,sourceid,executionid,starttime,endtime,datacode,databytes,message
OnPreValidate,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{172DD0B4-5054-4D61-83D6-584
OnPreValidate,H,H\Syncfusion,Execute SQL Task 1,{a0f87106-7dcd-457b-91d3-87f408895630},{172DD0B4-5054-4D61-83D6-584
OnPostValidate,H,H\Syncfusion,Execute SQL Task 1,{a0f87106-7dcd-457b-91d3-87f408895630},{172DD0B4-5054-4D61-83D6-584
OnPreValidate,H,H\Syncfusion,Execute SQL Task 2,{f3b0a7aa-9d54-42dd-9627-183726a37d4b},{172DD0B4-5054-4D61-83D6-584
OnPostValidate,H,H\Syncfusion,Execute SQL Task 2,{f3b0a7aa-9d54-42dd-9627-183726a37d4b},{172DD0B4-5054-4D61-83D6-584
OnPostValidate,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{172DD0B4-5054-4D61-83D6-584
OnPreValidate,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{BDDE4A11-E273-4D6F-ABBC-281F
OnPreValidate,H,H\Syncfusion,Execute SQL Task 1,{a0f87106-7dcd-457b-91d3-87f408895630},{BDDE4A11-E273-4D6F-ABBC-281F
OnPostValidate,H,H\Syncfusion,Execute SQL Task 1,{a0f87106-7dcd-457b-91d3-87f408895630},{BDDE4A11-E273-4D6F-ABBC-281F
OnPreValidate,H,H\Syncfusion,Execute SQL Task 2,{f3b0a7aa-9d54-42dd-9627-183726a37d4b},{BDDE4A11-E273-4D6F-ABBC-281F
OnPostValidate,H,H\Syncfusion,Execute SQL Task 2,{f3b0a7aa-9d54-42dd-9627-183726a37d4b},{BDDE4A11-E273-4D6F-ABBC-281F
OnPostValidate,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{BDDE4A11-E273-4D6F-ABBC-281F
PackageStart,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{BDDE4A11-E273-4D6F-ABBC-281F
Diagnostic,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{BDDE4A11-E273-4D6F-ABBC-281F
DiagnosticEx,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{BDDE4A11-E273-4D6F-ABBC-281F
OnPreExecute,H,H\Syncfusion,Event Handlers,{4D5BCA91-7EC2-4C5A-AECB-F5C81E5A1C56},{BDDE4A11-E273-4D6F-ABBC-281F
OnPreExecute,H,H\Syncfusion,Execute SQL Task 1,{a0f87106-7dcd-457b-91d3-87f408895630},{BDDE4A11-E273-4D6F-ABBC-281F
OnPreValidate,H,H\Syncfusion,Execute SQL Task 1,{a0f87106-7dcd-457b-91d3-87f408895630},{BDDE4A11-E273-4D6F-ABBC-281F
```

Figure 53: Logging Results

Creating a Simple Control Flow

To use a control flow task, start by dragging and dropping it to the design window. For this example, add a new **Execute SQL Task** as shown in the following figure.

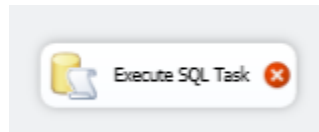


Figure 54: Execute SQL Task

Next, rename this task "Create Staging Table" and then double-click it to open the configuration window. The configuration window for this test task type is shown in the following figure.

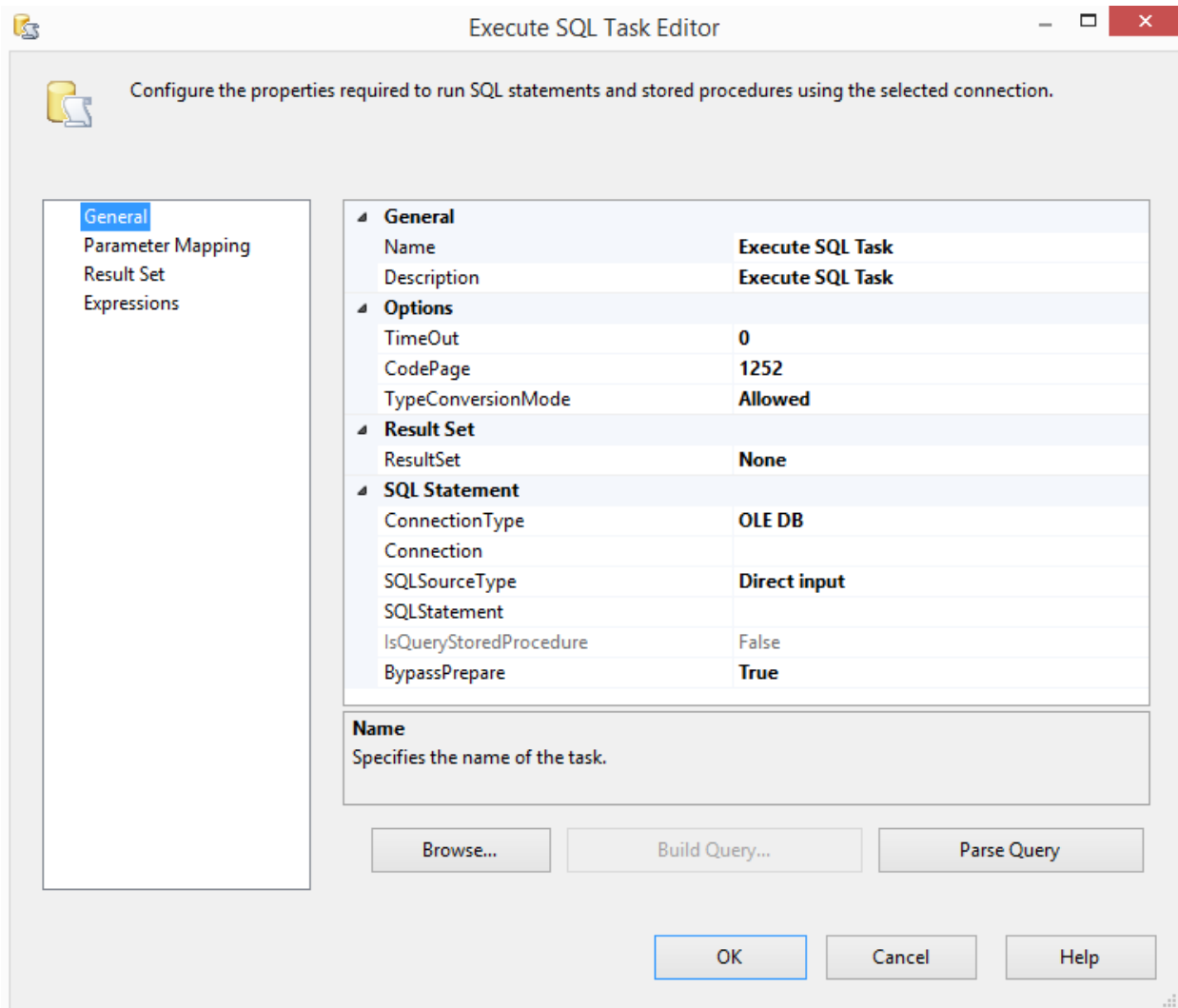


Figure 55: Execute SQL Task Editor

To make this task work, you just need to configure two properties: the connection and the SQL statement. In the connection you need to configure the SQL database in which this SQL statement will be executed. This statement is a DDL, DML, or DCL statement.

In this example, our control flow will be creating a new staging table using the execute SQL task we just configured, and then executing a simple data flow task. So, to create our staging table, we set the SQL statement in the task as in the following code sample.

```
CREATE TABLE [dbo].[STG_FOOT_STAT_TB](
    [Season] [varchar](9) NOT NULL,
    [Date] [datetime] NULL,
    [HomeTeam] [nvarchar](255) NULL,
    [AwayTeam] [nvarchar](255) NULL,
    [FTHG] [float] NULL,
    [FTAG] [float] NULL,
```

```

[FTR] [nvarchar](255) NULL,
[Referee] [nvarchar](255) NULL,
[HS] [float] NULL,
[AS] [float] NULL,
[HC] [float] NULL,
[AC] [float] NULL,
[HF] [float] NULL,
[HO] [float] NULL,
[AO] [float] NULL,
[HY] [float] NULL,
[AY] [float] NULL,
[HR] [float] NULL,
[AR] [float] NULL
) ON [PRIMARY]

```

The next step is to add the data flow task to the package control flow and connect the previous execute SQL task to it using precedence constraints. When adding the data flow task, you don't configure it; instead, when you double-click it, SSIS will take you to the data flow designer so that you can develop it according to your requirements. The final simple control flow is shown in the following figure.

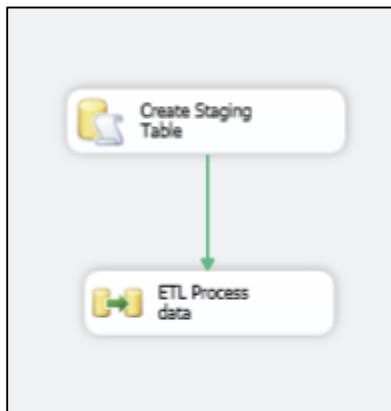


Figure 56: Simple Control Flow



Note: This is just a simple example of how you can create control flows. In your own development, you need to know and understand all existing tasks and know how to combine them so that your project requirements are fulfilled. The data flow development will be detailed in the following chapter.

Chapter 4 Data Flow

Introduction

Data flow is where all data extraction, transformation, mapping, and loading happens. Before starting to develop your data flow logic, you need to tell the control flow what you are going to do with it. So your first job is to add a data flow task to the control flow.

As I have previously explained, one package can only have one control flow; however, it can have as many data flows as you want. However, in the SSIS Designer, there is only one Data Flow tab. This is because it is a dynamic tab; whenever you click on a data flow task, this tab will assume the objects developed in it. This way, if you add a data flow task to your control flow, you can double-click it or select the Data Flow tab to open the data flow designer.

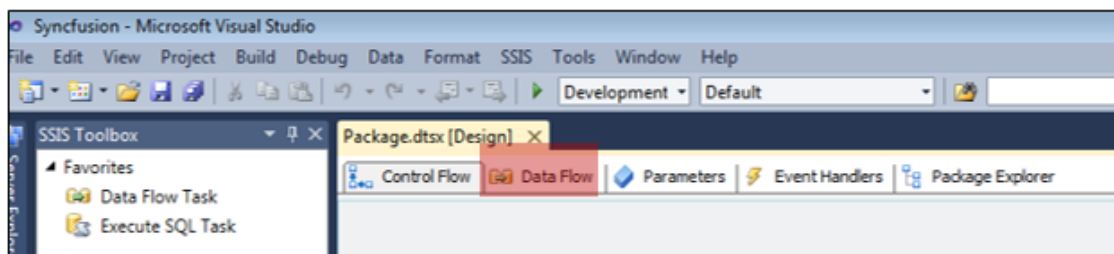


Figure 57: Opening the data flow designer

Data flow objects can be of four different types: sources, transformations, destinations, and paths. An interesting fact about data flows is that when you retrieve data from a source, the transformations, destinations, and paths that come after the initial source object will be executed in memory, which makes it a powerful engine.

Now, let's take a look at what these objects do. Sources are connections from which you will retrieve your data. Destinations are connections to the targets in which you want to insert or update data. In a perfect world, the schemas of source and destination would be the same but, as you know, they are not. Because of this, transformations are important because they allow you to transform the data to follow the destination schema. The transformations allow you to perform operations against your data sets such as column conversions, column creation, and many more that will be explained in the sections that follow.

Last but not least, we need to connect all these objects. For that we use paths, which are similar to precedence constraints but with fewer configuration options. However, they exist for the same purpose: to connect objects inside the data flow and define when to take flow (Failed, OnMatch, and NonMatch). The constraints of paths will be explained later.

When you connect two objects, the resulting behavior is a little bit different from the control flow. This is because, in this case, the connection doesn't define the conditions for the next task to be executed. Rather, it defines where the data will flow next when the execution of a transformation fails, when it succeeds, or some other option. This means that, when you make a connection, the information is about when or where the columns available to work in this data flow become available in the next component of the data flow path. Like the control flows, these objects are

designed using a drag-and-drop designer and then double-clicking them to open their configuration options. Now let's take a look at all available objects.

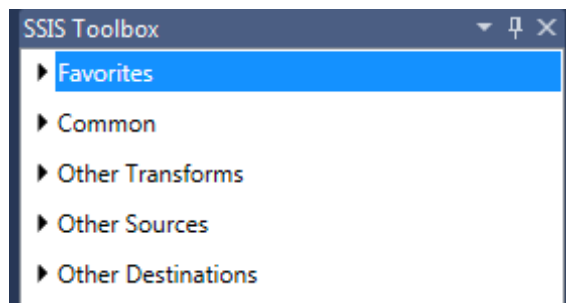




Figure 58: SSIS Toolbox with data flow objects

The first step when developing data flows is to define the source of your data. You can have as many sources as you need, and the SSIS data flow designer allows you to later join or merge that data (for instance, to insert it into a single destination). All of the objects available in the data flow toolbox are organized by their groups and explained in the sections that follow.

Favorites

The Favorites group contains two of the most important objects; they allow you to connect to external data sources or destinations. You can add new objects to this group by right-clicking an object and selecting **Move to Favorites**.








Table 5: Data flow transformations in the Favorites group








Shape	Name	Description
	Destination Assistant	This object allows you to configure data sending operations to a variety of destination platforms. Its wizard helps you through the configuration of the target system.
	Source Assistant	This object allows you to configure data retrieval operations from a variety of source platforms. Its wizard helps you through the configuration of the target system.

Common

The Common tasks group includes objects commonly used in data flows. Once again, you can move objects to this group by right-clicking them and selecting **Move to Common**. The following table explains each of the objects that you can find in it.

Table 6: Common data flow transforms









Shape	Name	Description
	Aggregate	This object allows you to aggregate data according to the following functions: Average, Sum, Count, Count Distinct, Max, and Min.
	Conditional Split	This object allows you to route data to multiple outputs based on conditions that can be specified within it. You can use conditions to specify the outputs of this object.
	Data Conversion	This object allows you to convert data between data types. You can use it to convert, for instance, an integer into a string.
	Derived Column	This object allows you to create new columns based on transformations applied to input columns. For example, you can create a single column based on the concatenation of the values of the other three input columns.
	Lookup	This object allows you to join additional columns to the data flow by looking up values in a table. This is commonly used, for example, to evaluate if a record always exists in a data warehouse table.
	Merge	This object allows you to combine rows from multiple, sorted data flows into one sorted data flow. If sorting is not important, use the Union All transformation object.
	Merge Join	This object allows you to combine two sorted data flows into one using FULL, LEFT, or INNER join.










Shape	Name	Description
	Multicast	This object allows you to distribute every input row to every row in one or more outputs.
	OLE DB Command	This object allows you to run SQL statements over each row in the data flow. This is different from the Execute SQL Task in the control flow because this affects each row; the task is applied to the database—commonly pre-DDL or DML operations.
	Row Count	This object allows you to count the number of rows in a data flow.
	Script Component	This object allows you to run custom script code. You can use it as a source, destination, or transformation object. You can use the C# and VB.NET programming languages.
	Slowly Changing Dimensions (SCD)	This object allows you to update an SSIS data warehouse dimension table. This object is fully detailed in the Chapter 5 .
	Sort	This object allows you to sort data in ascending or descending order. It is commonly used to remove duplicated values as well because, when using this object, you can tell it to remove duplicates after sorting them.
	Union All	This object allows you to combine multiple data flows without sorting. You can use it to combine data coming from multiple objects' outputs.

Other Transforms

The Other Transforms group includes objects less frequently used in data flows. Although they are not used as often, they can be equally as important to your ETL projects.

Table 7: Other Transforms









Shape	Name	Description
	Audit	This object allows you to include data about the environment in which the package runs. Map system variables to new output columns.
	Cache Transform	This object allows you to write data to a cache (a .caw file) with the Cache connection manager.
	CDC Splitter	This object allows you to direct streams of net change records into different outputs based on the type of change (insert, delete, and update).
	Character Map	This object allows you to apply string functions to character data. For example, convert uppercase to lowercase.
	Copy Column	This object allows you to copy input columns to new columns in the transformation output.
	Data Mining Query	This object allows you to query data mining objects from Analysis Services.
	DQS Cleansing	This object allows you to use Data Quality Services (DQS) to correct data from a connected data source by applying approved rules that were created for the connected data source or a similar data source.
	Export Columns	This object allows you to read data from a data flow and insert it into a file.

Shape	Name	Description
	Fuzzy Grouping	This object allows you to identify potential duplicate rows, and helps standardize the data by selecting canonical replacements.
	Fuzzy Lookup	This object allows you to perform data cleaning tasks such as standardizing data, correcting data, and providing missing values. Uses fuzzy matching to return one or more close matches from a reference table.
	Import Column	This object allows you to read data from files and adds it to columns in a data flow.
	Percentage Sampling	This object allows you to randomly sample a percentage of rows from an input data flow.
	Pivot	This object allows you to compact an input data flow by pivoting it on a column value, making it less normalized.
	Row Sampling	This object allows you to randomly sample a specific number of rows from an input data flow.
	Term Extraction	This object allows you to extract frequently used, English-only terms from an input data flow.
	Term Lookup	This object allows you to determine how frequently specific terms occur in a data flow.
	Unpivot	Converts an unnormalized data set into a more normalized version. Values from multiple columns of a single record expand to multiple records in a single column.

Other Sources

The Other Sources group includes other source objects used in data flows.










Table 8: Other Sources





Shape	Name	Description
	ADO.NET Source	This object allows you to consume data from SQL Server, OLE DB, ODBC, or Oracle using .NET providers.
	CDC Source	This object allows you to read changed data from SQL Server CDC shadow tables. This object is fully detailed in Chapter 5 .
