Evaluation Scheme:

|  |  |
| --- | --- |
| Log of the source files  processed | 5 |
| Customer file | 5 |
| Archive File | 5 |
| Transfer Cust Pkg | 5 |
| Master pkg | 5 |
| Dashboard | 5 |
| Objectives | 10 |
| Conclusion | 10 |
| Overall Report | 10 |
| Total | 60 |

Notes for completing assignments:

In this assignment, you are going to implement 6 different mechanisms of the ETL process in building and deploying 3 packages. Please provide screenshots for the steps and add appropriate explanatory notes using the information provided in the lectures.

Please save the three CSV files in a backup directory., they will be used as input to the ETL packages.

Your report should have at least the following sections:

Objectives:

Here you will list the anticipated benefits of the ETL mechanisms. ETL processes are used in many areas in addition to cleansing and importing data into the data warehouse. You are expected to research and document its importance to large computer environments where the daily processes depend on the overnight data cleansing processes.

Each “time for action” section:

Summarize the steps and provide the screen shots of the intermediate processes and summarize the results.

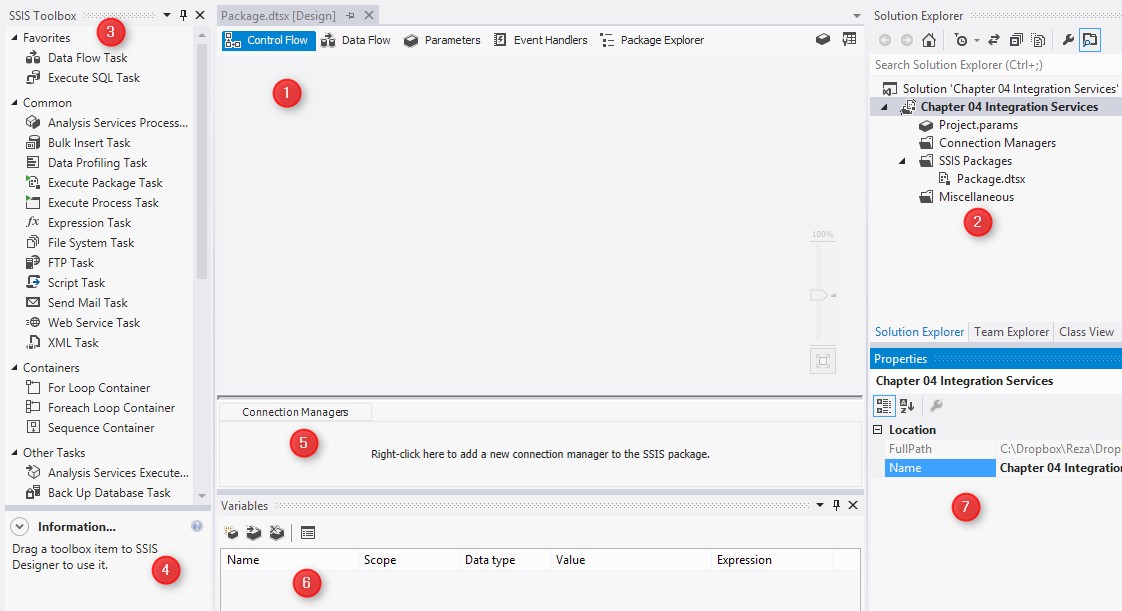
Conclusion:

Your conclusion should reflect your experience performing the ETL exercise: what was easy, difficult and obscure.

Time for action – creating your first SSIS project

In this example, we will go through an SSIS project and packages to explore the functionality and structure of these objects. We will create an SSIS package and explore the development environment features in **SQL** **Server** **Data** **Tools** (**SSDT**) for Integration Services. Perform the following steps to create a project:

1. Open SSDT and create a new project of type **Integration Services Project**. Name the project **Chapter 04 Integration Services**.
2. When the new project opens, you will see **Solution** **Explorer**, **Package** **Designer**, and **SSIS** **Toolbox**:



What just happened?

As you've just seen, the SSDT has a special design for the development of SSIS packages. The preceding screenshot shows each part of SSDT, and the description of each part is given as follows:

* **Package** **Designer**: In the preceding screenshot, the package designer shows the **Control** **Flow** button.
* **Solution** **Explorer**: The SSIS projects contain three folders. **SSIS** **Packages** is used as a location for SSIS package files. **Connection** **Managers** is used for the project's shared connection managers, which are accessible from all packages in this project. **Miscellaneous** is used for storing any related file, such as documents, to this project and **Project** **Parameters**, which we will discuss in the following sections.
* **SSIS** **Toolbox**: This toolbox can be different if the package designer is in **Data** **Flow**. The preceding screenshot shows **SSIS** **Toolbox** for **Control** **Flow**. This toolbox shows built-in tasks and components that we can drag-and-drop or double-click in order to add them into the package.
* **Information** **box**: This tab provides more information about each selected item in the toolbox.
* **Connection** **Managers**: This tab is used as a location for the package's connection managers. Connection managers of a package can be used only in that package, and other packages cannot access it. Connections to different data sources can be made using connection managers such as OLE DB, ODBC, flat file, and so on.
* **Variables**: The local variables of the package can be managed in this tab.
* **Properties**: This is the default window in SSDT that provides the listed properties of the selected object (such as a package, task, container, and so on) for the developer to view and change the property's values.