	Excel Source	This object allows you to connect and extract data from worksheets or named ranges in Microsoft Excel workbooks.
	Flat File Source	This object allows you to read and extract data from text files. You can define data widths or data delimiters to identify rows and columns. This objects is used for CSV files, for example.
	ODBC Source	This object allows you to extract data from an Open Database Connectivity database. You can use it to extract data from tables or views.
	OLE DB Source	This object allows you to extract data from an OLE DB relational database. You can use it to extract from tables or views.
	Raw File Source	This object allows you to extract raw data from flat files previously written by the raw file destination. Reading local files uses this to optimize packages.
	XML Source	This object allows you to extract data from XML files.

Other Destinations

The Other Destinations group includes other used destination objects in data flows.

Table 9: Other Destinations

Shape	Name	Description
	ADO.NET Destination	This object allows you to load data into an ADO.NET-compliant database that uses a database table or view.
	Data Mining Model Training	This object allows you to train data mining models by passing the data that the destination receives through the data mining model's algorithms.
	Data Reader Destination	This object allows you to expose data in a data flow to other applications by using the ADO.NET Data Reader interface.
	Dimension Processing	This object allows you to load and process a SQL Server Analysis Services dimension.
	Excel Destination	This object allows you to load data into worksheets or named ranges in Microsoft Excel workbooks.
	Flat File Destination	This object allows you to write to a text file.
	ODBC Destination	This object allows you to load data into an Open Database Connectivity (ODBC)-compliant database.
	OLE DB Destination	This object allows you to load data into an OLE DB-compliant relational database such as SQL Server.
	Partition Processing	This object allows you to load and process a SQL Server Analysis Services partition.

Shape	Name	Description
	Raw File Destination	This object allows you to write raw data that will not require parsing or translation.
	Record Set Destination	This object allows you to create and populate an in-memory ADO record set that is available outside of the data flow. Scripts and other package elements can use the record set.
	SQL Server Compact Destination	This object allows you to write data to a table in a SQL Server Compact database.
	SQL Server Destination	This object allows you to connect to a local SQL Server database and bulk loads data into SQL Server tables and views. To optimize performance, it is recommend that you use the OLE DB destination instead.

Data Viewers

Introduction

Data viewers are great debug components because they allow you to see the data that is flowing from one data flow component to another. As I wrote before, paths connect data flow components by connecting the output of one data flow component to the input of another data flow component. Data viewers operate in these paths, allowing you to intercept the “communication” and watch what input will be used in the next component or in another point of view (with the data set that the previous component originated). When you run the package, it will stop that data viewer and only continue its execution when you tell it to. By using data viewers, you can evaluate whether or not the result of a data flow component meets your requirements. If not, it allows you to correct it.

An important but logical prerequisite of using data viewers is that you must have at least one data flow in your package control flow and, inside the data flow, at least two components connected.

Using Data Viewers

To use a data viewer, right-click the path you want to evaluate and then select **Enable Data Viewer** as shown in the following figure.

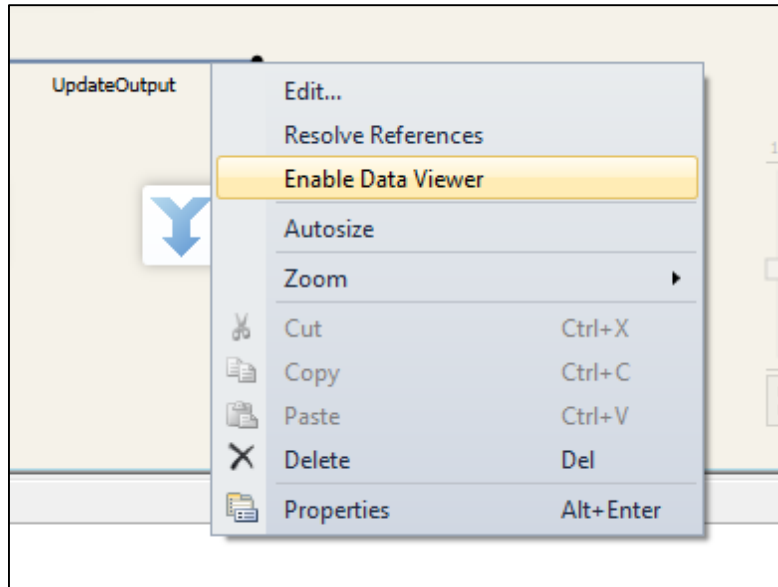


Figure 59: Enable Data Viewer

And that's it. If you run a package with a data viewer in a data flow, the result will look like the following figure—a window showing the data being moved in that particular path.

OLE DB Source Output Data Viewer at Data Flow Task							
<div> <div>Detach</div> <div>Copy Data</div> </div>							
Season	Date	Home Team	Away Team	FTHG	FTAG	FTR	Referee
2007/...	2008-0...	Birmingham	Arsenal	2	2	D	M Dean
2007/...	2008-0...	Fulham	West Ham	0	1	A	H Webb
2007/...	2008-0...	Liverpool	Middlesbrough	3	2	H	L Mason
2007/...	2008-0...	Newcastle	MANCHESTER UNITED	1	5	A	C Foy
2007/...	2008-0...	Portsmouth	Sunderland	1	0	H	P Dowd
2007/...	2008-0...	Wigan	Derby	2	0	H	A Wiley

Figure 60: Data Viewer Result

Creating a Simple Data Flow

Introduction

To show you how these components work, we will create a simple data flow that gets a data set from a SQL Server database, transforms the data according to the schema in the destination SQL Server table, and then inserts it. It's a simple data flow that will allow us to work with these objects for the first time. The components we will use are:

- Source
- Conversion
- Derived Column
- Destination

Creating the Data Flow

To create this example data flow, we start by dragging and dropping each object we need into the data flow designer and, after linking them, making all the configurations needed. So let's start by defining the flow.

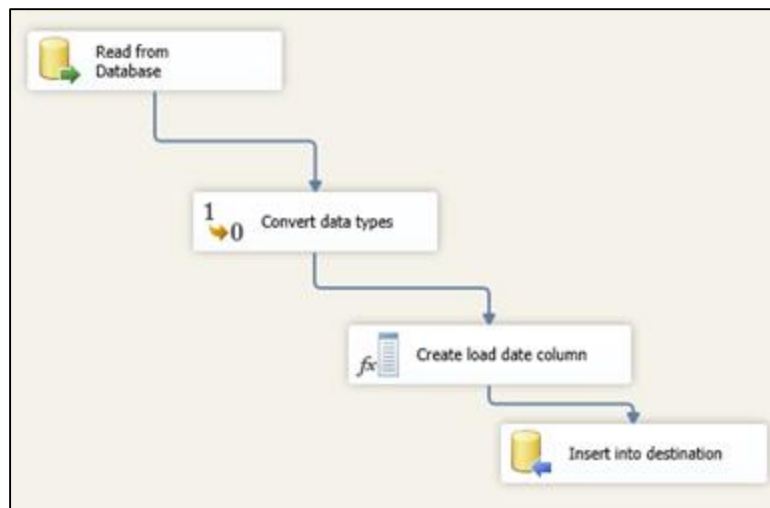


Figure 61: Simple Data Flow

Now, the way you develop a data flow is always the same: add the objects you need to the data flow designer, connect them using tasks, and then configure their behavior. Because we have already done the first two steps, it's now time to configure each of the objects in the data flow. The first object to configure is the source object. In this object, you need to define the connection manager for the source database (create a new one if none exists) and the table or view name.

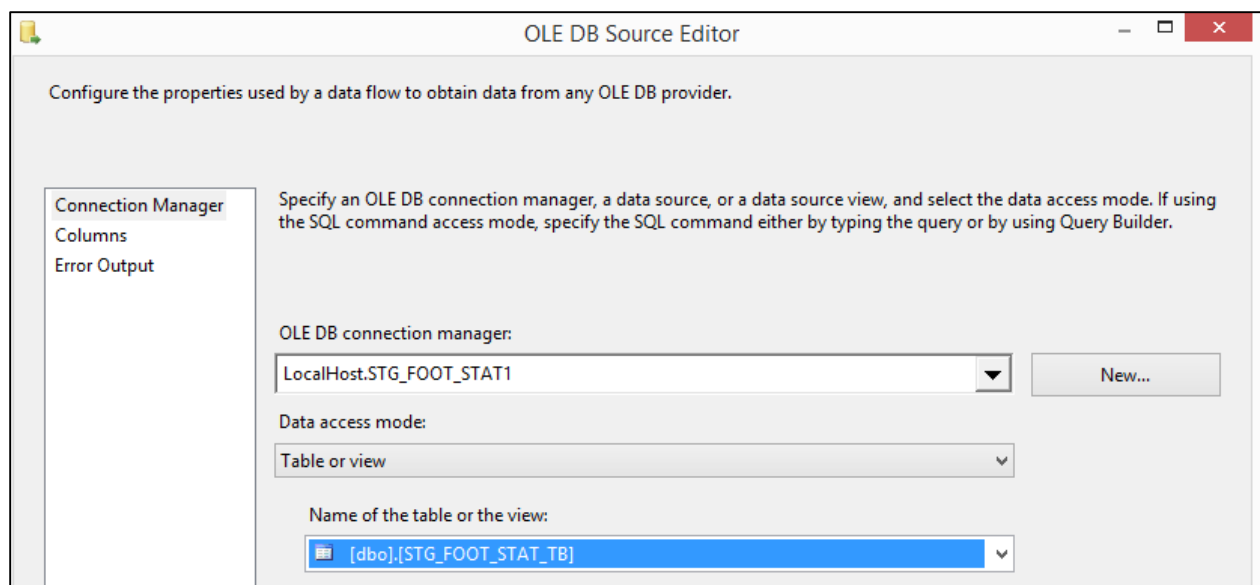


Figure 62: Source Configuration

After configuring the source object, click **OK** to close the editor and double-click the **Conversion** object. This will open its configuration screen where you can create expressions to transform the columns as needed. In this screen, select the columns from the source table or view you want to convert. In the conversion grid, select the destination data type and size.

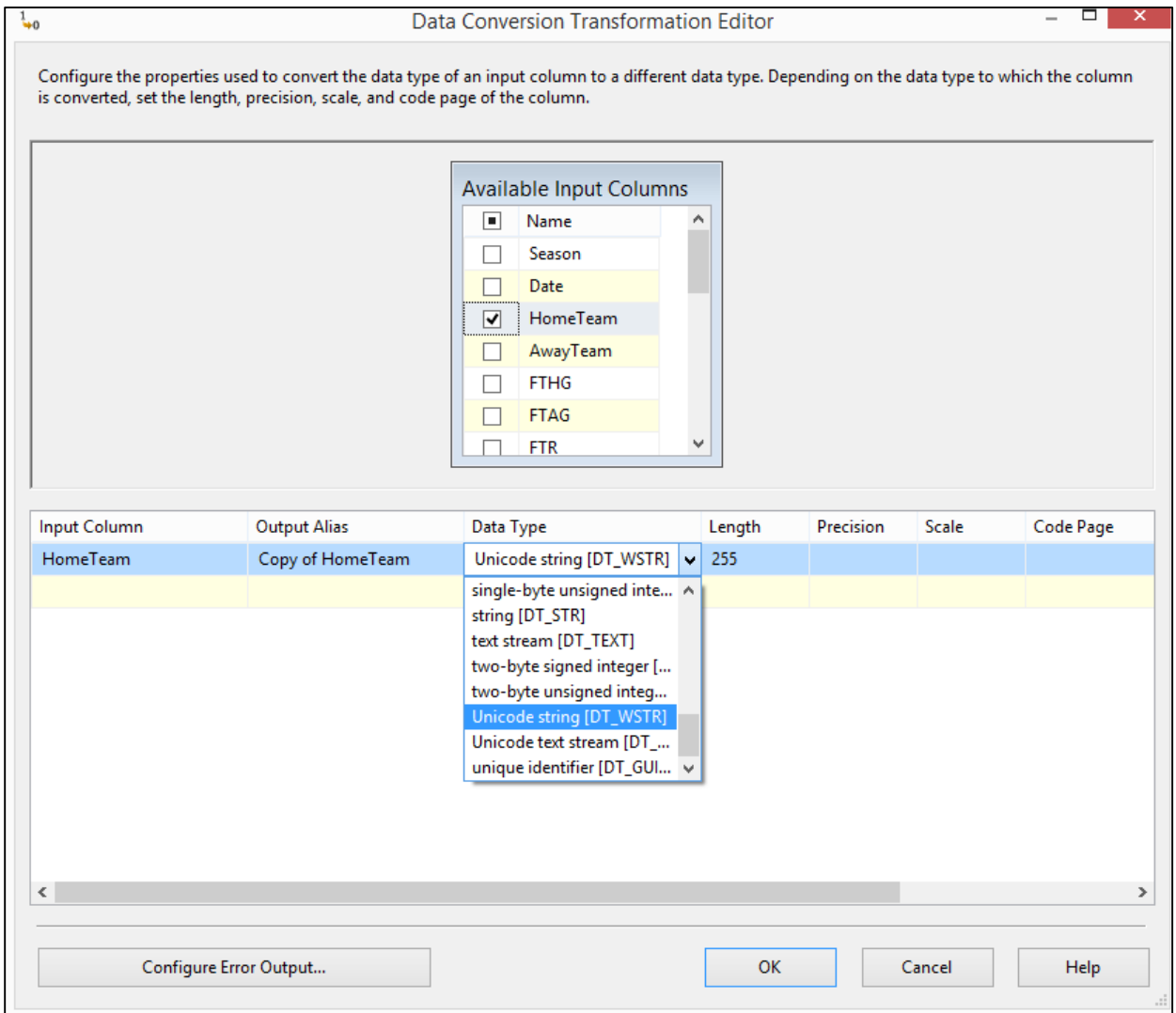


Figure 63: Conversion Configuration

Now we want to create a new column and add it to the existing data flow columns. This will be a load date value which indicates when this record was inserted. To do so, double-click the **Derived Column Transformation**. When the editor opens, you need to give this column a name. In the derive column option, select **Add a new column** and, in the expression, use the GETDATE() function to get the current date and time.

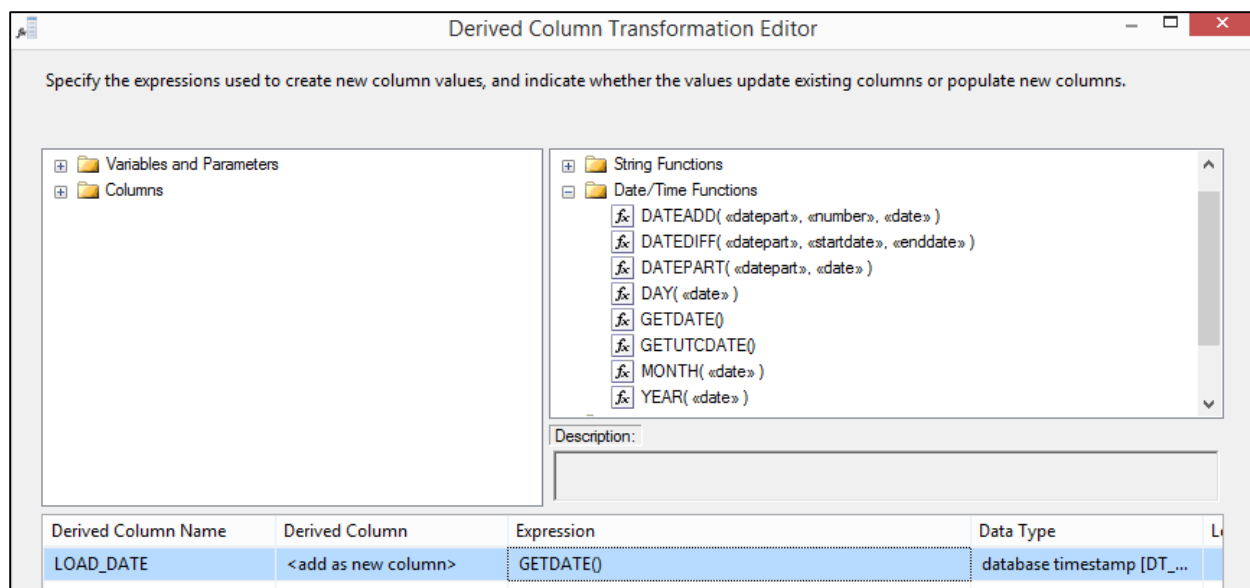


Figure 64: Creating a new derived column

Last but not least, we need to configure the destination object. If you double-click the destination object, you will notice a big difference between it and the source object. You will see a new tab on the left called Mappings. In this tab, you are going to map the columns from the data flow which have been transformed or created to the columns in the destination table. However, the first step is configuring the connection manager.

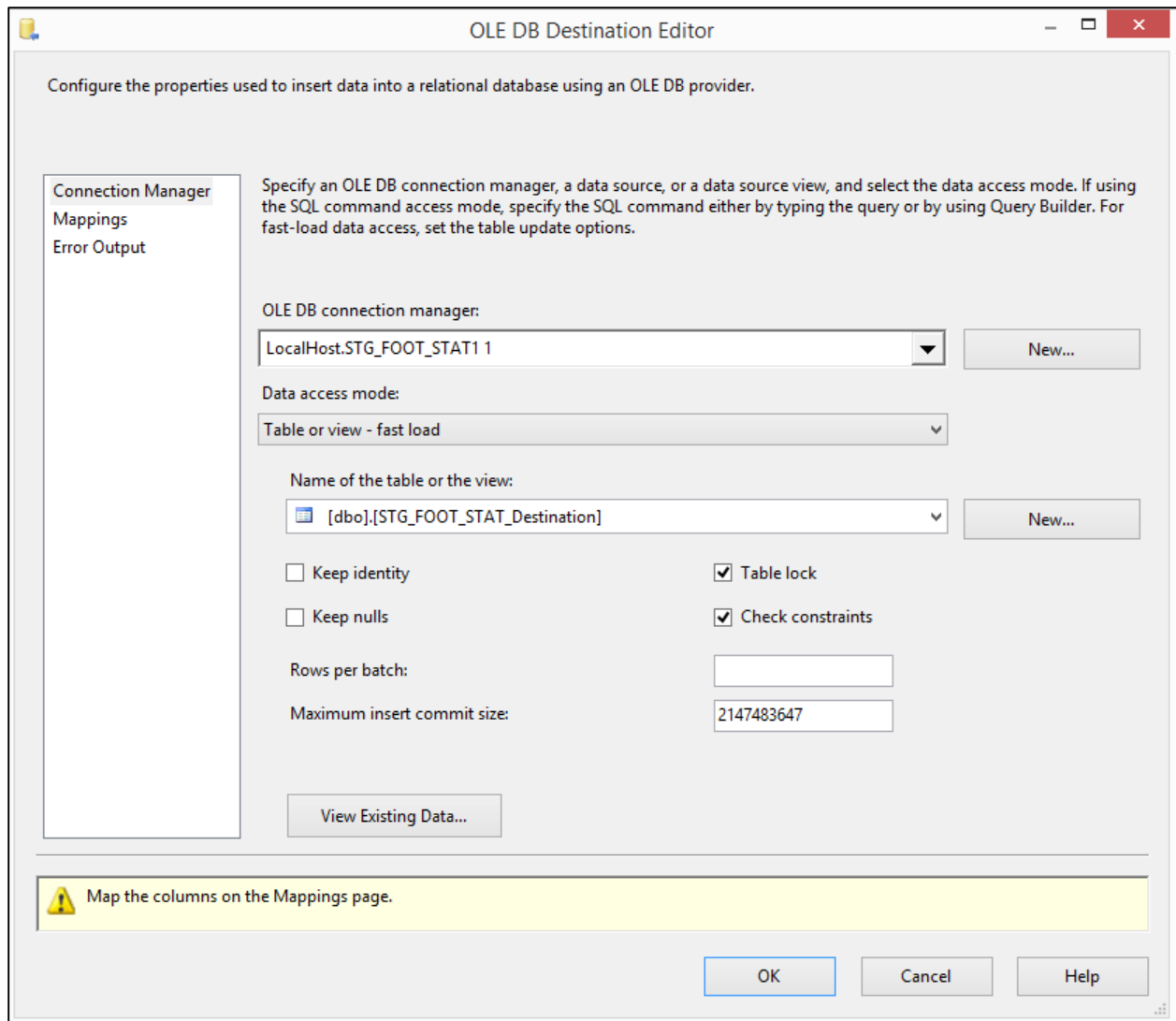


Figure 65: Destination object connection manager

As you can see in the previous figure, there is a warning at the bottom of the editor that tells you that you need to map the columns from the data flow to the destination table. So let's click the **Mappings** tab.

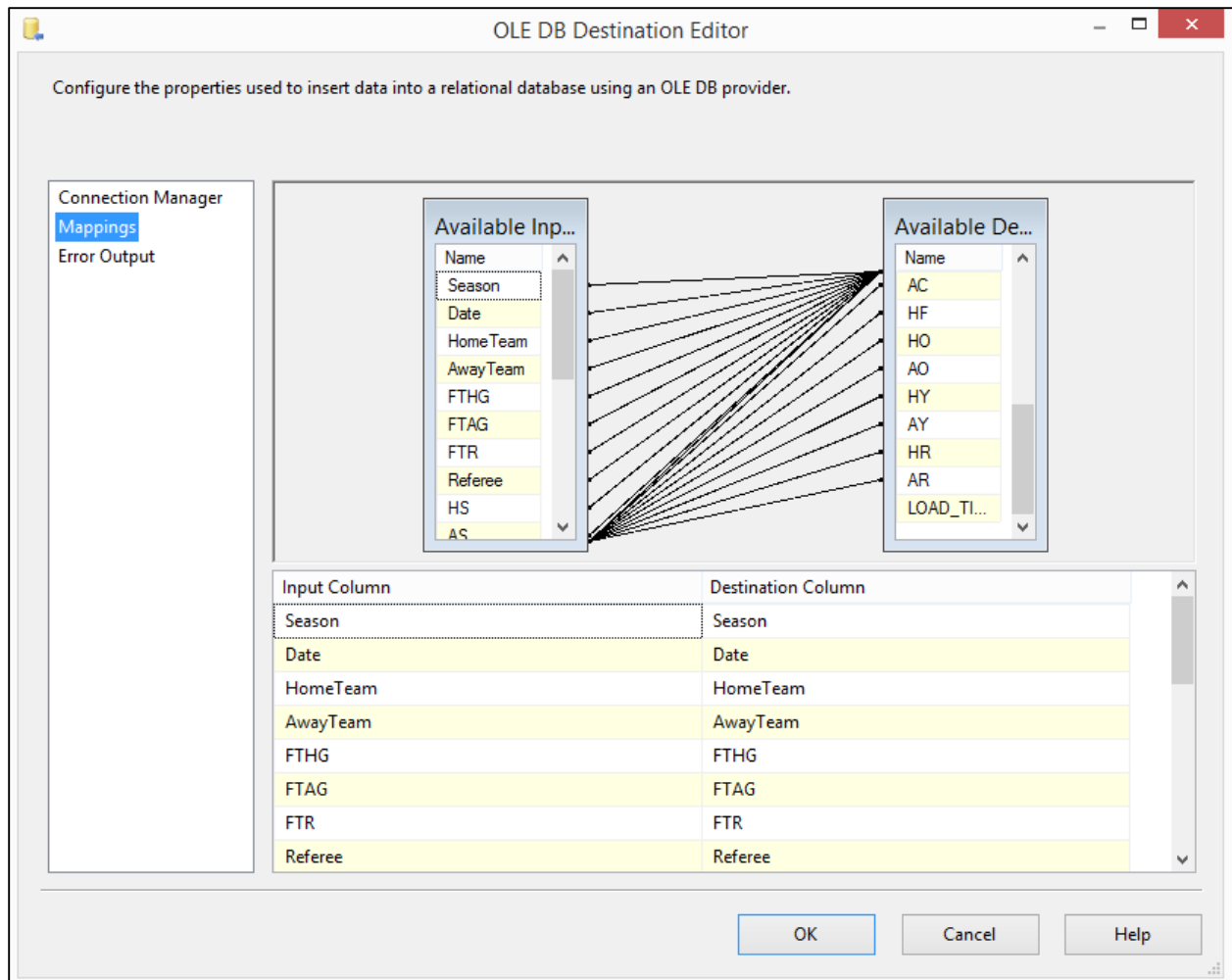


Figure 66: Mapping Columns

The goal in this editor screen is to be able to map between the available input columns and the available destination columns. Once you complete this task, you can run your package and the data will move from the source to the destination database.



Note: This is just an introductory example; it doesn't have any error control or validations. It is meant for you to see how to start developing packages with data flows in its control flows.

Chapter 5 Variables, Expressions, and Parameters

Introduction

Integration Services packages might be difficult to maintain if the rules specified in them change often. Imagine, for instance, a conditional split for a specific value coming from the source databases. In this case, you need to define a value as a condition inside your package but, if this value changes, you need to change it in the package and redeploy it.

Integration Services has powerful pieces in its architecture that will help you to resolve these problems without having to change your package execution, including variables, expressions, parameters, and some other well-known functions, operators, and literals. This chapter will focus on the first three as they are true mechanisms of adding abstraction and dynamics into your package runtime intelligence.

Understanding the components

Variables enable you to develop dynamic mechanisms inside your packages so that, according to a certain condition, they will reconfigure themselves and their execution logics will change—much like in any .NET language. In this way, you will be creating what are commonly called dynamic packages. You will be able to create and use two types of variables: system variables and user variables. User variables are created by you. Variables can also vary in scope, with the default scope being the entire package. But they can also be set to be in the scope of a container, task, or an event handler inside the package. These variables can be used in many more scenarios such as iterating through a loop, concatenating multiple values together to create a file directory, or passing an array between objects. So let's take a further look at all applicable scenarios in which variables can be used:

- Updating properties of package elements at run time, such as the number of concurrent executables that a Foreach Loop container allows.
- Including an in-memory lookup table such as running an Execute SQL Task that loads a variable with data values.
- Loading variables with data values and then using them to specify a search condition in a WHERE clause. For example, the script in a script task can update the value of a variable that is used by a Transact-SQL statement in an Execute SQL Task.
- Loading a variable with an integer and then using the value to control looping within a package control flow, such as using a variable in the evaluation expression of a For Loop container to control iteration.
- Populating parameter values for Transact-SQL statements at run time, such as running an Execute SQL Task and then using variables to dynamically set the parameters in a Transact-SQL statement.
- Building expressions that include variable values. For example, the Derived Column transformation can populate a column with the result obtained by multiplying a variable value by a column value.

Although variables are one of the key concepts of this dynamic mechanism, there are several other components that interact with variables to bring you the flexibility you need inside your package developments. Expressions, for instance, are used to evaluate a value in order to decide which path of execution the package should take. This value can be defined by a variable or by the combination of identifiers, literals, functions, and operators.

Another nice feature that SQL Server 2012 provides is parameters which, at first look, might look like variables because they are also used in expressions to condition a certain logic based in its store data (and even more, inside a package these parameters work just like a variable). However, there is a small detail that makes a huge difference: parameters are defined externally to the package and cannot be changed in development time, as opposed to variables which are used and defined internally. These parameters can be defined in two scopes: package-level in which they are available for a particular package, or project-level in which all packages of a project can be used.

If you need a simple way to decide whether you need a variable or a parameter, decide if you want to set the value of something from outside the package or if you want to create or store values only within a package. If the first case, use parameters; otherwise, use variables.

Creating New Variables

Creating variables is easy using the Variables window of the SSIS Designer. To create a new variable, click **Add Variable**. Give your new variable a name, data type, and value. This variable value can come from an expression as well; if you want to use an expression, leave the value blank.

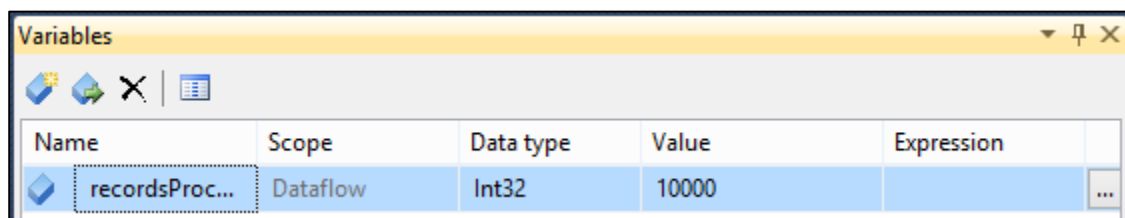


Figure 67: Creating a new variable

To call variables inside your packages, you can use the following syntax: @[User::VarName] for user variables and @[System::VarName] for system variables.

Creating New Expressions

Expressions are used along many components inside the Integration Services Designer. However, the window used to create them is the same in each. What we are going to do in this example is create a string with the following pattern: "Today is:" concatenated with a dynamic GETDATE() function. So let's start by defining our variable name and data type as shown in the following figure.

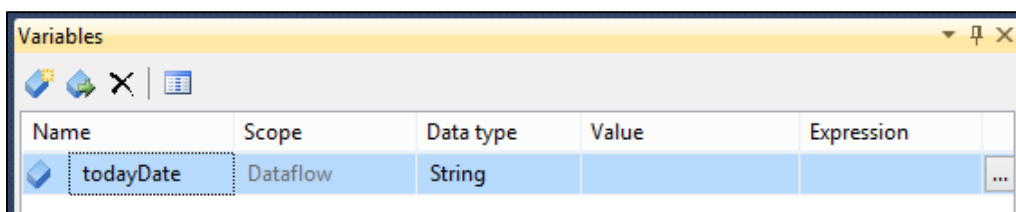


Figure 68: Created Variable

Now let's create a new expression inside the Variable window by clicking the **Expression** button. The Expression Builder will open, and you can use it to write your expressions.

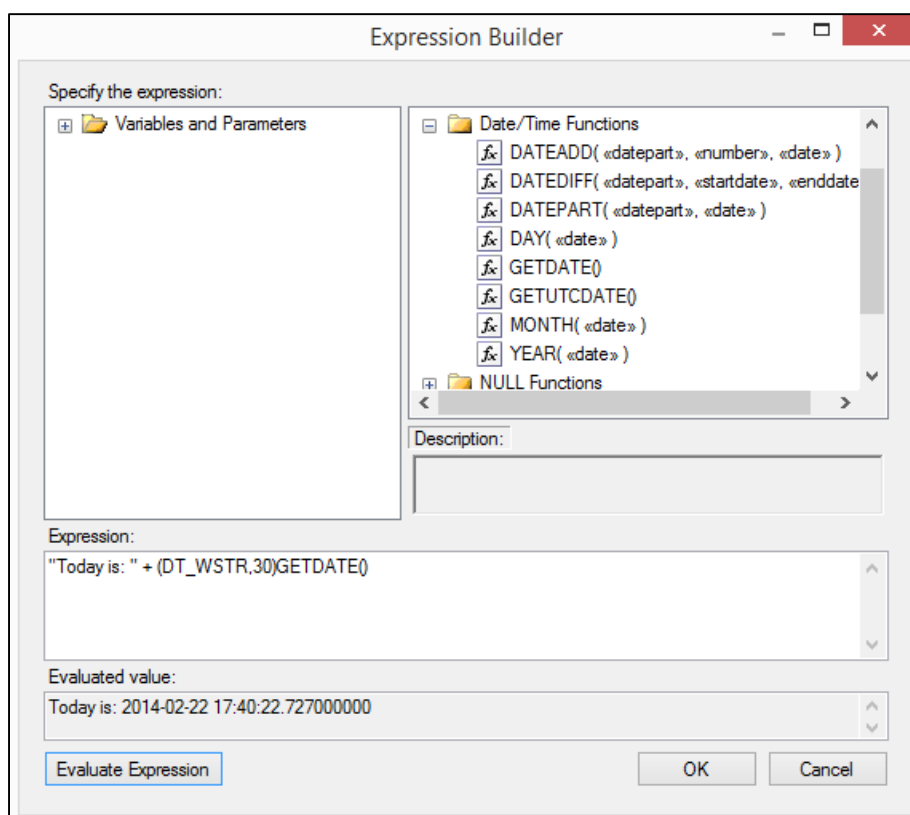


Figure 69: Writing the Expression

Creating New Parameters

In older versions of SSIS, you could use configurations to update the values of package properties at run time; in SSIS 2012, these configurations are replaced by parameters which can be applied to a single package or at the project level. The slowly changing dimensions (SCD) modeling technique and the mechanism of extracting only changed records from source systems is called Change Data Capture (CDC).

Although I will briefly introduce each one of these topics, my main goal is to make sure you understand the importance of including these two topics inside your ETL projects—and of seeing how you can start using them right now.

Handling Slowly Changing Dimensions

Introduction

Slowly changing dimensions (SCD) are a special data warehouse modeling technique that allows you to manage how your ETL process will resolve updates and inserts into a dimension so that you can tell your packages what to do if an attribute value changes. SCDs were first introduced by Ralph Kimball in the form of three possible types which Kimball has recently reviewed. He has recently presented [more types](#) of SCDs. However, in this book, I will be explaining only the three original types (note that the SSIS SCD component doesn't support SCD3):

- SCD1, in which an attribute value will be overwritten if the record has changed (used if historical data is not needed).
- SCD2, in which a new record will be created whenever an attribute value changes (used to keep all historical changes).
- SCD3, in which N columns are created in a dimension to keep N changes in it (only records the last N changes).
- Some authors refer to another type when no change is applied to an attribute (fixed attribute); the SCD type is known as 0. I will refer to this fourth type as "SCD0".

For some time, the integration software available on the market didn't support any kind of wizard or automation for developing SCD ETL packages; this meant that if you wanted to handle an SCD inside your packages, you had to create the entire data flow logic to handle it.

However, SSIS gives you the SCD component inside data flows so that you can use a wizard to create the flow for handling SCD. But there are two important things to note.

First, the SCD component doesn't support SCD3 so, if you want to use it, you need to develop the flow yourself. Because SCD presents several performance issues, it's better to always think twice about using it.

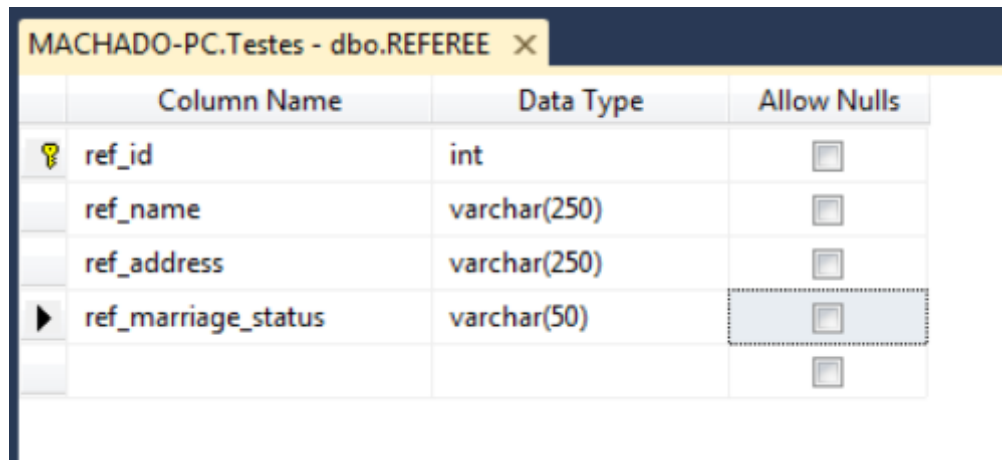
Second, if you want to use SCD2, you have to create two new columns in your dimension: the beginning date of the record life (SCD_DT_BEGIN, for example) and the end date of the record life. The latter represents the time in which a new record with different attribute values is identified in your ETL process (SCD_DT_END, for example).