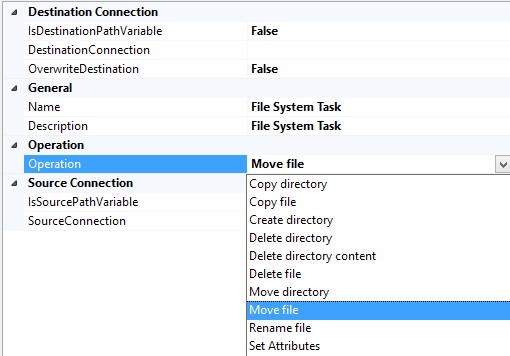
The Control Flow tab

The flow of the execution in the SSIS package will be organized in the **Control** **Flow** tab. **Control** **Flow** is the location where we add tasks (to do an operation) and **Containers** (containers of the tasks), and set the order of execution with **Precedence** **Constraint**.

Time for action – working with Control Flow tasks

In this example, we will start working with the **Control** **Flow** task, use **File** **System** **Task** to move a file, and also use **Execute** **SQL** **Task** to interact with a database and write a log entry. We will also work with **Precedence** **Constraint** to define the priority and order of execution of items in the **Control Flow** window. Perform the following steps to explore the options in the **Control Flow** window:

1. Create a package-level variable in the **Variables** pane, name it **FilePath** with a string data type, and set its path to the **Customers\_1.csv** file.
2. Drag-and-drop a **File** **System** **Task** from the SSIS toolbox into the **Control** **Flow** tab.
3. Double-click on the **File** **System** **Task** in the **Package** **Designer**; the **File** **System** **Task** **Editor** will be opened.
4. Set the **Operation** option as **Move File**, set **IsSourcePathVariable** to **true**, and choose **User::FilePath** as **SourceVariable**:



1. Keep **IsDestinationPathVariable** as **False** and create a destination connection to an archive folder (a folder that we use for the purpose of archiving files). Set **OverwriteDestination** to **True**.
2. Name the **File** **System** **Task** as **Archive** **File** and click on **OK**.

Now, we want to write an entry into a log table after we successfully moved the file.

1. Firstly, create the log table with the following script in the **Packtpub\_BI \_2014** database:

**CREATE TABLE [dbo].[LogSourceFile](**

**[ID] [int] IDENTITY(1,1) NOT NULL,**

**[FileName] [varchar](500) NULL,**

**[LoadDateTime] [datetime] NULL,**

**CONSTRAINT [PK\_LogSourceFile] PRIMARY KEY CLUSTERED**

**(**

**[ID] ASC**

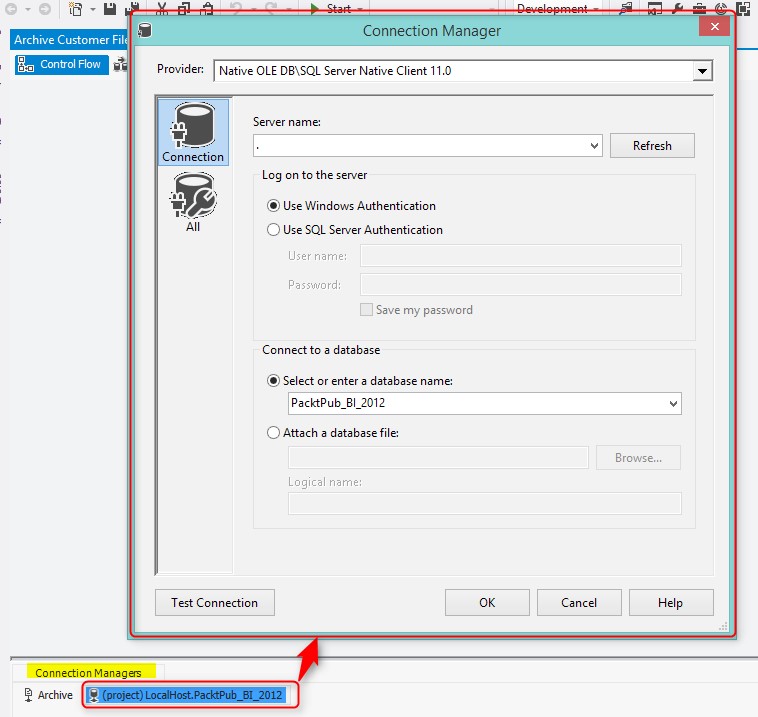
**)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF,**

**IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON,**

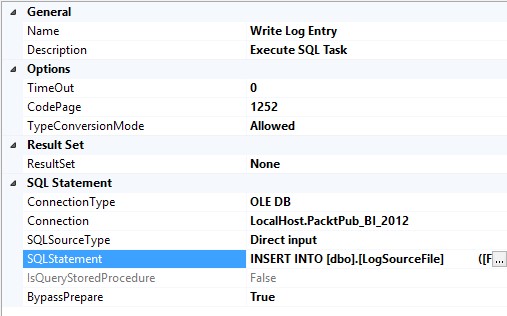
**ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]**

**) ON [PRIMARY]**

1. In the SSIS project, under the **Solution** **Explorer** tab, right-click on the **Connection** **Managers** folder and create a new connection manager.
2. Choose the type of connection manager as **OLE** **DB** and then set up a connection to the **Packtput\_BI\_2014** database on your SQL Server instance. After creating the connection, you will see that the **Connection** **Managers** pane in **Package** also shows that connection but with a prefix (project), which means that this connection is a shared project connection.
3. Drag-and-drop an **Execute** **SQL** **Task** from **SSIS** **Toolbox** to the **Control** **Flow** tab. Rename this task as **Write** **Log** **Entry**.