If you aren't familiar with using SCD dimensions, another important thing to note is that in the same dimension you can have SCD1 attributes, SCD2 attributes, and even SCD3 attributes. This is because when a change occurs in a specific attribute, you might not need to record it (SCD2) but you might want to update its value. These requirements will vary from business to business.

Demo: Using the Slowly Changing Dimension Component

Let's start with an example that will demonstrate how easy it is to use the wizard inside SSIS 2012. I will show you all available options inside it by using all three SCD types in a single dimension. My scenario is based on a football referee operational system that will lead to a SCD

dimension called DimReferee. Both are very simple dimensions with few attributes. The source table is shown in the following figure.





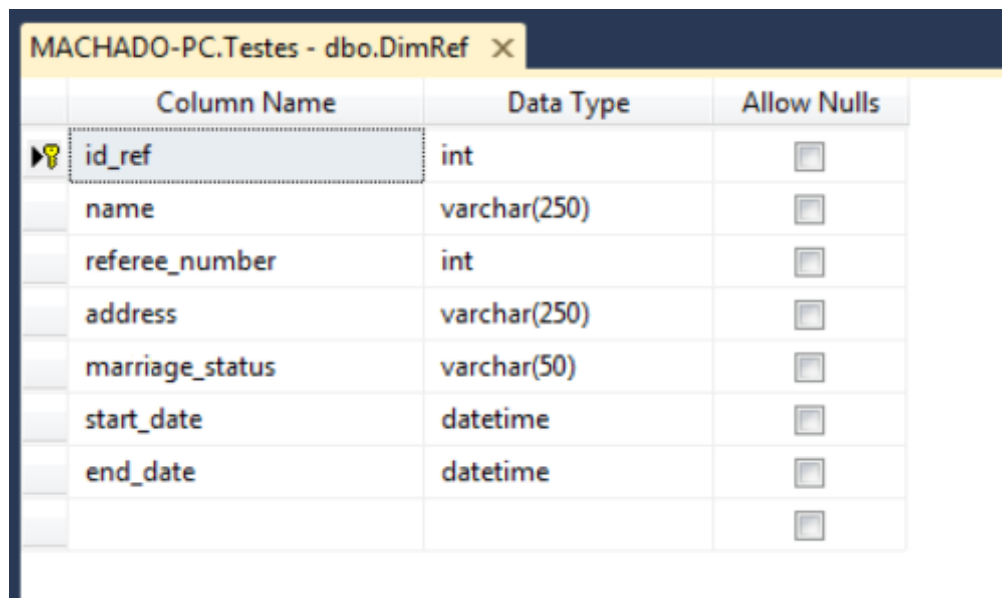
	Column Name	Data Type	Allow Nulls
	ref_id	int	<input type="checkbox"/>
	ref_name	varchar(250)	<input type="checkbox"/>
	ref_address	varchar(250)	<input type="checkbox"/>
	ref_marriage_status	varchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

Figure 70: SCD2 Source Table

The DimReferee dimension is structured as follows. As you can see, I have already added the start_date and end_date attributes which will be used to flag whether or not a SCD2 record is active. The SSIS SCD component allows you to use a Status column as well.




	Column Name	Data Type	Allow Nulls
	id_ref	int	<input type="checkbox"/>
	name	varchar(250)	<input type="checkbox"/>
	referee_number	int	<input type="checkbox"/>
	address	varchar(250)	<input type="checkbox"/>
	marriage_status	varchar(50)	<input type="checkbox"/>
	start_date	datetime	<input type="checkbox"/>
	end_date	datetime	<input type="checkbox"/>
			<input type="checkbox"/>

Figure 71: SCD2 Destination Table

Now that both columns are created, let's go to Integration Services. The first task is to add the SCD component to the data flow of your dimension. The following figure shows what it should look like. Our example package will be very simple: just a source component to read the referee data from the source, the flow generated with the SCD component, and a destination component to send our data to the dimension.

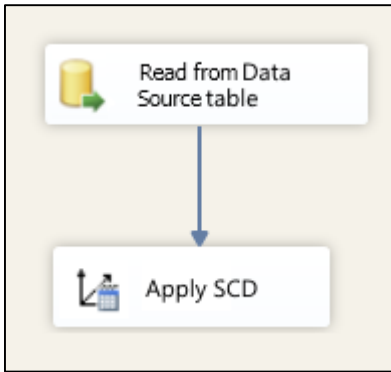


Figure 72: Adding the SCD component to the data flow

There is an intermediate step that might or might not be needed depending on your attribute types in the source and destination tables, and that step is converting the data types. If you detect that your attribute types do not match, add a conversion component to your data flow to be executed before the SCD component. Your data flow should now look like the following figure.

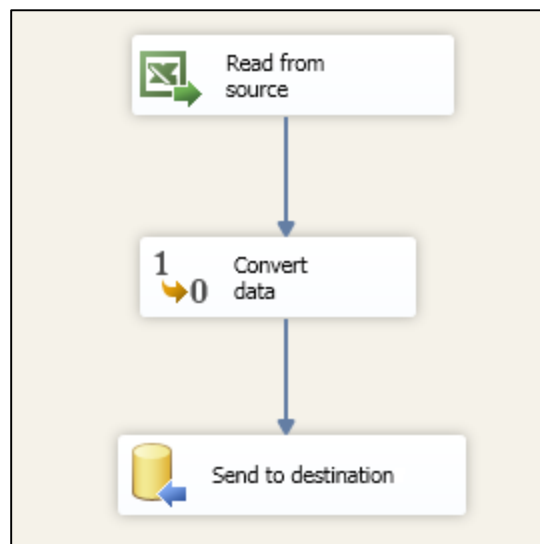


Figure 73: SCD component data flow—Adding a conversion component

To start configuring the SCD component, double-click it to open a wizard that will help you configure your SCD. On the initial screen, click **Next**.



Figure 74: SCD Wizard

This screen is very important. It is where you define the mappings between source and destination attributes and the business key, which will be used to look up the record in the source table. As you can also see, you need to define the connection manager for the dimension and the table you want to transform in an SCD dimension.

For our example, let's map all attributes as in the following figure and choose the `id_referee` (primary key of the Referee table) as our business key. This business key is a key that works as a primary key for a record in our dimension and identifies a record in the operational systems. It is also commonly called a natural key. The start and end date will remain blank because they will be configured later.

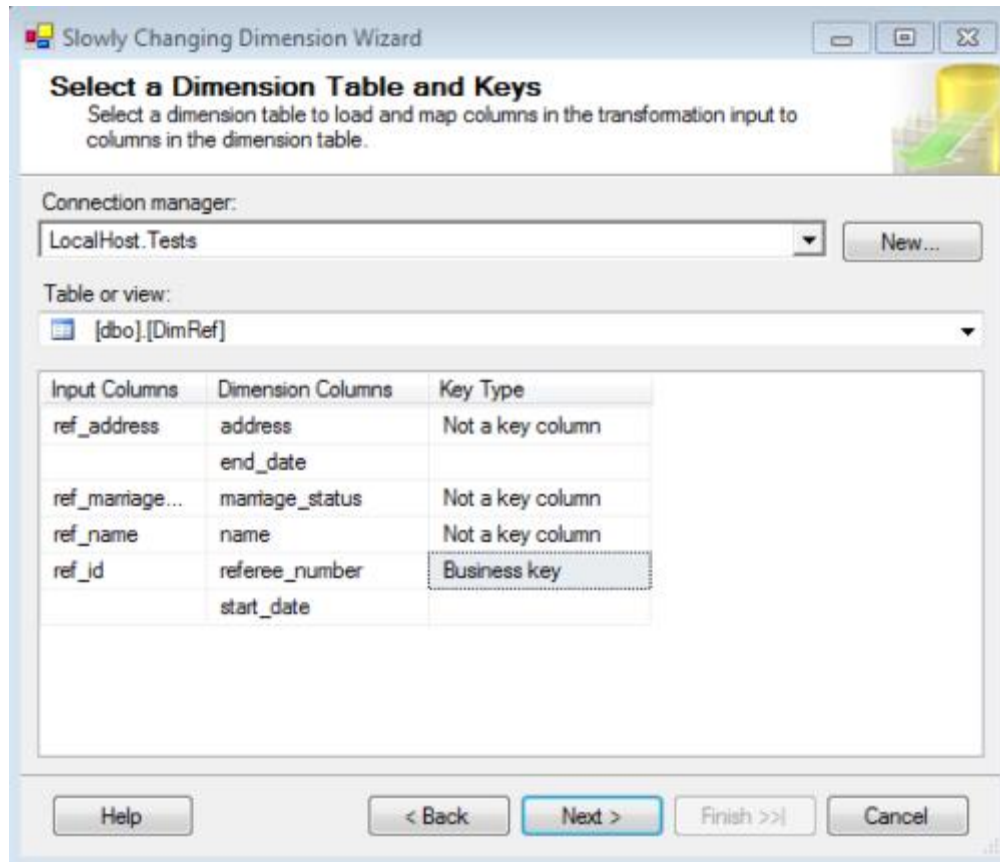


Figure 75: Configuring Business Keys

Clicking **Next** will bring us to the most important screen in the SCD Wizard. It is in this screen that you define the type of SCD for each attribute mapped in the first screen. There are three available options:

- Fixed Attribute (SCD0)
- Changing Attribute (SCD1)
- Historical Attribute (SCD2)

As I have explained before, SCD3 is not supported in the SSIS component. In this example, I will use all of the types. After setting a referee's name, I don't plan to do anything else with it because I assume that the name won't change, so I make it a fixed attribute (SCD0). However, when a referee's marital status changes, I want to overwrite its value to the new one, so I set it as a changing attribute (SCD1). When the referee's address changes, I want to save the old value, so it needs to be a historical attribute (SCD2).

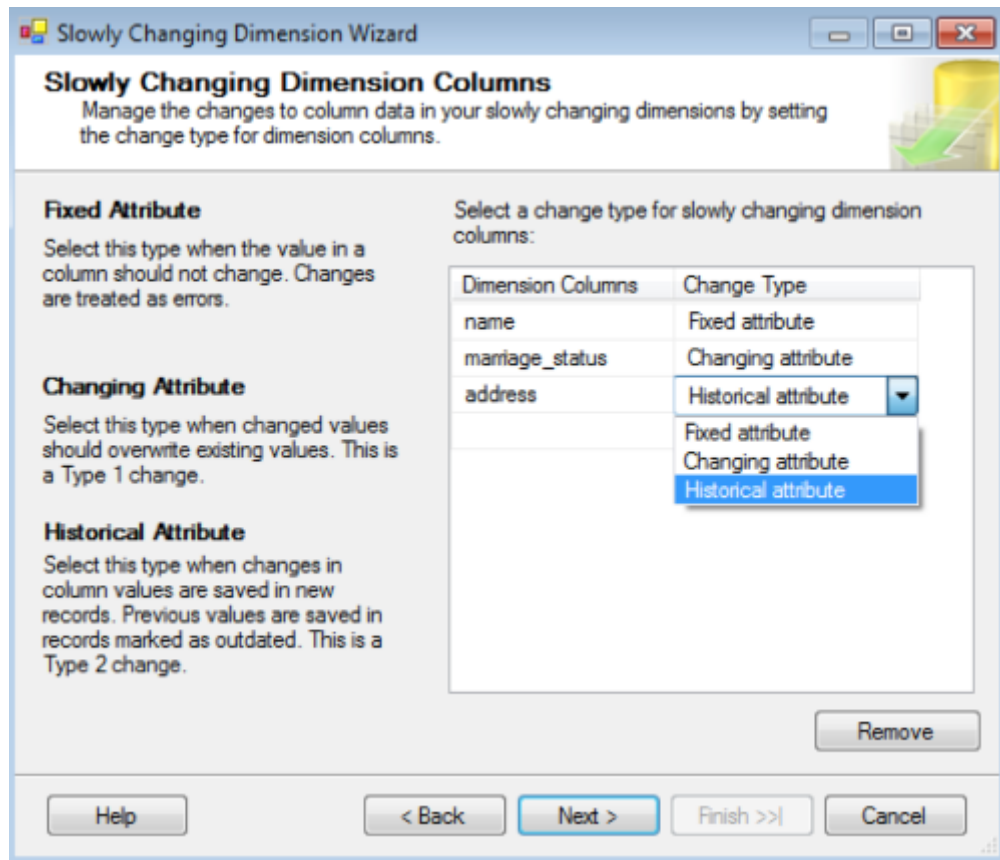


Figure 76: Configuring the change type for SCD attributes

The next screen lets you choose what SSIS should do when a fixed attribute value changes, and if you want to update even the SCD2 historical records if a type SCD1 value changes. In the first case, you can tell SSIS to raise an error if a defined SCD0 attribute changes its value and the change is detected. In the second case, if an SCD attribute changes its value, you need to define whether you want to overwrite its value only in the most recent record or in all of them (historical ones).

The last case only makes sense if you have an SCD2 attribute in the dimension. In our example, let's raise an error if the referee name changes, if we might have misunderstood the business, or if it might need to be adjusted. In the second check box, let's leave it blank since we just want to update the active record only.

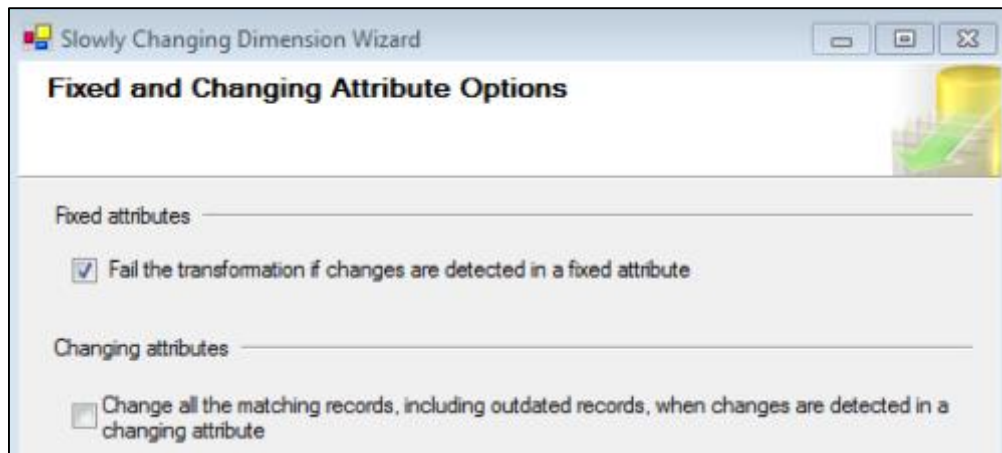


Figure 77: Configuring the behavior when changes are detected in non-changing attributes

The next screen is where you will define whether or not you will use a Status column to indicate the current (most recent/active/alive) record or, if you prefer, to use a time interval. I prefer to use dates as they allow me to know which period that record existed in. This time, let's use our start_date and end_date attributes as a flag to the current record. We also need to indicate which type of variable we are going to use to populate the dates. To do so, we have three options:

- ContainerStartTime—The start time of the container which the package is in.
- CreationDate—Date of creation of the package.
- StartDate—The start date of execution of the package.

I prefer the last option as it is the closest to the SCD flow in the package, reflecting the most recent date. However, for this example, I used the CreationDate. After selecting the variable, click **Next**.

Slowly Changing Dimension Wizard

Historical Attribute Options
You can record historical attributes using a single column or start and end date columns.

☐ Use a single column to show current and expired records

Column to indicate current record:

Value when current:

Expiration value:

☒ Use start and end dates to identify current and expired records

Start date column:

End date column:

Variable to set date values:

Help < Back Next > Finish >> Cancel

Figure 78: Configuring how to identify current and expired records

This screen is delicate. An interesting concept related to data warehouses using SSIS and SQL Server is the inferred member; this is a record from your dimension that is referenced by fact tables. However, the record is not yet loaded to the dimension. This means that a particular record can be referenced without existing.

This might occur if, for some reason, you need to load facts without having the dimension record data yet. So you create a blank/null record in a dimension that won't have any data, just its primary key. The next screen allows you to tell SSIS what to do when the data for that inferred member arrives. You can either update the record or create a new one. If you activate the inferred member support, you have two options:

- All columns with a change type are null. This means that for every column in an inferred member record that is configured with an SCD type, SSIS will set its values to null.
- Use a Boolean column to indicate whether or not the current record is an inferred member. In this case, SSIS will use a column created by you to flag whether or not the current record is an inferred member.

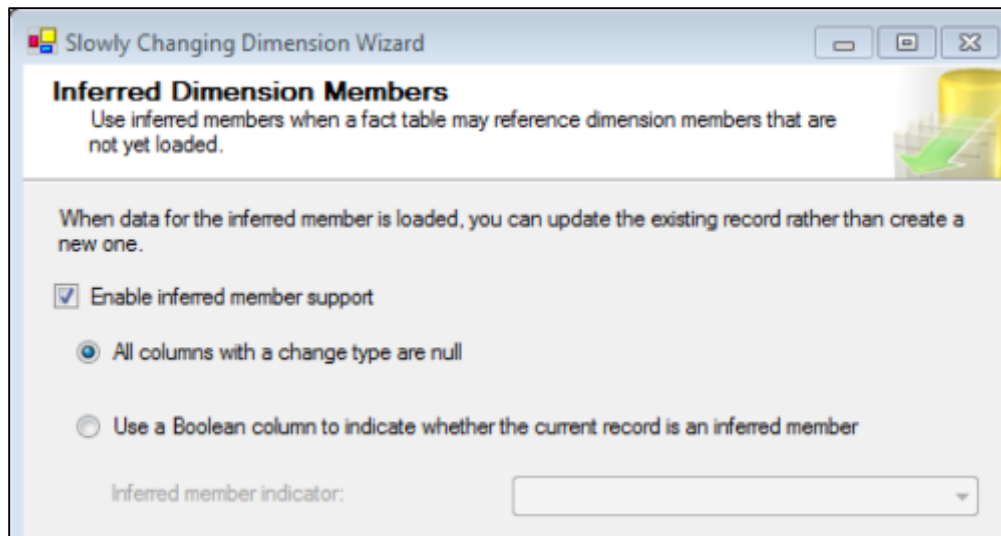


Figure 79: Configuring Inferred Members

After configuring the inferred members, click **Next**, and you are done. SSIS will now automatically create the flow for you which, as you can see in the following figure, represents a lot of time saved with transformations and component configurations. However, if there are further transformations (such as data type conversions), you still need to do them. For instance, in our example we still need to convert some data types before running our package.