1. Click on the **Archive** **File** task; you will see a green arrow out of this task. Drag-and-drop this green arrow to the **Write** **Log** **Entry** task. This arrow is called **Precedence** **Constraint**.
2. Double-click on the **Write** **Log** **Entry** task. In the **Execute SQL Task Editor** window, keep the **ConnectionType** as **OLE** **DB**, and choose the **Packtpub\_BI\_2014** database as a connection manager for it, as shown in the following screenshot:



1. Write the following code in the **SQL** **Statement** property:

**INSERT INTO [dbo].[LogSourceFile]**

**([FileName]**

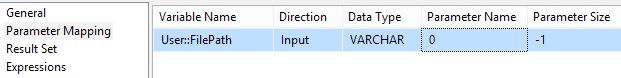
**,[LoadDateTime])**

**VALUES (?**

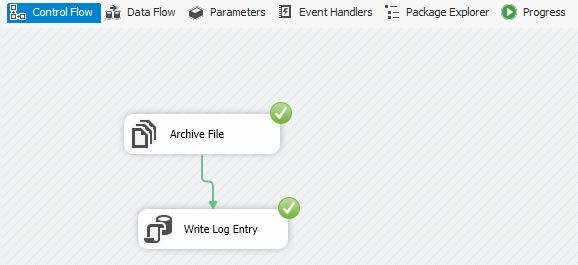
**,getdate())**

1. The statement that we wrote in the previous step asks for a parameter, which is the filename; we should set up this parameter in the **Parameters** **Mapping** tab of the **Execute** **SQL** **Task**.
2. Go to the **Parameter** **Mapping** tab of the **Execute** **SQL** **Task** **Editor** and click on **Add**.
3. Set the **Variable Name** option to the **User::FilePath** variable, set **Data** **Type** to

**VARCHAR**, and set **Parameter Name** as **0**, as shown in the following screenshot:



1. Close the **Execute** **SQL** **Task** **Editor** and run the package by pressing the *F5* key or with the run icon in the SSDT.
2. You will see the execution status of each task with icons, which shows a running or successful state. You can also view the execution result in the **Progress** tab of the **Package** **Designer**. This tab shows information about each task's status along with possible warnings or error messages. The following screenshot shows the execution of the package along with its entire schema:



1. Rename the package **Archive** **Customer** **File**.

What just happened?

Congratulations, you've just made your first SSIS package! You've learned how to work with

**Control** **Flow** tasks in this example. However, this example also revealed some other parts of SSIS, such as variables, connection managers, and precedence constraints. We will now dive deep into these parts in these following sections.

##### **Variables**

In the first step of this example, we created a package variable. Variables are very important when you want to pass a value between multiple tasks in a package. Variables can have some basic types such as **String**, **Int32**, **Int64**, **Boolean**, and so on. In this example, we used a variable to store the source file's full path.

##### **File System Task**

In steps 2 and 3, we used a **File** **System** **Task**. The **File** **System** **Task** is useful for file and directory operations. This task uses the System.IO classes to work with files and directories. Some of the operations that we can perform with this task are **Copy file**, **Move file**, **Delete file**, **Create directory**, **Set attributes**, and so on. In steps 4 and 5, we set the operation and the operation's properties such as source and destination.

All the **Control** **Flow** tasks in SSIS have a **Task** **Editor** window, similar to what you've seen in **File** **System** **Task** **Editor**; this editor is a GUI that is used to set properties of that task. For the **File** **System** **Task**, this GUI provided an opportunity to set the operation, source file, and destination directory. **Source** and **Destination** in the **File** **System** **Task** can be an exact connection to a file or folder (as you can see in step 5, we used a direct connection to the destination archive folder), or they can be set with **Variables** (as we've set the source file path with the **User::FilePath** variable in step 4). Working with variables provides us with the ability to make our tasks more dynamic than using static connections and values, because every time that variable's value changes, tasks will work with new values.

##### **Control Flow tasks**

As you can see in the SSIS toolbox, there are many control flow tasks, and each of them is suitable for a specific use case in real-world scenarios. The following table briefly explains each task's main duty with a short description (all the tasks are not listed):

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | | **Description** | |
| **Execute** **SQL** **Task** | | This task executes SQL statements on databases, and it may return results | |
| **File** **System** **Task** | | This task performs file and directory operations such as copy, move, delete, and others | |
| **Data** **Flow** **Task** | | This task performs extensive data transfer operations | |
| **FTP** **Task** | | This task sends and receives files between local and remote FTP locations | |
| **Send** **Mail** **Task** | | This task sends e-mails | |
| **Script** **Task** | | This task runs .NET scripts | |
| **Web** **Service** **Task** | | This task consumes a web service and fetches the result into a file and a variable | |
| **XML** **Task** | | This task performs XML operations such as XSLT, Xpath, and XML validation | |
| **Execute** **Process** **Task** | | This task runs an executable file or application (with or without arguments) | |
| **Execute** **Package** **Task** | | This task executes other SSIS packages | |
| **Task** | | **Description** | |
| **Expression** **Task** | | This task resolves the result of an SSIS expression into a variable | |
| **Bulk** **Insert** **Task** | | This task performs bulk insert operations and loads data from a flat file into a database | |
| **Data** **Profiling** **Task** | | This task performs data profiling for a database connection based on existing data rows | |
| **CDC** **Control** **Task** | | This task manages working with the change data capture feature in order to get informed about the changes | |
| **WMI** **Event** **Watcher** **Task** | | **Windows** **Management** **Instrumentation** (**WMI**) events can be watched, for example, watching a folder for new upcoming files | |
| **WMI** **Data** **Reader** **Task** | | This task reads WMI information such as the number of drives, disk space, and others | |
| **Analysis** **Service** **Processing** **Task** | | This task processes an SSAS object such as a database, cube, dimension, and partition | |
| **Analysis** **Service** **Execute** **DDL** **Task** | | This task executes an XMLA command on the SSAS engine, such as synchronizing SSAS databases, backing up an SSAS database, and others | |

In steps 9 and 10, we created a shared connection manager. Shared connection managers can be created in **Solution** **Explorer**, and they are visible for all packages in that project. In SSIS, **Connection** **Managers** are of different types. The **OLE** **DB** connection, **ADO.NET**, and **ODBC** connections are used for connecting to databases. The **Flat** **File** connection is used to work with text files such as CSV files. There are many more connection types that you can explore; more information about each connection manager is available at [**http://msdn. microsoft.com/en-us/library/ms140203%28v=sql.90%29.aspx**.](http://msdn.microsoft.com/en-us/library/ms140203%28v=sql.90%29.aspx)

##### **Precedence Constraints**

Steps 11 and 12 show us how to set the order of execution with **Precedence** **Constraints**. A green arrow (which is the default constraint) indicates successful execution of the previous task. So when we connect **Archive** **File** to **Write** **Log** **Entry** with a green arrow, this means that the second file will only be executed if the first task executes successfully. There are other constraints such as Completion and Failure as well. Using the failure precedence constraint is good for error handling. An example of error handling will be sending mail to the system administrator.

**Precedence** **Constraints** can also be empowered with expressions. Expression is a special language for an SSIS package, and it provides the ability to use some built-in functions and operators with variables and parameters to generate and calculate something dynamically at the package's run time. Using expressions in **Precedence** **Constraints** provides the ability to apply IF conditions and change the order of execution based on the result of the expression.

##### **Execute SQL Task**

In steps 13 and 14, we created an **Execute SQL Task** to write an entry log record into a database table. As you've seen in the **Control** **Flow** tasks table, **Execute** **SQL** **Task** is used to execute a SQL statement on the underlying database. The standard of writing a SQL statement is based on the database that we are connected to. If we are connected to SQL Server 2012 or a higher database, we can use specific functions such as **LAG** and **LEAD**. If we are connected to an Oracle database, we can use functions such as **DECODE**.

In step 14, we used a question mark sign in the SQL statement. The question mark is a parameter marker in the statement for OLE DB connections. We use the parameter marker to pass a variable value to the SQL statement through Execute SQL Task. In step 15, you can see that we used **0** as the parameter name. Parameter names in OLE DB connections start from 0, and this means that if we had another question mark, we would use 1 for the parameter marker of the second parameter in the **Parameter** **Mapping** tab.

The following table (sourced from the MSDN page at [**http://technet.microsoft. com/en-us/library/ms140355.aspx**)](http://technet.microsoft.com/en-us/library/ms140355.aspx) shows the signature of a parameter marker and parameter names for each connection manager in SSIS for **Execute** **SQL** **Task**:

|  |  |  |
| --- | --- | --- |
| **Connection Manager** | **Parameter Marker** | **Parameter Name** |
| ADO | ? | Param1, Param2, and so on |
| ADO.NET and SQLMobile | @<parameter name> | @<parameter name> |
| ODBC | ? | 1,2,3, and so on |
| Excel and OLE DB | ? | 0,1,2,3,… |

The **Execute** **SQL** **Task** used in this example executes an insert statement and doesn't return any result set. There are some cases in which **Execute** **SQL** **Task** returns a result set; in such cases, you can set the **ResultSet** property of this task to **Single Row** or **Full** **Result** **Set** and then set the output variable in the **Result** **Set** tab.

##### **Progress and execution results**

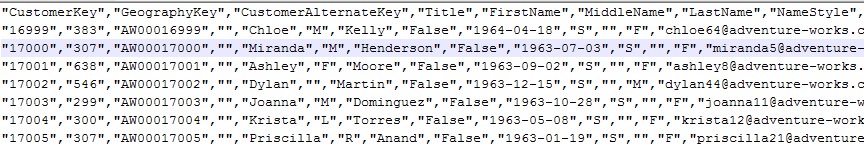
When you run a package in SSDT, you can see the execution status of the tasks with the help of icons that show whether the package is in the running mode, was successful, or failed. Also, there is a **Progress** tab in the run time (which will be named **Execution** **Results** when SSDT is not in the debug mode), which shows information about each task, container, and package. It also shows if there is a warning in the process of execution or if an error has occurred.