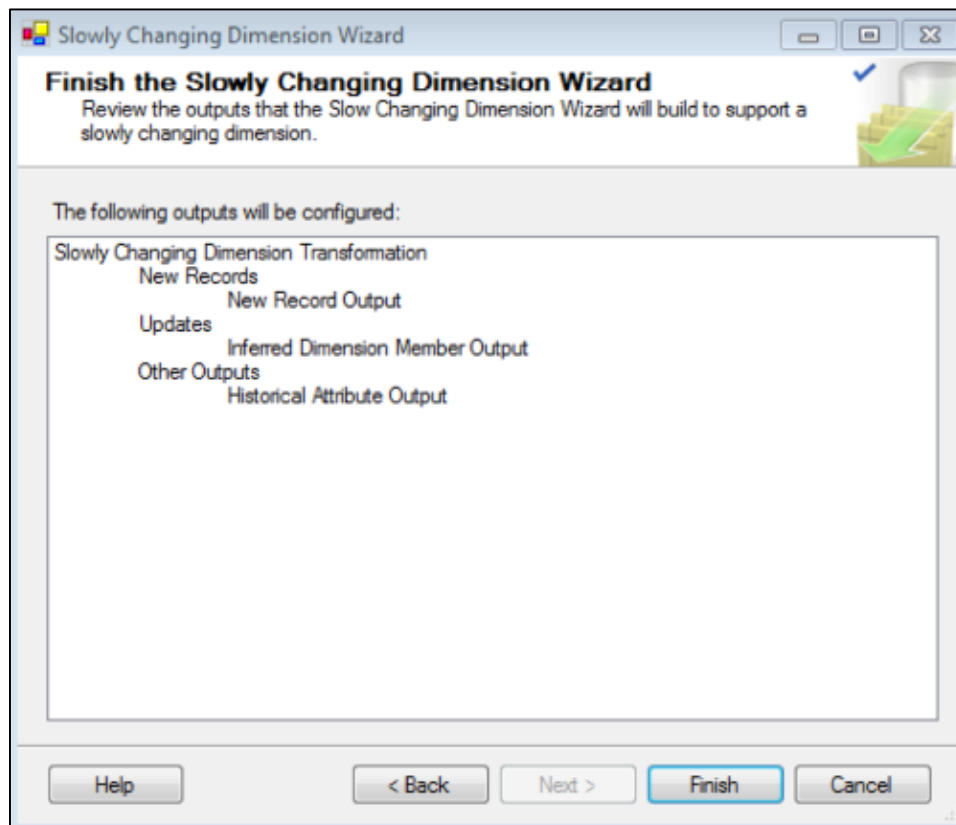


Figure 80: Finishing the Configuration

Once you have added a conversion component to ensure data type consistency, your data flow should look like the following figure.

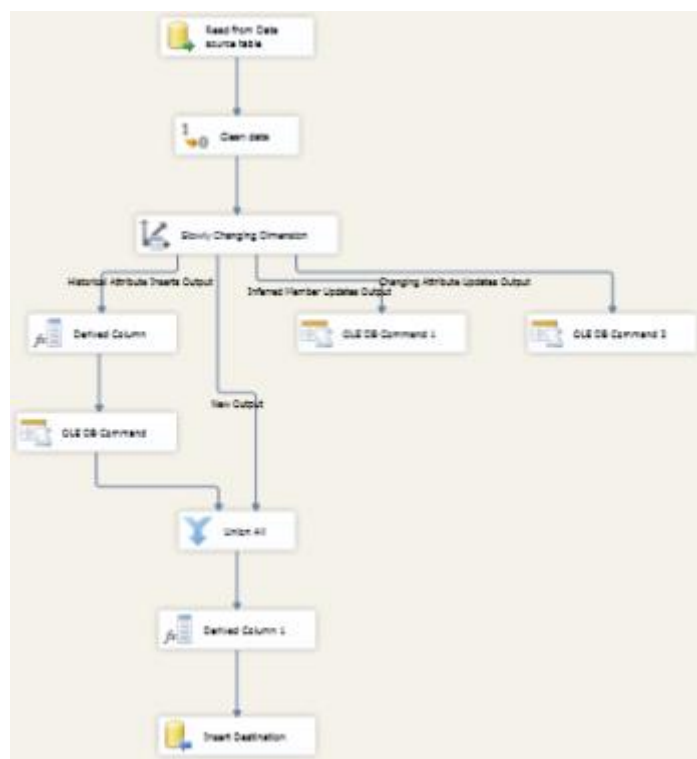


Figure 81: Final data flow with SCD component

And it's done! Your package for dimension DimReferee will now perform SCD evaluation and resolve it according to the rules specified in the SCD component. You can now run the package and test it yourself.

Change Data Capture

Introduction

Along with slowly changing dimensions (SCD), change data capture (CDC) is one of the most important mechanisms in handling changes. While SCDs define how you want to treat changes in your dimension attributes, CDC allows you to detect changes in the operational system since the last extraction process and retrieve only the changed data. By changed data, I mean updates, inserts, and also deletes from operational sources.

As you can imagine, the extraction of data from operational sources is very important in ETL operations using Integration Services or any other provider. This is because you do not want to process all the data again when you run your packages; you just want to process the records that have changed (inserts, updates, and deletes). If you schedule the execution of your ETL process every day at 2:00 in the morning for the processing of a four-year operational system database, you don't want to process all the data from those four years. You just want to process

the records that have changed since the previous night. In this way, your ETL process will be faster and much more efficient.

This mechanism was introduced in SQL Server 2008 and, since then, Microsoft has made many improvements to it. Proof of this are the new CDC components that SSIS 2012 includes. This is not the only option available; however, it probably is the most efficient. I will explain why.

Over time, database developers have created mechanisms to solve this problem. Some of those mechanisms are very smart; however, they bring with them particular problems. To understand these problems, let's look at all options available to capture changed data in a specific time window. Here, I explain the most used alternatives:

- **Audit Columns:** This technique requires you to add datetime columns to each table you want to monitor for changes in the operational systems. By adding a start date and end date to the tables you change, you are able to change them every time the records change and then extract only the records for which the end date is null or higher than the last ETL round. This technique also requires you to create triggers in the source tables or external applications to update the audit column with the latest update changes in your data source table structure.
 - **PROBLEM:** In some systems, the size and amount of tables, as well as DBAs with bad tempers, make this an impossible solution due to cost inefficiency.
- **Triggers:** This technique requires you to add triggers into each of the source tables you want to monitor. Every time an insert or delete occurs, the trigger will store the changed record business key into a log table. Your ETL process will then process only the records from the log table.
 - **PROBLEM:** This will affect the performance of the operational systems, and you will also have to change all the triggers of the tables to monitor. Again, you will need to change the operational system.

Now comes the CDC technique, which solves the previously mentioned problems. Although you still need to enable CDC within each table you want to track changes, it does not affect the schema of the tables. So the impact over the operational systems' database schemas is less when compared to audit columns or triggers.

Another interesting aspect of this technique is that it runs asynchronously. By this I mean that the job that reads from the SQL Server log to detect changes will only run when the system is idle, removing overhead from the process (when compared to triggers and audit columns which run synchronously).

The way the SQL Server CDC mechanism works together with Integration Services CDC components is also a huge benefit because you are able to extract the changed records into a package control flow and also know what kind of operation has occurred in that record. In this way, you can make separate developments over an inserted, updated, or deleted record.

Last but not least comes the usability of the CDC API. It is very easy to activate this mechanism as a SQL Server table and then handle it inside an SSIS package. The steps needed to do so will be shown in a brief demonstration so that you can start using it to optimize the extraction stage in your ETL projects.

The use of this CDC API requires you to do the following steps in order to start processing only changed records inside SSIS packages:

1. Enable CDC in the database.
 - This will create a new database schema (*cdc*) to record the information it needs to manage the mechanism.
2. Enable the SQL Agent in the server of the CDC.
 - This will enable the job that will analyze the logs and write the change to the CDC tables (these are the tables that record the changes, also called shadow tables). This new table has the same schema as the source plus some extra columns.
3. Enable CDC in the tables we want to monitor for changes.
 - This will create the instance table for the particular table you are monitoring. The SQL Agent job that is tracking the changes writes the changed records into these tables. Your ETL process will then read from these tables so that it only processes the changed records. There is an internal mechanism that cleans all the data inside it after three days by default, but you can change this limit. However, after processing the data, the ETL process should clean the table so that you won't process them again.
4. Process the changed data in SSIS and use the SSIS CDC component to make it easier.
 - There are two main components:
 - In the control flow, the CDC Control task is used to check the CDC state (if it is time to reprocess or if the data is updated). This task will create a new table in a database so that it can keep this information.
 - In the data flow, the CDC Source queries the change tables and the CDC Splitter splits the records based on the SQL Server DML operation (Insert, Delete, or Update).

Processing the data involves the following algorithm in your SSIS package:

1. Check the CDC state.
 - If it is time to reprocess:
 - a. Query the data using the CDC Source.
 - b. Send the data to the appropriate path (Insert, Delete, or Update).
 - c. Process the data according to your needs in each path.
 - If the data is updated, don't do anything.

And that's it! By following these steps, you are able to create and maintain a CDC-based ETL solution. Now it's time to show you how these steps are accomplished, which queries you should run in SQL Server, and what developments are required inside SSIS 2012.

Demo: Using CDC in an ETL Project

SCENARIO: We will be monitoring a table related to football games. Whenever a new record is inserted, deleted, or updated, we want to track that change and process it in our daily ETL process. The data will have our business intelligence data warehouse as the destination and, in this particular case, we will focus on updating our dimension (perspective of analysis in a business intelligence data warehouse architecture) related to referees.

Enable CDC on the Database

Let's start by opening the SQL Server Management Tools to run some queries or, if you prefer, execute them using an Execute T-SQL task inside a package. Although I love Integration Services, the first option seems less geeky.

The first step is to run the following SQL query. In your case, you just need to change the database name. `sp_cdc_enable_db` is the store procedure used by SQL Server to enable CDC in the current database.

```
USE STG_FOOT_STAT; -- DATABASE_NAME
GO
--Enable CDC on the database
EXEC sys.sp_cdc_enable_db;
GO
```

After running the previous command, SQL Server will create a special schema, `cdc`, which will be used to store all the objects it needs to manage the mechanism. In these objects, you will be able to find all your shadow tables. You are now able to check if the CDC is correctly enabled for that particular database. To do so, you can run the following SQL query which will retrieve the name and the value of the `is_cdc_enabled` attribute for your current database.

```
USE STG_FOOT_STAT; -- DATABASE_NAME
GO
--Check if CDC is enabled on the database
SELECT name, is_cdc_enabled
FROM sys.databases WHERE database_id = DB_ID();
```

Running the previous query will result in the following, in which the `is_cdc_enabled` attribute will be one of two possibilities:

- 0—Not enabled
- 1—Enabled

	name	is_cdc_enabled
1	STG_FOOT_STAT	1

Figure 82: Result of checking if CDC is enabled

Enable the SQL Agent in the Server of the CDC

The next important step is to check if the SQL Server Agent is running on the servers on which you want to use CDC. This is important because CDC uses a SQL job to crawl the change logs in order to write to the shadow tables.

This means that if the agent isn't running, you won't be able to capture changes automatically. An interesting thing about this job is that you can schedule the periodicity of its execution (daily,

hourly, etc.). You have two options to activate the SQL Server Agent: either use the SQL Server Configuration Manager or use SQL Server Management Tools.

Let's start with the first option. Open the SQL Server Configuration Manager and, in the explorer on the left, select **SQL Server Services**. When you do this, you will see all your servers' services in the right pane which indicate whether or not they are running. Identify the SQL Server Agent of the server you want to activate, which in my case is the MSSQLSERVER instance. Right-click it and select **Start** if it isn't running yet. If everything worked fine, you should then see that the agent is running.

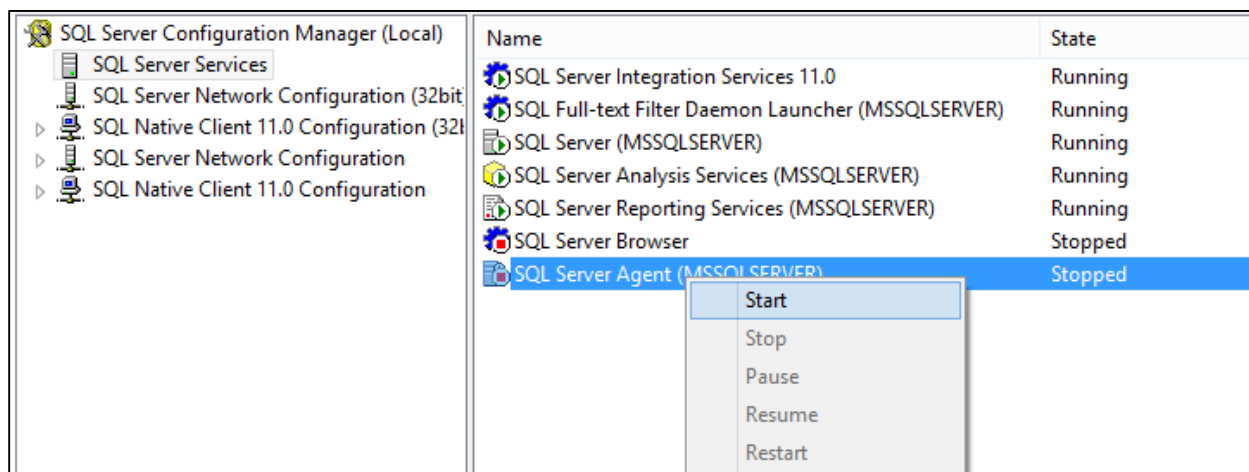


Figure 83: Activating the SQL Agent in Configuration Manager

Another option is to use the SQL Server Management tools. It's also very simple to use this option. Connect to the SQL Server instance in which you want to activate the Agent and then find the SQL Server Agent inside the Object Explorer. To start it, just right-click it and select **Start**. And that's it! The small state indicator in the icon should now be green.

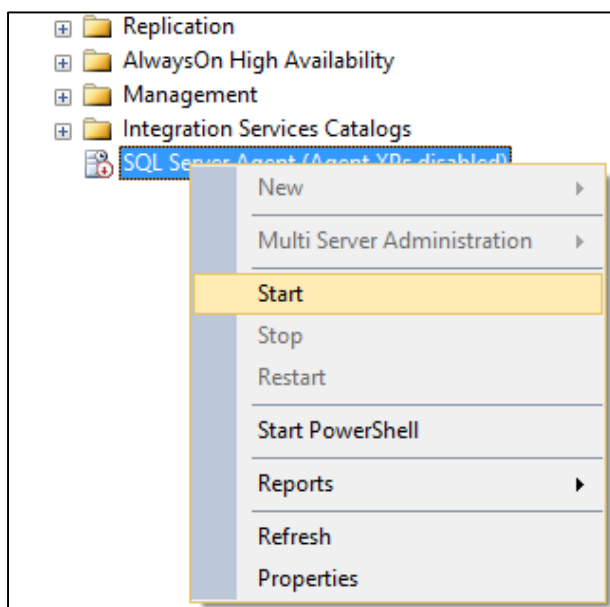


Figure 84: Starting SQL Agent using SQL Server Management Tools

Enable CDC in the Tables We Want to Monitor for Changes

Now it's time to show you how you can start monitoring the changes of a table. As I have already explained, this will create shadow tables in the cdc schema with all your source table columns, plus some special CDC ones.

The mechanism will then work in a very simple algorithm. When a record of each Data Manipulation Language (DML) operation applied to the table is written to the transaction log, the CDC process retrieves this information from the log and writes it to these shadow tables that are accessed by using a set of functions.

To enable it in a table, you need to run the stored procedure `sys.sp_cdc_enable_table` with some special parameters as shown in the following code example.

```
USE STG_FOOT_STAT;
GO
--Enable CDC on a specific table
EXECUTE sys.sp_cdc_enable_table
@source_schema = N'dbo'
,@source_name = N'STG_FOOT_STAT_TB'
,@role_name = N'cdc_Admin'
,@capture_instance = N'STG_FOOT_STAT_TB'
,@supports_net_changes = 0;
```

As you can see in the stored procedure, I have used five parameters. However, there are some more that will help you align the CDC with your requirements. You need to understand all of them to execute it correctly. The definition for each one is as follows.

```
sys.sp_cdc_enable_table
[ @source_schema = ] 'source_schema',
[ @source_name = ] 'source_name' ,
[ @role_name = ] 'role_name'
[, [ @capture_instance = ] 'capture_instance' ]
[, [ @supports_net_changes = ] supports_net_changes ]
[, [ @index_name = ] 'index_name' ]
[, [ @captured_column_list = ] 'captured_column_list' ]
[, [ @filegroup_name = ] 'filegroup_name' ]
[, [ @allow_partition_switch = ] 'allow_partition_switch' ]
```

The following table gives you a short definition of the parameters you can use in the `sys.sp_cdc_enable_table` stored procedure.

Table 10: Enable CDC in table parameters

Parameter	Definition
@source_schema	Name of the schema in which the source table belongs.
@source_name	Name of the source table in which to enable change data capture.
@role_name	Name of the database role used to gate access to changed data.
@capture_instance	Name of the capture instance used to name instance-specific changed data capture objects. If not specified, the name is derived from the source schema name plus the source table name in the format schemaname_sourceName.capture_instance.