The Data Flow tab

One of the most useful and powerful tasks in **Control** **Flow** is **Data** **Flow** **Task**. The importance of this task is mostly because it functions as the major part of ETL, which extracts data from different sources, transforms it, and loads it into the destination. As this task plays an important role in an SSIS package and in real-world scenarios, there are few packages that don't contain Data Flow task; SSDT has a special designer for **Data** **Flow**, and the SSIS toolbox's items are different in the **Data** **Flow** designer.

|  |
| --- |
| Time for action – loading customer information from a flat file |
|  |
| into a database table with a Data Flow Task |

In this section, we will go through an example of fetching information from a CSV file that contains customer information. Then, we will apply a simple transformation to calculate the age of a customer based on their birthday, and finally, we will load the result set into a database table. For this example, we will use a **Data** **Flow** **Task** with **Flat** **File** **Source**, **Derived** **Column** **Transformation**, and **OLE** **DB** **Destination**. The following screenshot shows data rows in the **Customers\_2.csv** file, which need to be transferred to a database table:



Perform the following steps to load the information into a database table:

1. Create a new SSIS package; name it **Transfer** **Customer** **Data**.
2. Create the destination table for this example by running the following script on the **Packtpub\_BI\_2014** database:

**CREATE TABLE [dbo].[Customer](**

**[CustomerKey] [varchar](50) NULL,**

**[GeographyKey] [varchar](50) NULL,**

**[CustomerAlternateKey] [varchar](50) NULL,**

**[Title] [varchar](50) NULL,**

**[FirstName] [varchar](50) NULL,**

**[MiddleName] [varchar](50) NULL,**

**[LastName] [varchar](50) NULL,**

**[NameStyle] [varchar](50) NULL,**

**[BirthDate] [varchar](50) NULL,**

**[MaritalStatus] [varchar](50) NULL,**

**[Suffix] [varchar](50) NULL,**

**[Gender] [varchar](50) NULL,**

**[EmailAddress] [varchar](50) NULL,**

**[YearlyIncome] [varchar](50) NULL,**

**[TotalChildren] [varchar](50) NULL,**

**[NumberChildrenAtHome] [varchar](50) NULL,**

**[EnglishEducation] [varchar](50) NULL,**

**[SpanishEducation] [varchar](50) NULL,**

**[FrenchEducation] [varchar](50) NULL,**

**[EnglishOccupation] [varchar](50) NULL,**

**[SpanishOccupation] [varchar](50) NULL,**

**[FrenchOccupation] [varchar](50) NULL,**

**[HouseOwnerFlag] [varchar](50) NULL,**

**[NumberCarsOwned] [varchar](50) NULL,**

**[AddressLine1] [varchar](50) NULL,**

**[AddressLine2] [varchar](50) NULL,**

**[Phone] [varchar](50) NULL,**

**[DateFirstPurchase] [varchar](50) NULL,**

**[CommuteDistance] [varchar](50) NULL,**

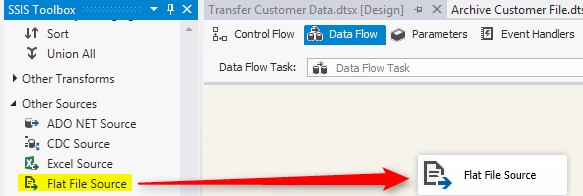
**[Age] [int] NULL**

**) ON [PRIMARY]**

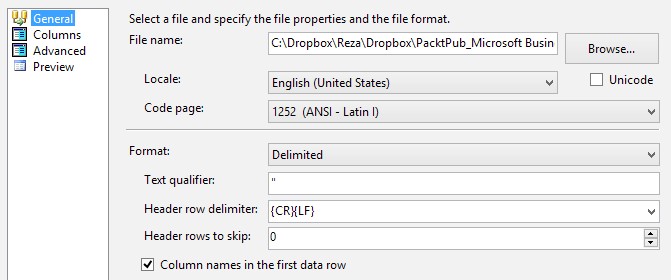
1. Drag-and-drop a **Data** **Flow** **Task** from **SSIS** **Toolbox** into the package designer.

Double-click on the task. You will be redirected to a new tab named **Data** **Flow**.

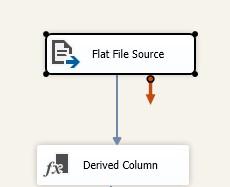
1. In the **Data Flow** tab, drag-and-drop a flat file source from **SSIS** **Toolbox** into the **Package** **Designer**. Then, double-click on the flat file source.



1. In **Flat** **File** **Source** **Editor**, create a new connection. After this step, the **Flat** **File** **Connection** **Manager** menu will be opened.
2. Browse for the **Customers\_2.csv** file, and leave the locale and code page configurations as is. Verify the **Format** field to be set as **Delimited**, set **Text** **qualifier** to **"** (double quote) as the next screenshot shows, and check the box that says **Column names in the first data row**:



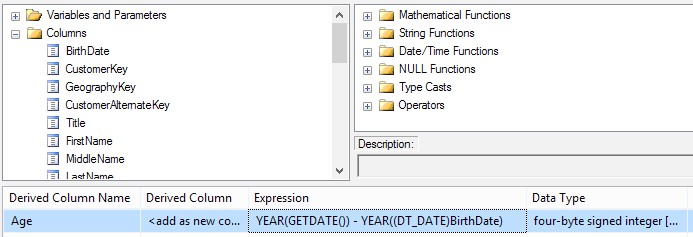
1. Go to the **Columns** tab and you will see how columns are recognized by the settings that we made in the **General** tab. You can also change the row and column delimiters in this tab.
2. The **Advanced** tab shows detailed information about each column; you can set the data type, length, and some other properties for each column in that tab. Finally, you can view the data rows as they are processed in the flat file source's connection manager in the **Preview** tab. Do not change anything in the **Advanced** tab and click on **OK**.
3. In the **Flat** **File** **Source** **Editor**, set the **Retain** **null** **values from the source** option as **null** **values** in the Data Flow.
4. Go to the **Columns** tab; here, you can check as many columns as you want to be fetched from the source. By default, all columns will be fetched. Leave it as is and close the **Flat** **File** **Source** **Editor**.
5. Drag-and-drop **Derived** **Column** **Transform** after the flat file source is processed and then click on the flat file source; you will see a blue and red arrow going out from the component; connect the blue arrow to the **Derived** **Column** **Transform**. These arrows are called data paths because they are moving data rows.



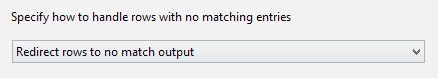
1. In the derived column transform, we can apply calculations with expressions. In this transform example, we want to calculate the age of the customer based on the current year and birth year.
2. Double-click on the **Derived** **Column** **Transformation** and write the following expression in the **Expression** box of the first row in the grid:

**YEAR(GETDATE()) - YEAR((DT\_DATE)BirthDate)**

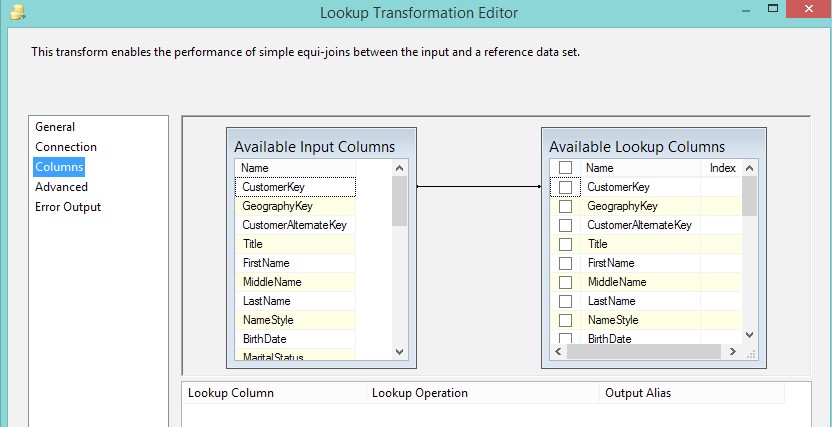
1. Set the **Derived Column Name** as **Age** and close the derived column transform. The following screenshot shows how a new derived column is added:



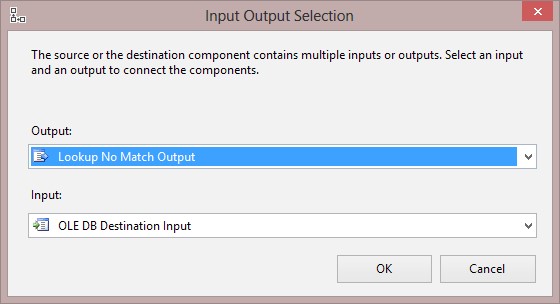
1. Now, we want to check the existence of the record with a **CustomerKey** in the destination table. So, we use a lookup transformation to do this.
2. Drag-and-drop a lookup transform after the **Derived** **Column**, and connect the data path (the blue arrow) from **Derived** **Column** to the lookup transform.
3. Double-click on the **Lookup** component. In the **Lookup** **Transformation** **Editor**, in the **General** tab, leave **Connection** **Type** and **Cache** **Mode** as is. In the **Specify how to handle rows with no matching entries** option, choose **Redirect rows to match output**:



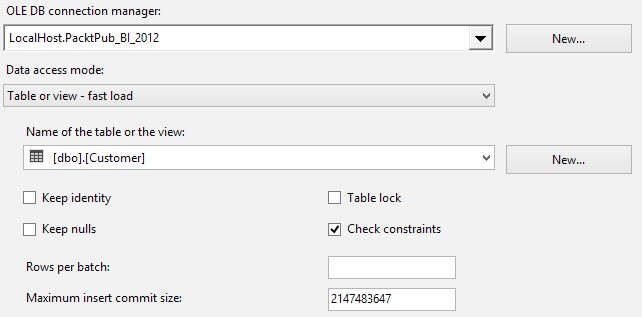
1. Go to the **Connections** tab, verify the **OLE** **DB** connection manager to be set in the **Packtpub\_BI\_2014** connection, and in the **Use a table or view** drop-down list, choose **[dbo].[Customer]**. You can also preview the existing data. Note that the table is currently empty, so there will be no data rows in the preview.
2. Go to the **Columns** tab; you will see two columns set: the first column is incoming columns and the second one is lookup table columns; connect the **CustomerKey** of both columns set by dragging-and-dropping **Available Input Columns** to **Available** **Lookup** **Columns**.



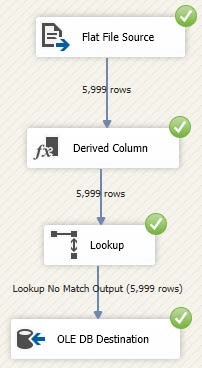
1. Click on **OK** and close the lookup transform.
2. Now, add an OLE DB destination after you close the lookup transform, and connect the data path from a lookup transform to an OLE DB destination. You will see an **Input** **Output** **Selection** dialog box that asks if it can select the input data path for the OLE DB destination. This data path appears because the lookup transform, by default, has two data path outputs: one for rows that matched and another for rows that don't match. In this example, we want to insert only new records that are not matched with the lookup table. Choose **Lookup No Match Output** and then click on **OK**.