@supports_net_changes	<p>The supports_net_changes option enables you to retrieve only the final image of a row, even if it was updated multiple times within the time window you specified. This parameter can only be activated if you have a primary key defined in the source table, otherwise you will receive the following error:</p> <p>Msg 22939, Level 16, State 1, Procedure sp_cdc_enable_table_internal, Line 194</p> <p>The parameter @supports_net_changes is set to 1, but the source table does not have a primary key defined and no alternate unique index has been specified.</p>
@index_name	Name of a unique index to use to uniquely identify rows in the source table.
@captured_column_list	Source table columns that are to be included in the change table. captured_column_list is nvarchar(max) and can be NULL. If NULL, all columns are included in the change table.
@filegroup_name	Filegroup to be used for the change table created for the capture instance.
@allow_partition_switch	Indicates whether the SWITCH PARTITION command of ALTER TABLE can be executed against a table that is enabled for change data capture. Its default is 1

If everything in your execution is correctly defined, you will see the following message in the output windows of SQL Server.

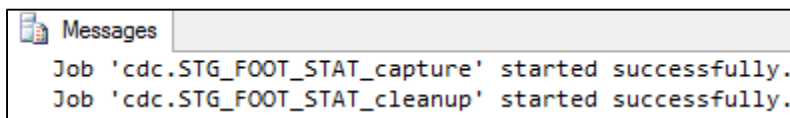


Figure 85: Created the job for the STG_FOOT_STAT table

Now you can use several commands to evaluate whether or not CDC is correctly working for your table. The first option is to use the `is_tracked_by_cdc` attribute in the “tables” system table of SQL Server. To use it, run the following command which, as you can see, only shows you whether or not CDC is running.

```
--Check CDC is enabled on the table
SELECT [name], is_tracked_by_cdc FROM sys.tables
WHERE [object_id] = OBJECT_ID(N'dbo.STG_FOOT_STAT_TB');
```

This following figure shows the result of executing the previous command, which results in 0 if the table is not being tracked and 1 if it is. In this case, the table is being tracked as it was supposed to, after running the enable stored procedure.

The screenshot shows the 'Results' window in SQL Server. It displays a single row with two columns: 'name' and 'is_tracked_by_cdc'. The value for 'name' is 'STG_FOOT_STAT_TB' and the value for 'is_tracked_by_cdc' is '1'.

	name	is_tracked_by_cdc
1	STG_FOOT_STAT_TB	1

Figure 86: CDC enabled for the table

Another option that will give you more information on the tracking is to use the built-in CDC stored procedure `sp_cdc_help_change_data_capture`. It will give you much more information including which columns are being tracked, the capture instances available (you can have two per table) and some others that you can explore. To run it, use the following code.

```
--Use the built-in CDC help procedure to get more information
EXECUTE sys.sp_cdc_help_change_data_capture
@source_schema = N'dbo',
@source_name = N'STG_FOOT_STAT_TB';
GO
```

And, as you can see, the result is much more interesting.

The screenshot shows the 'Results' window in SQL Server. It displays a single row with eight columns: 'source_schema', 'source_table', 'capture_instance', 'object_id', 'source_object_id', 'start_lsn', 'end_lsn', and 'supports_net_changes'. The values are: 'dbo', 'STG_FOOT_STAT_TB', 'STG_FOOT_STAT_TB', 1237579447, 261575970, 0x000000440000018A0099, NULL, and 0.

	source_schema	source_table	capture_instance	object_id	source_object_id	start_lsn	end_lsn	supports_net_changes
1	dbo	STG_FOOT_STAT_TB	STG_FOOT_STAT_TB	1237579447	261575970	0x000000440000018A0099	NULL	0

Figure 87: CDC information about table tracking

With the source tables being tracked, you can now take advantage of the shadow tables directly from SQL Server because it allows you to query them and see the changes that have occurred. The CDC mechanism will create and name these shadow tables with a standard “Source table name” + “_CT”, like “myTable_CT”, for instance. This means that by knowing the source table name, you also know the shadow table name. To query it, you can use a simple SQL SELECT statement as in the following code example.

```
SELECT * FROM cdc.STG_FOOT_STAT_TB_CT
```

Because you haven’t made any changes to the source table, the shadow table is empty as well. However, you can see the special columns in this shadow table.

__\$start_isn	__\$end_isn	__\$seqval	__\$operation	__\$update_mask	Season	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	Referee	HS
---------------	-------------	------------	---------------	-----------------	--------	------	----------	----------	------	------	-----	---------	----

Figure 88: Querying the shadow table

The columns are very important for Integration Services because it will use them to know, for instance, what operations have been performed on a record. Let’s take a closer look at all of them. The **__\$start_isn** and **__\$seqval** columns identify the original transaction and the order in which the operations occurred. The **__\$operation** column shows the source operation that caused the change (1 = Delete, 2 = Insert, 3 = Update (before image), 4 = Update (after image), and 5 = Merge). The **__\$update_mask** column contains a bit mask indicating which specific columns changed during an update.

Now, let’s perform a DML operation and then check this shadow table again. In this case, I will use **Update**.

```
USE STG_FOOT_STAT;
GO
UPDATE stg_foot_stat_tb
SET HomeTeam = 'MANCHESTER UNITED'
WHERE HomeTeam = 'Man United';
```

If you re-query the shadow table, you will see the changes tracked in it.

Results		Messages						
	__\$start_isn	__\$end_isn	__\$seqval	__\$operation	__\$update_mask	Season	Date	HomeTeam
483	0x00000048000001600024	NULL	0x0000004800000160001E	3	0x000004	2007/2008	2007-12-08 00:00:00.000	Man United
484	0x00000048000001600024	NULL	0x0000004800000160001E	4	0x000004	2007/2008	2007-12-08 00:00:00.000	MANCHESTER UNITED
485	0x00000048000001600024	NULL	0x0000004800000160001F	3	0x000004	2007/2008	2007-12-23 00:00:00.000	Man United
486	0x00000048000001600024	NULL	0x0000004800000160001F	4	0x000004	2007/2008	2007-12-23 00:00:00.000	MANCHESTER UNITED
487	0x00000048000001600024	NULL	0x00000048000001600020	3	0x000004	2007/2008	2008-01-01 00:00:00.000	Man United
488	0x00000048000001600024	NULL	0x00000048000001600020	4	0x000004	2007/2008	2008-01-01 00:00:00.000	MANCHESTER UNITED
489	0x00000048000001600024	NULL	0x00000048000001600021	3	0x000004	2007/2008	2008-01-12 00:00:00.000	Man United
490	0x00000048000001600024	NULL	0x00000048000001600021	4	0x000004	2007/2008	2008-01-12 00:00:00.000	MANCHESTER UNITED
491	0x00000048000001600024	NULL	0x00000048000001600022	3	0x000004	2007/2008	2008-01-30 00:00:00.000	Man United
492	0x00000048000001600024	NULL	0x00000048000001600022	4	0x000004	2007/2008	2008-01-30 00:00:00.000	MANCHESTER UNITED

Figure 89: Changes tracked in shadow table

Process the Changed Data in SSIS and Use the SSIS CDC Component to Make It Easier

Now that you understand CDC, let's go back to Integration Services and make use of it. As previously mentioned, it's one of the most important concepts to understand. Otherwise, if you don't understand it, while processing the ETL of big data, your process will be very inefficient and, consequently, slow. Of course, this only makes sense when you have periodic ETL processes. If you only run a single execution, such as a data migration, CDC isn't useful.

Process Initial Load

The first step is to add a new CDC Control task to your package data flow. This task will evaluate whether or not there are records to process and mark the initial load state in the CDC state variable. After that, we need to add our control flow logics and, in the end set, the initial load end state.

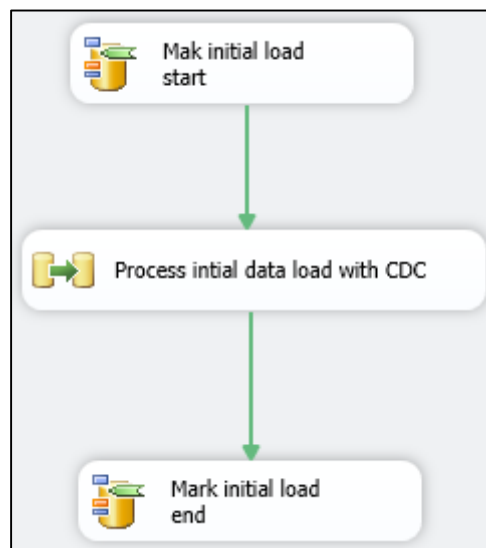


Figure 90: Initial Load

We will now see how to configure the CDC Control tasks. The data flow that will process the data will be explained later in this chapter.

The first step is to configure the **Mark initial load start** task. So, let's double-click it to open its configuration options.

Figure 91: Configuring the initial load start CDC Control task

In this window, you will need to make several configurations:

- **SQL Server CDC database ADO.NET connection manager:** Select an existing connection manager from the list or click **New** to create a new connection. The connection must be to a SQL Server database that is enabled for CDC and where the selected change table is located.
- **CDC control operation:** Select the operation to run for this task. All operations use the state variable that is stored in an SSIS package variable (which stores the state and passes it between the different components in the package).
- **Variable containing the CDC state:** Select the SSIS package variable that stores the state information for the task operation.
- **Automatically store state in a database table:** Select this check box for the CDC Control task to automatically handle loading and storing the CDC state in a state table contained in the specified database connection manager for the database in which the state is stored.
- **Connection manager for the database where the state is stored:** This connection is to a SQL Server database that contains the state table.
- **Table to use for storing state:** This is the SQL table that will store the CDC state. You won't need to handle it since it's an internal table used by the CDC mechanism.

- **State name:** Name of the state table to be used for storing the CDC state. You can create a new one using the **New** button.

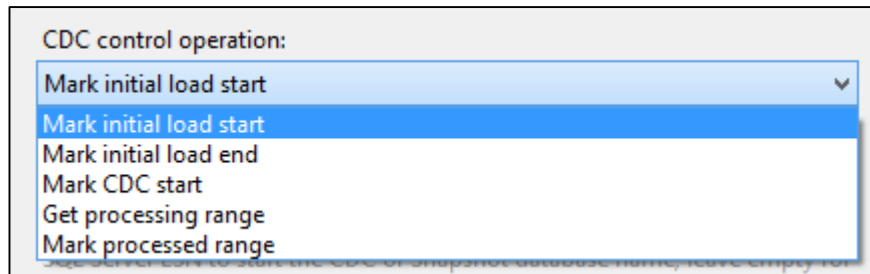


Figure 92: Operation Types

As you can see in the previous figure, the CDC Control task has several options for control operations:

- **Mark initial load start**—This option records the first load starting point. This operation is used when executing an initial load from an active database without a snapshot. It is invoked at the beginning of an initial-load package to record the current log sequence number (LSN) in the source database before the initial-load package starts reading the source tables. This requires a connection to the source database.
- **Mark initial load end**—This option records the first load ending point. This operation is used when executing an initial load from an active database without a snapshot. It is invoked at the end of an initial-load package to record the current LSN in the source database after the initial-load package finishes reading the source tables. This LSN is determined by recording the time when this operation occurred and then querying the `cdc.lsn_time_mapping` table in the CDC database, looking for a change that occurred after that time.
- **Mark CDC start**—This option records the beginning of the CDC range. This operation is used when the initial load is made from a snapshot database or from a quiescence database. It is invoked at any point within the initial-load package. The operation accepts a parameter which can be a snapshot LSN, a name of a snapshot database (from which the snapshot LSN will be derived automatically), or it can be left empty, in which case the current database LSN is used as the start LSN for the change processing package. An important note about this operation is that it can be used instead of the Mark Initial Load Start and Mark Initial Load End operations.
- **Get processing range**—This option retrieves the range for the CDC values. This operation is used in a change processing package before invoking the data flow that uses the CDC Source data flow. It establishes a range of LSNs that the CDC Source data flow reads when invoked. The range is stored in an SSIS package variable that is used by the CDC Source during data flow processing.
- **Mark processed range**—This option records the range of values processed. This operation is used in a change processing package at the end of a CDC run (after the CDC data flow is completed successfully) to record the last LSN that was fully processed in the CDC run. The next time `GetProcessingRange` is executed, this position determines the start of the next processing range.

Save and close this task editor. Open the **Mark initial load end** task to tell the CDC mechanism that this initial load has ended. If you don't mark this end state and you try to process the next

range of data, SSIS will give you an error. The configuration of this task only differs from the previous one on the operation type.

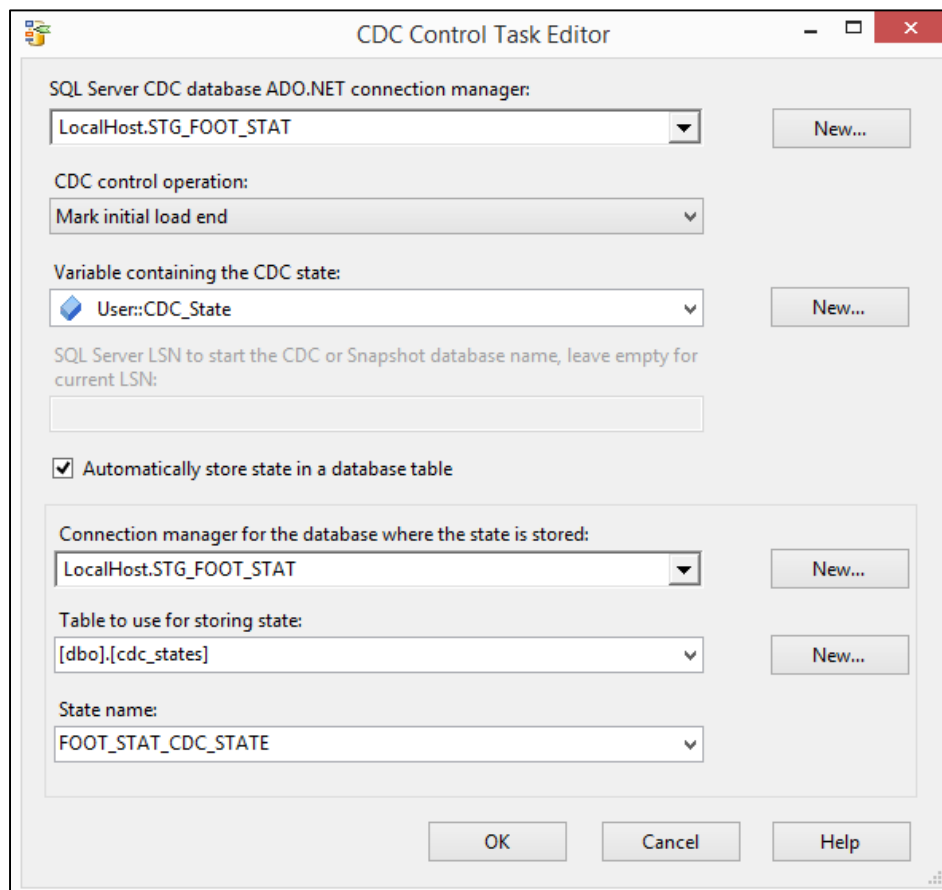


Figure 93: Configuring the Mark initial load end CDC Control task

To process incremental updates, you just need to develop new packages or reuse the current ones and change their operation type as follows:

- Mark initial load start > Get processing range
- Mark initial load end > Mark processed range

Create the CDC Data Flow

After developing the control flow, we need to develop the data flow that is going to process the changed data. To do so, start by opening the data flow, and then drag and drop a new CDC source object into it.

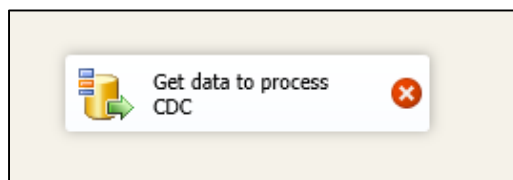


Figure 94: CDC Source

The next step is to configure this CDC source. This configuration involves setting the following properties:

- **ADO.NET connection manager:** The connection must be to a SQL Server database that is enabled for CDC and where the selected change table is located.
- **CDC-enabled table:** Select the CDC source table that contains the captured changes that you want read and feed to downstream SSIS components for processing.
- **Capture instance:** Select or type in the name of the CDC capture instance with the CDC table to be read. A captured source table can have one or two captured instances to handle seamless transitioning of table definition through schema changes. If more than one capture instance is defined for the source table being captured, select the capture instance you want to use here.
- **CDC processing mode:** Select the processing mode that best handles your processing needs.
- **Variable containing the CDC state:** Select the SSIS string package variable that maintains the CDC state for the current CDC context.