1. Double-click on **OLE DB Destination**. In **OLE DB Destination**, verify the **OLE DB Connection** to be set with the **Packtpub\_BI\_2014** connection manager.
2. Leave **Data** **Access** **Mode** as **Table or view - fast load** and choose the **Customer** table from the drop-down list. Uncheck the **Table** **Lock** option:



1. Go to the **Columns** tab and verify all the columns to be mapped to each other, from **Input** **Columns** to **Destination** **Columns**. The column names in the dataset are similar to the destination table's columns as the mapping is applied automatically.
2. Close **OLE DB Destination**. Run the package. You will see that data rows fetched from the CSV file are transformed by the derived column, and after a lookup on the destination table, new rows are added to the destination table:



What just happened?

In this example, you learned how to work with **Data** **Flow** **Task**. The Data Flow tasks do the processing for the data stream on a row-by-row basis or as a batch of records (based on the configuration). Data Flow provides a more effective way of controlling the data stream between the source and the destination. You worked with the SSIS toolbox in Data Flow, which is different from the toolbox in **Control** **Flow**. In Data Flow's SSIS toolbox, you can see three types of components: **Sources**, **Transformations**, and **Destinations**. You've learned how to work with Data Flow components and you transferred data rows from a CSV file into a database table with a small transformation and a lookup for existing records.

In step 3, we created a Data Flow task. The Data Flow task, as you've seen in this example, utilizes a specific tab in the package designer with a specific SSIS toolbox that shows Data Flow components.

**The Data Flow components**

The Data Flow components are of three types: **Source**, **Transformation**, and **Destination**. Source components only have outputs. They might have more than one output based on the source type; for example, an XML Source may have more than one output. Transformations have at least one input and one output; they might have more than one input, such as Union All Transform, or more than one output, such as Multicast. Destination components accept input; they usually don't have a default output.

There is a data path used to connect the **Data** **Flow** components. As you've seen in step 11, a data path contains data rows. So, the difference between a data path in Data Flow and a precedence constraint in a control flow is that the data path contains data rows, but a precedence constraint defines the order of execution. A blue data path means correct data rows and a red data path means failed or erroneous data rows.

**The Source component**

In step 4, we created a flat file source to get data from a CSV file. There are different source components in a Data Flow that can fetch data from different data sources, such as databases, excel files, and flat files. The following table shows a list of Source components in an SSIS Data Flow:

|  |  |
| --- | --- |
| **Source component** | **Type of data source that can be used** |
| OLE DB Source | SQL Server, Oracle, and any data source that provides an OLE DB connection provider |
| ADO.NET Source | SQL Server, Oracle, and any data source that provides an ADO.NET connection provider |
| ODBC Source | MySQL, DB2, and any data source that provides an ODBC connection provider |
| CDC Source | Changed data from Change Data Capture in the SQL Server DB |
| Flat File Source | Text or CSV files that can be delimited or have a fixed width |
| Excel Source | Microsoft Excel files and spreadsheets |
| Raw File Source | An internal binary file structure that passes data between Data Flows |
| XML Source | XML content loaded from a variable or a file connection |

**Flat File Connection Manager**

In step 6, we created a flat file connection manager. The **Flat file Connection Manager** menu connects to text files and has the ability to fetch data rows by delimiters or by a fixed width. There are three different formats that are recognizable for the flat file connection manager: delimited, fixed width, and ragged right. The delimited type, used in this example, is for the text files of which the columns and rows are delimited by one or more characters. In this example, the CSV files that are delimited by a single comma for columns and rows are delimited by a carriage return. Fixed width is useful when columns have a fixed length. Ragged right determines the last column with a row delimiter, which is a little different than fixed length, because fixed length determines the last column with fixed length as well.

We also set the **Text Qualifier** field as **"**(double quotes) because all data values in the text file are surrounded by double quotes. Since the first row of the text file contains column headers, we checked the **Column** names in the first data row option. We can also choose the number of rows that we want to skip from the first row of the file. This option is useful when there are some header rows for the flat file that doesn't contain data rows.

In the **Columns** tab (step 7) of the **Flat** **File** **Connection** **Manager**, you can change the row delimiter and column delimiter and reset columns to see how columns will be picked with the new setting. The **Advanced** tab (step 8) provides detailed information about each column; you can change properties such as data type, length, and delimiter for each column in this tab.

By default, null values in the flat file will be picked as empty string values. To change this behavior, you can check the **Retain null values from the source** option as null values under the **Data** **Flow** option in the **Flat** **File** **Source** **Editor** as we did in step 9.

**Derived Column transformation**

We used a **Derived** **Column** transformation in this example to calculate a customer's age from the birth year. **Derived** **Column** is one of the most useful components in the **Data** **Flow** task because it uses expression language in SSIS, which contains some built-in functions and operations; it can use functions and operations on parameters, variables, and also input columns. As you've seen in steps 13 and 14, we used a simple expression to calculate the customer's age.

The **Derived** **Column** transform can generate more than one column using expressions, and the generated column can be used to replace the existing columns or acts as a new column in the data stream.

**Expression language**

The Expression language is a simple functional language for SSIS, and its structure looks similar to C# expressions; there are many built-in functions that can be used. Functions in the expression language are categorized into five main categories: string functions, mathematical functions, date functions, type cast, and null functions. There are operators for conditional

statements such as AND and OR as well. More description about SSIS expressions can be found in the MSDN website, [**http://technet.microsoft.com/en-us/library/ ms137547.aspx**.](http://technet.microsoft.com/en-us/library/ms137547.aspx)

**Lookup transformation**

We used a Lookup Transformation to look up through a reference table (in this example, the customer table) and find the matching or nonmatching records (step 16). Lookup transformation only works with two types of connection managers: the OLE DB connection manager and Cache connection manager. So, as a result, if you want to look up through a reference table that you cannot connect to, with an OLE DB connection, you can use another Data Flow task and load the content of that table into a Cache transform, and then use a Cache connection manager to the Cache transform object in lookup transform.