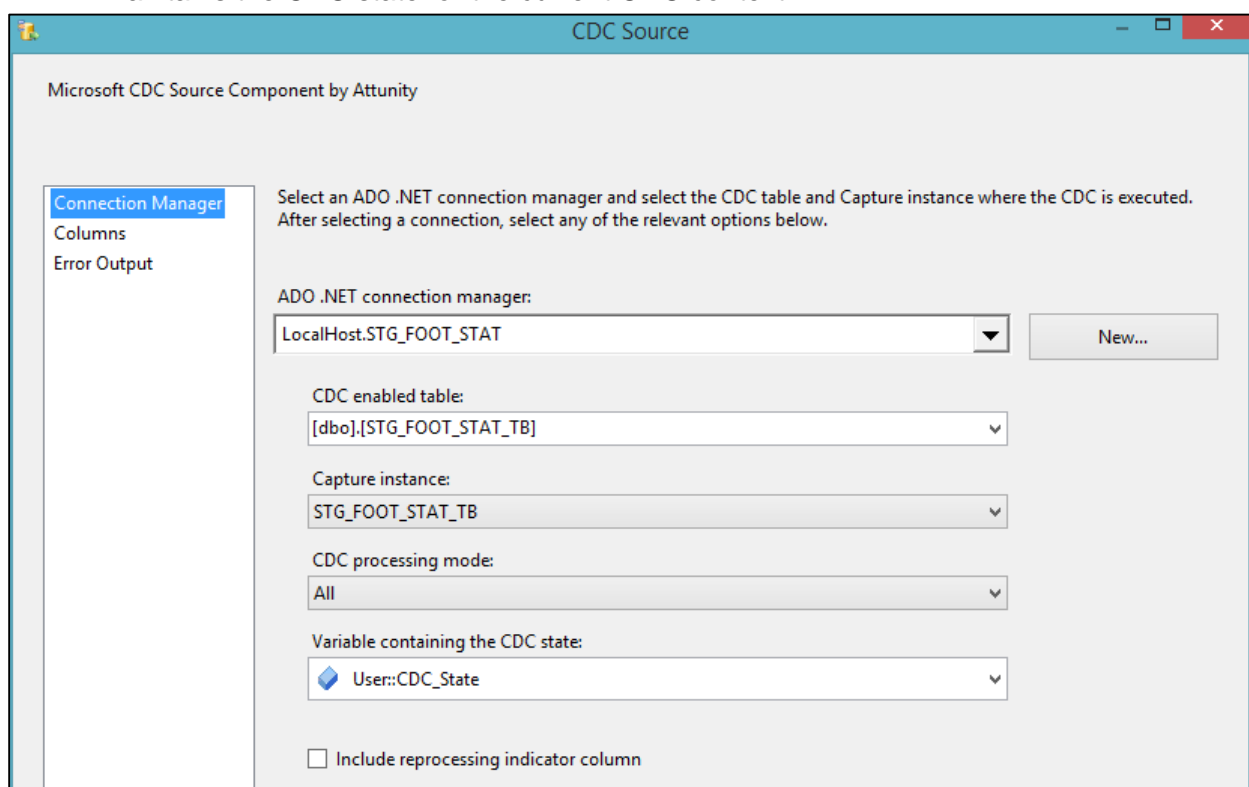


Figure 95: Configuring the CDC source

You can see the column that is being added to your data flow by selecting the **Columns** tab in this CDC source object.

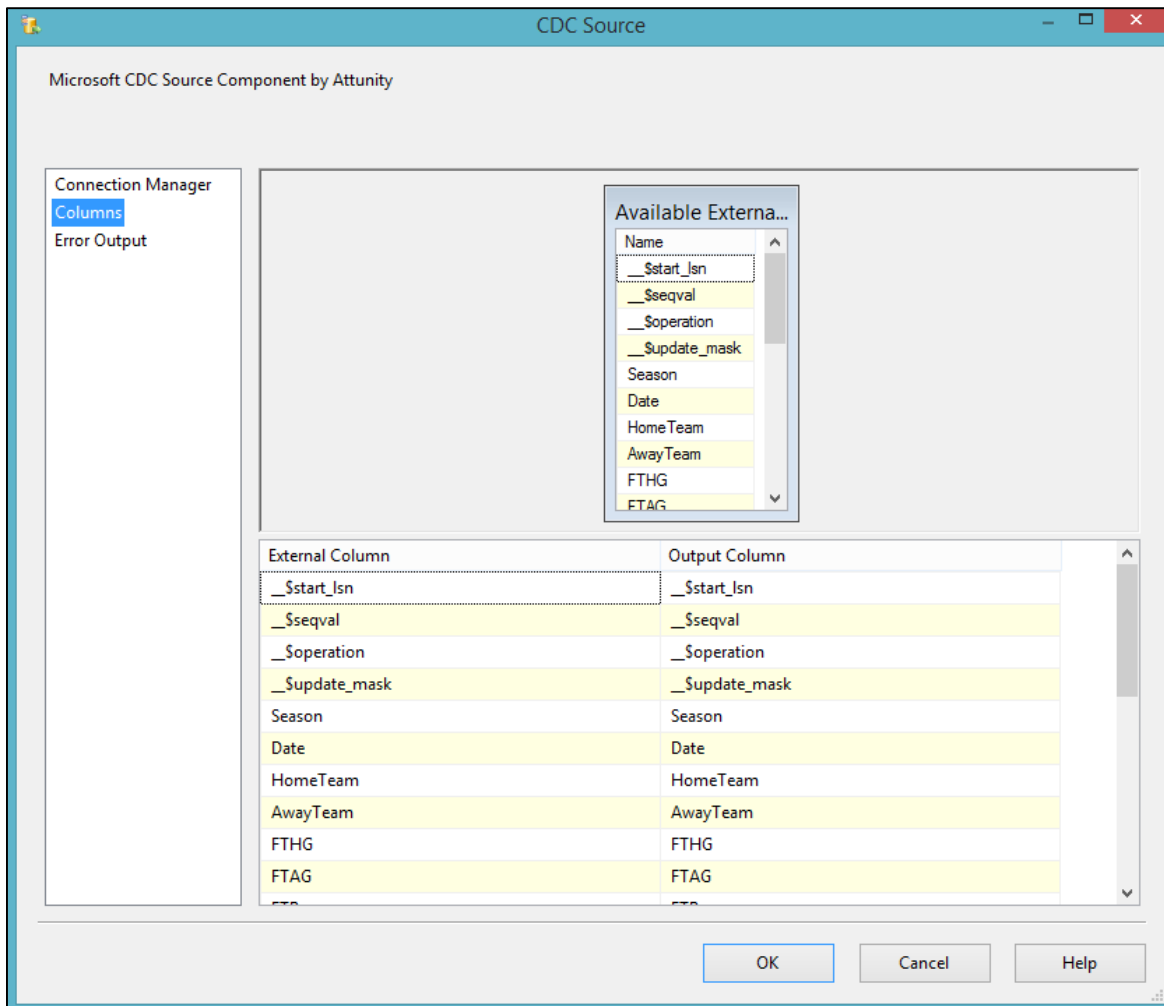


Figure 96: CDC Source Columns

The next step is to add a CDC splitter which will create three different paths according to the records' change operation type (insert, update, or delete). In this object, you don't need to make any configuration. Just connect the CDC source to this object and the data flow engine will resolve it for you.

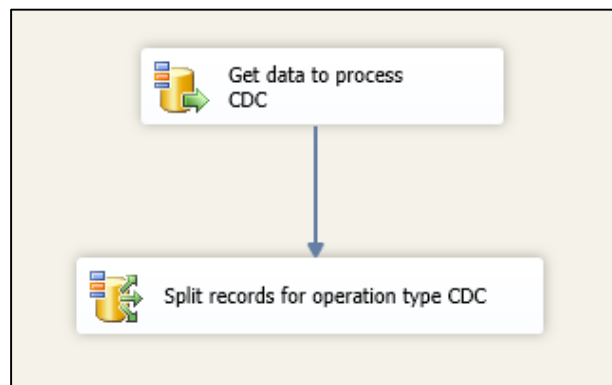


Figure 97: CDC Splitter

You can now make the logics you need by using the correct paths and changing its output when connecting the CDC splitter to another object.

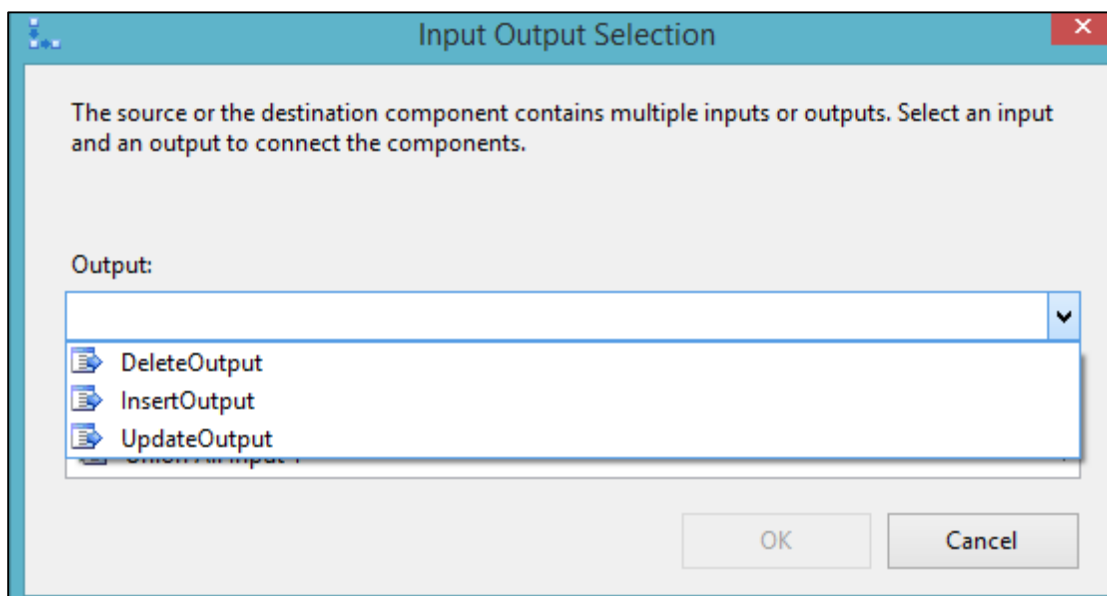


Figure 98: Configuring the Outputs

In this example, I am using a **Union All** object just to be able to see the data through data viewers. You can make whatever developments you wish.

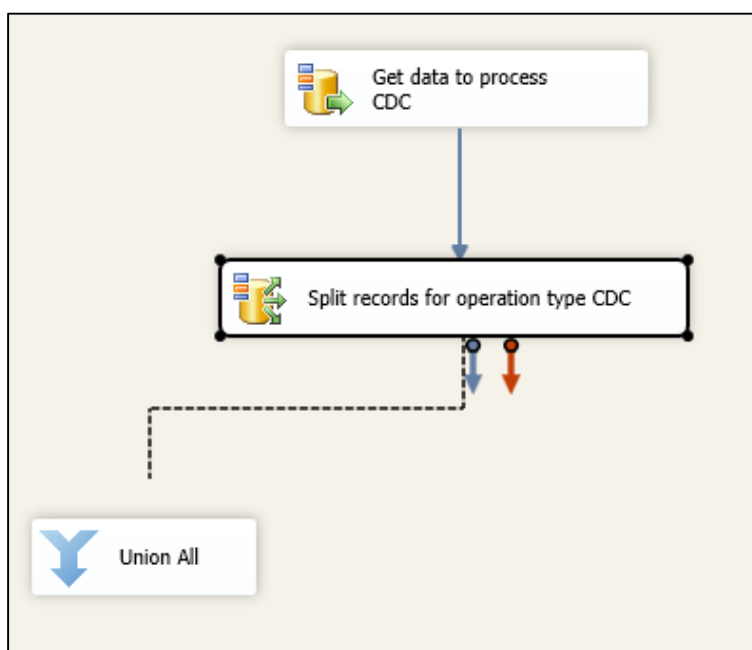


Figure 99: Developing Your Logics

And that's it! By using these components, you are able to process only changed records that are coming through your source systems. The following figure shows my final result.

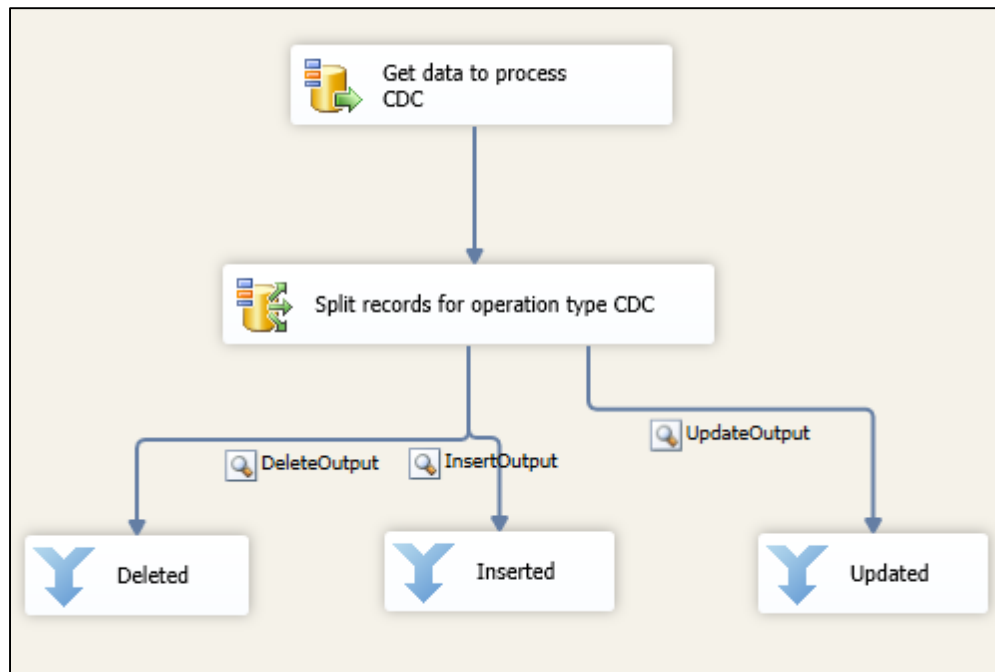


Figure 100: Final Data Flow

If you run this data flow, the result will be shown in the data viewers we have configured.

UpdateOutput Data Viewer at Process data range with CDC

Detach Copy Data

__start_jsn	__seqval	__...	__update_mask	Season	Date	HomeTeam
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2007/...	2008-0...	West Ham
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2007/...	2008-0...	Sunderland
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2007/...	2008-0...	Tottenham
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2007/...	2008-0...	Man City
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2007/...	2008-0...	Everton
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2007/...	2008-0...	Newcastle
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-0...	Wigan
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-0...	Man City
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-0...	Stoke
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-1...	Middlesbrough
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-1...	Hull
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-1...	Blackburn
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-1...	West Brom
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-1...	Bolton
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-1...	Everton
0x00 0x00 0x00 0x4B 0x...	0x00 0x00 0x00 0x4B 0x...	4	0x00 0x00 0x08	2008/...	2008-1...	Fulham

Attached Total rows: 0, buffers: 0 Rows displayed = 247

Figure 101: Data Flow Result

Chapter 6 Deploying the Packages

Introduction

After you have finished all the developments and checked to make sure that everything works as it should, it's time to deploy your solution to the server. For doing so, Integration Services 2012 comes with two deployment models:

- Project deployment model
- Package deployment model

The first model, project deployment, is a new feature of SSIS 2012. It uses an SSIS catalog that should be created in a SQL Server instance to store all the objects in a deployed project. When you deploy a project using this model, it will aggregate all the objects inside your project and deploy them as one to the server, creating an .ispac file.

In the second case, the package deployment model (which has been used since SSIS 2005), the packages aren't deployed as one; they are either treated separately where you can deploy one package at a time if you want, or a deployment utility can be used to deploy all of them.

Although one of the concerns related to these deployment models is the atomicity of the project objects, there are some other differences between the project and package deployment models. To better understand the differences, take a look at the following table.

Table 11: Project deployment model versus the package deployment model

	Project Deployment Model	Package Deployment Model
Unit of Deployment	Project	Package
Assignment of Package Properties	Parameters	Configurations
File output	Project deployment file (.ispac extension).	Packages (.dtsx extension) and configurations (.dtsConfig extension) are saved individually to the file system.
Deployment	A project containing packages and parameters is deployed to the SSISDB catalog on an instance of SQL Server.	Packages and configurations are copied to the file system on another computer. Packages can also be saved to the MSDB database on an instance of SQL Server.

	Project Deployment Model	Package Deployment Model
Package Validation	Projects and packages in the catalog can be validated on the server before execution. You can use SQL Server Management Studio, stored procedures, or managed code to perform the validation.	Packages are validated just before execution. You can also validate a package with dtExec or managed code.
Running and Scheduling Packages	Packages are run in a separate Windows process. SQL Server Agent is used to schedule package execution.	Same as project deployment model.
Event Handling	During execution, events that are produced by the package are captured automatically and saved to the catalog.	During execution, events that are produced by a package are not captured automatically. A log provider must be added to the package to capture events.
Environment Specific variables	Environment-specific parameter values are stored in environment variables.	Environment-specific configuration values are stored in configuration files.
CLR Integration	CLR integration is required on the database engine.	CLR integration is not required on the database engine.

Now that we understand the main differences between these two models, let's learn how we can use them so that our projects can go into a production environment.

Project Deployment Model

As I have explained before, this is the new deployment model that SSIS 2012 has introduced. It's very important to keep in mind that, if you want to use the new parameters component, your projects should be deployed using this model. So, to deploy a project using this model, you must start by creating a new SSIS catalog which will store your deployed projects. To do so, you should open SQL Server Management Studio. After connecting to the SQL Server instance into

which you want the project to be deployed, locate the **Integration Services Catalogs** folder in the Object Explorer.

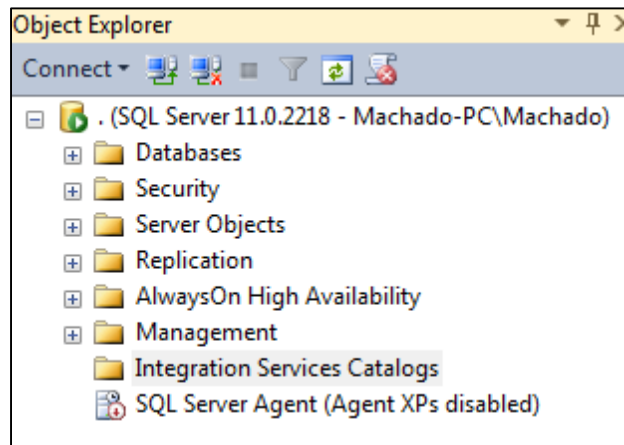


Figure 102: Integration Services Catalogs folder

After locating the **Integration Services Catalog** folder in the Object Explorer, right-click it and select **Create Catalog**. This will open a new **Create Catalog** window in which you only need to select the **Enable CLR Integration** check box to enable CLR integration; otherwise you will not be able to create the catalog and configure your catalog password. Don't forget that this will store your developed objects with all parameters included. Since you don't want anyone to access them and ruin your developments, being able to set a password is an important feature.

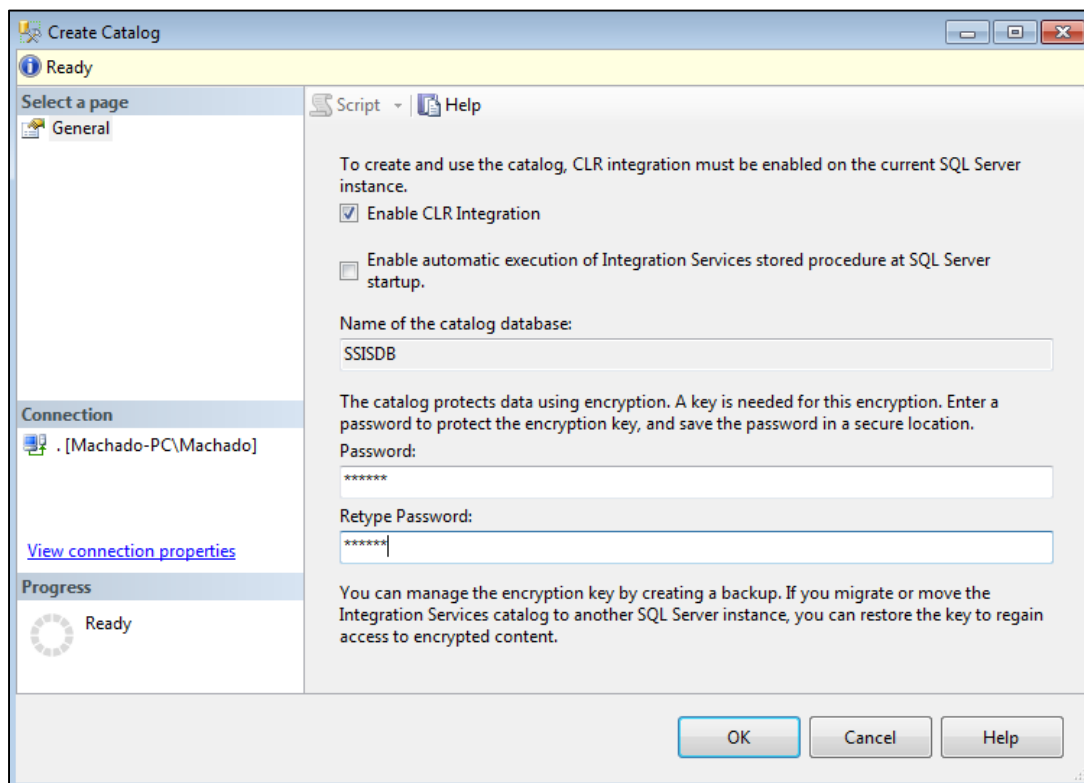


Figure 103: Creating a new catalog

After setting all the required property values, click **OK**. Now you are able to see a new catalog inside your Integration Services Catalog folder. If you expand the folder, you'll see the new catalog folder inside.

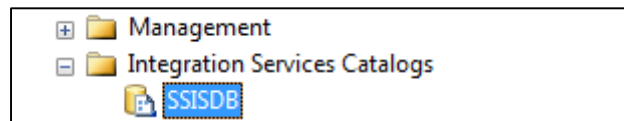


Figure 104: New Catalog Created

Now that you have a catalog, you are able to deploy your project to it. To start the deployment, go to your SQL Server Data Tools and open the project to be deployed. Once you have the project open, right-click your project node (root) and select **Deploy** as shown in the following figure. This will start the deployment tool.

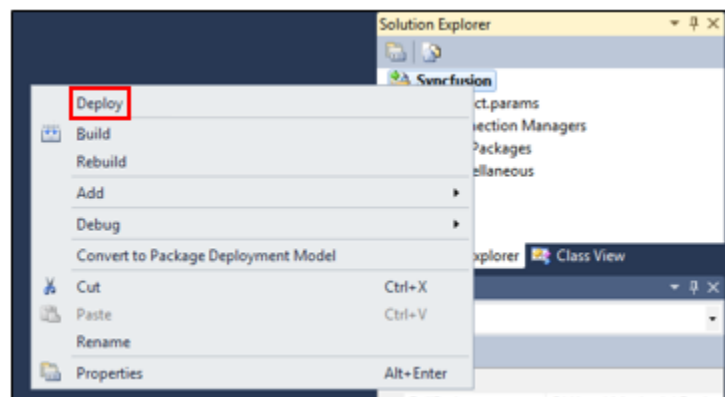


Figure 105: Starting the deployment tool

Once the tool starts, you will need to complete a wizard that will assist you in the deployment process. This wizard has the following four steps; however, the most important ones are Select Source and Select Destination:

- Select Source
- Select Destination
- Review
- Results

In the Select Source screen, you need to specify the project you want to deploy (which will be our example) and if you want another SSIS catalog coming from another server. In the Select Source screen, select the project deployment file option and SSIS will automatically set the path of the project to the one that is currently open.

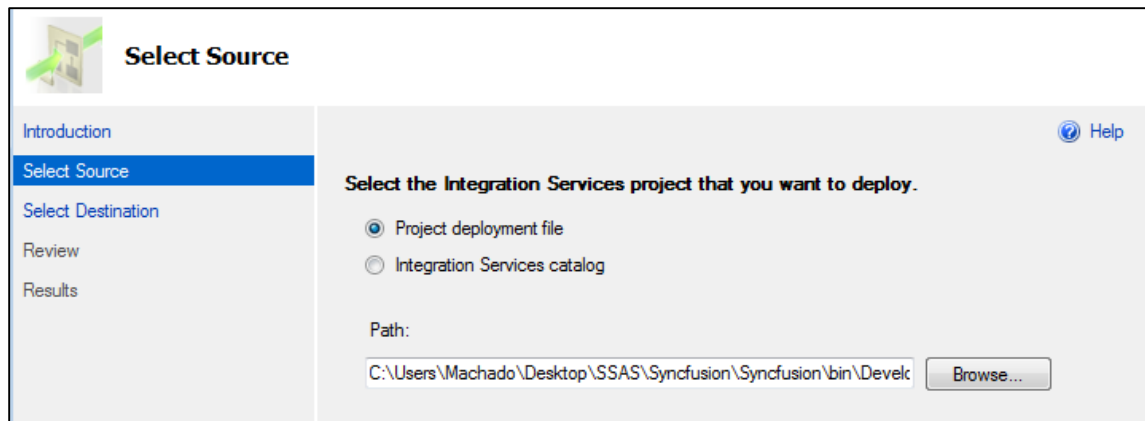


Figure 106: Selecting a project to deploy

The next step is to select the destination. In the Select Destination screen, you need to specify the server into which you want to deploy the project and the path to the SSIS catalog folder to which it will be deployed. Because an SSIS catalog may store multiple SSIS projects, you need to create a new folder for this new project. If you were redeploying a solution, you would select an existing one.

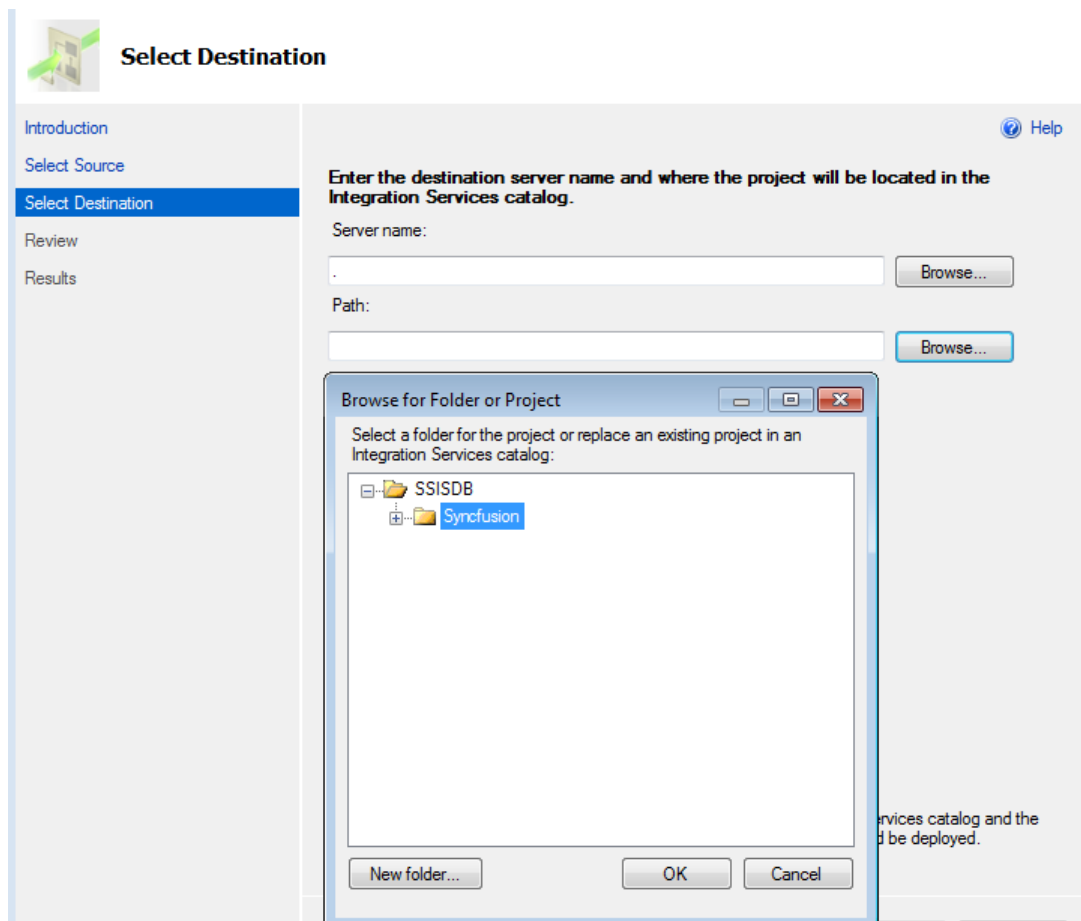


Figure 107: Select Destination

And that's it! The final screen allows you to validate the settings you have just set in the Select Source and Select Destination screens. If everything is as it should be, click **Deploy** to finish the deployment. After the deployment is over, go to the Management Studio and expand the **Integration Services Catalogs** folder. You should now see your project deployed. If you want to know how to view and change your project parameters, revisit the [Variables, Expressions, and Parameters](#) chapter.

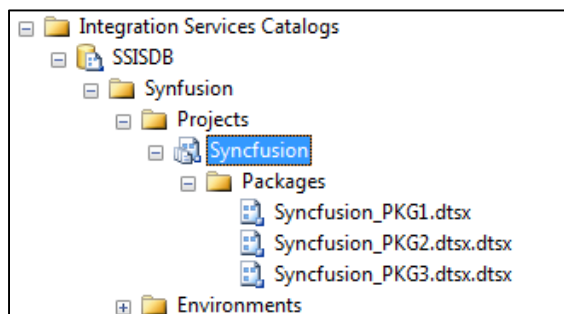


Figure 108: Deployed Project

Package Deployment Model

The package deployment model is a legacy model for package deployment that it is based on packages instead of projects. This means that each package deploys independently. In my personal opinion, if you are developing new SSIS projects, you should take the project model approach. However, the package deployment model can still be useful for legacy projects that you need to maintain, for example. Once again, be aware that you cannot use parameters in this project model.

The first step is to change your deployment model inside your project. To do this, go to SQL Server Data Tools and open the project that has the packages you want to deploy. Once you have the project open, right-click your project node (root) and select **Convert to Package Deployment Model** as shown in the following figure. Before you do that, there are some particular things you should be aware of: You cannot have any project-level objects such as connection managers; instead, you can use data sources but you need to manually add one to each package you have. Another thing to note is that you do not have parameters to work with; instead, you need to use package configurations, once again adding them manually to each package.

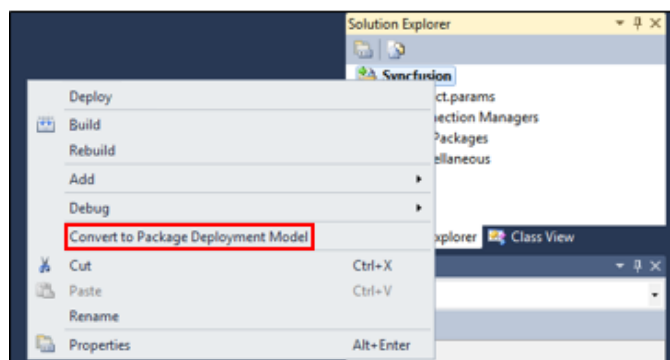


Figure 109: Convert to Package Deployment Model option

Once you finish this step, your project will be under the package deployment model. Now it's time to deploy our individual packages. To do this, open the package you want to deploy. Next, click an empty area in the control flow and then open the **File** menu. Select **Save Copy of [package name]** as shown in the following figure. It's important that you are aware that you can deploy all of your packages at once by using a deploy manifest. However, like the project deployment model, this is an "all-or-nothing" method, which means you cannot select which packages you want to deploy; it will deploy all of them. If you want to learn more about this method, use this [reference](#) on the Microsoft TechNet website. On that page it will explain how to create a deployment utility (manifest).

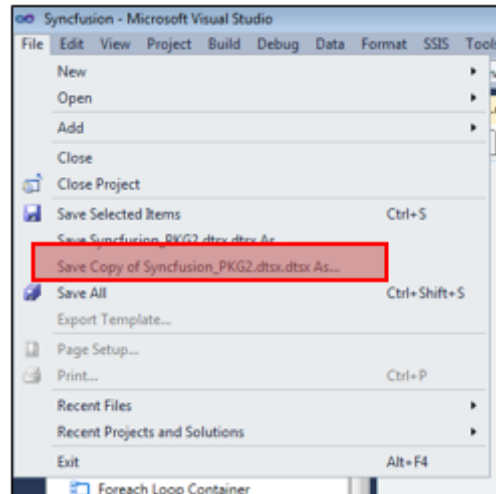
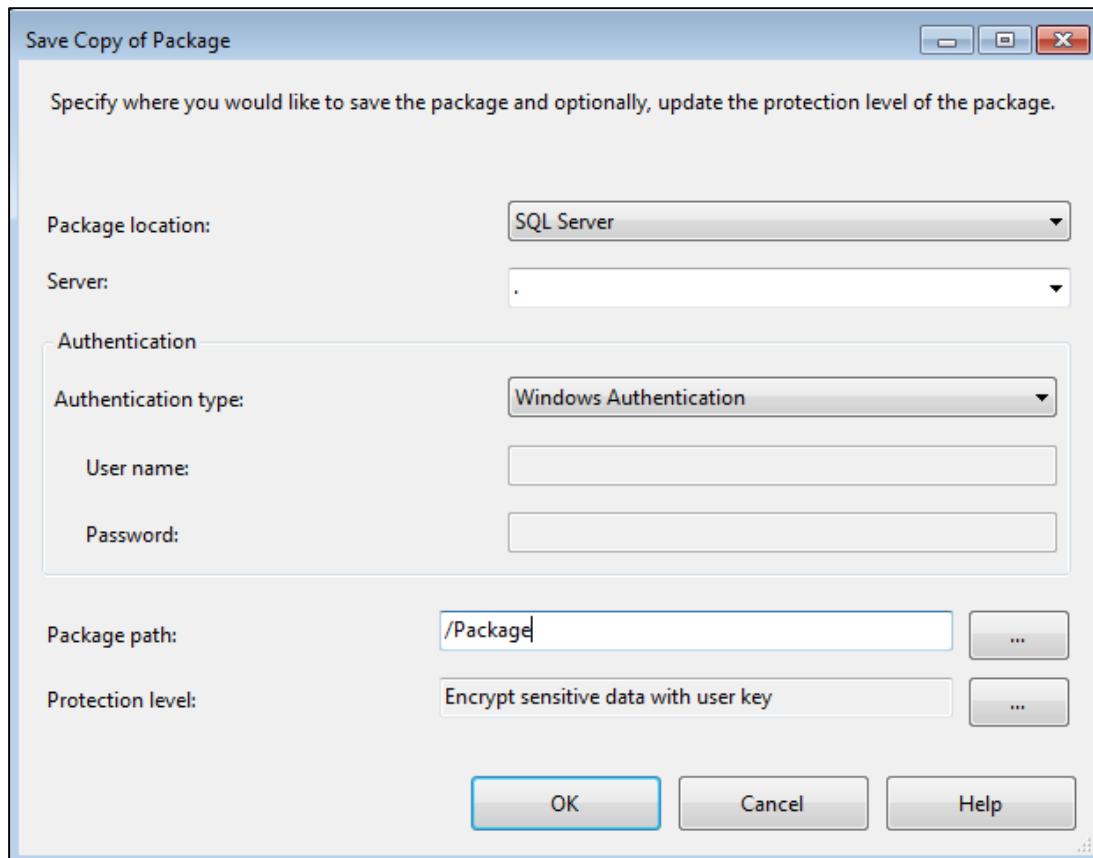


Figure 110: Saving a copy of a package

Now comes the last step. We are going to deploy the package to a SQL Server instance. In the Save Copy of Package screen, select **SQL Server** as the **Package location** and set the server name and authentication. Last but not least, select the **Package** root folder as the **Package path**.



Save Copy of Package

Specify where you would like to save the package and optionally, update the protection level of the package.

Package location: SQL Server

Server: .

Authentication

Authentication type: Windows Authentication

User name:

Password:

Package path: /Package

Protection level: Encrypt sensitive data with user key

OK Cancel Help

Figure 111: Saving to SQL Server

Now you can see the package if you connect to your Integration Services server type using the SQL Server Management Tools. It will be under **Stored Packages** in the **MSDB** folder.



Connect to Server

Microsoft SQL Server 2012

Server type: Integration Services

Server name: .

Authentication: Windows Authentication

User name: Machado-PC\Machado

Password:

☐ Remember password

Connect Cancel Help Options >>

Figure 112: Connecting to SSIS 2012