Lookup components can also use three options for **Cache Mode**: **Full Cache** (default), **Partial Cache**, and **No Cache**. **Full** **Cache** will load all data rows from the reference table into the memory, so if the reference table is huge or memory of the server is low, you may need to choose other cache modes.

In step 17, we redirected rows that do not match with the transform to find out new records that do not exist in the reference table. In step 18, we set the reference table, and in step 19, we mapped input columns to reference columns. For this example, we find out the existence of a customer by checking the **CustomerKey**. We can also pick any of the reference columns to be fetched as the output (in match output).

As you've learned, Lookup transform has two outputs; when we connect the data path out of this component, we should choose the one that we want to work with (step 21).

**Data Flow Transformations**

There are many useful transformations in SSIS **Data** **Flow**. You can perform many data transformations without the need to write scripts. The following table shows the most useful transformations with their usages (not all the transformations are listed in the table):

|  |  |
| --- | --- |
| **Transformation** | **Functionality** |
| Derived Column | Creates new columns with expressions |
| Data Conversion | Converts data types |
| Aggregate | Applies aggregation on one or more columns in the data stream |
| **Transformation** | **Functionality** |
| Conditional Split | Splits the data rows based on one or more expressions |
| Lookup | Looks up existing values in the reference table |
| Merge Join | Joins two data streams from different data sources (left or inner join) |
| Multicast | Creates copies of the existing data stream |
| OLE DB Command | Executes a SQL statement on an OLE DB connection (with or without parameters) |
| Row Count | Counts the number of rows and insert the result into a variable |
| Script Component | Executes a .NET written script on the data stream; this component can be used as a source, transform, or destination |
| Slowly Changing Dimension | To load data into a data warehouse dimension, you can choose the type of SCD to apply, such as 0 (fixed), 1 (changing), and 2 (historical) |
| Sort | Sorts a data stream based on one or more columns |
| Union All | Unites a data stream from all sources |
| Pivot | Changes values in rows to columns |
| Unpivot | To change columns to rows |
| Audit | Adds audit columns in the data stream |
| Fuzzy Lookup | Applies a fuzzy lookup with threshold on a reference table |
| Term Lookup | Applies text mining to find terms in an existing string in the data stream |

Transformations can be categorized in different aspects. One type of categorization is that it can be based on blocking and non-blocking. For example, Derived Column is a non-blocking transformation because each record will be processed and sent to the output, and then the next record will be processed. Sort transformation is a blocking transformation; all data rows will be loaded first into the memory and then sorted out and sent to the output. It is always the best practice to avoid blocking transformations as they reduce the performance of a package and ETL.

**OLE DB Destination**

We used the OLE DB destination in step 21 to load data into a SQL Server database table. This kind of destination can be used for all databases in which we can create an OLE DB connection. Usually, you can find the OLE DB connection provider of a database vendor in its website. The OLE DB destination can load data into a database table with the fast load or regular option. The fast load option inserts multiple rows in one batch, which has a good performance. You can also set the batch size and other options (as you saw in step 23).

Input data stream columns were mapped to the destination columns in step 24, and finally in step 25, you saw an execution sample of this Data Flow.

**Destination Component**

There are different destinations to be used when loading data for example, a flat file, Excel, database table, and others. The following table shows a list of destinations and their usage (all the destinations are not listed):

|  |  |
| --- | --- |
| **Destination component** | **Type of destination that can be used** |
| OLE DB destination | SQL Server, Oracle, and any databases that allow an OLE DB connection |
| ADO.NET destination | SQL Server, Oracle, and any databases that allow an ADO.NET connection |
| ODBC destination | MySQL and any databases that allow an ODBC connection |
| Flat File destination | A flat text file (having a delimited or fixed width) |
| Excel destination | A Microsoft Excel spreadsheet file |
| Raw File destination | Internal binary file for moving data between Data Flows |
| Recordset destination | Object type variable |
| SQL Server destination | Only works with a local instance of SQL Server |

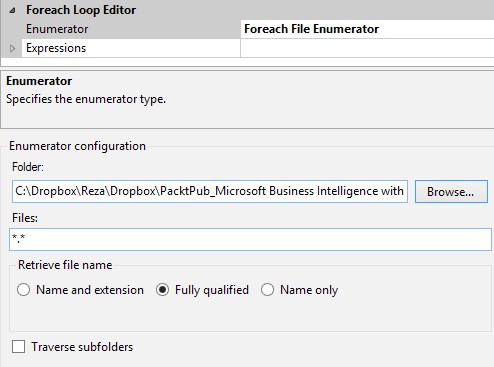
Containers and dynamic packages

You've learned about connection managers and control flow tasks in SSIS, but so far, all you've seen are static connections. There is a vital requirement for tasks and connections to be dynamically set at runtime, for example, when you get the name of the source database servers from an Excel spreadsheet. As another example, server names would be changed for development, testing, and production purposes. In this section, we will learn more about containers to learn how to loop through items of a collection, and then how to set the connection managers dynamically.

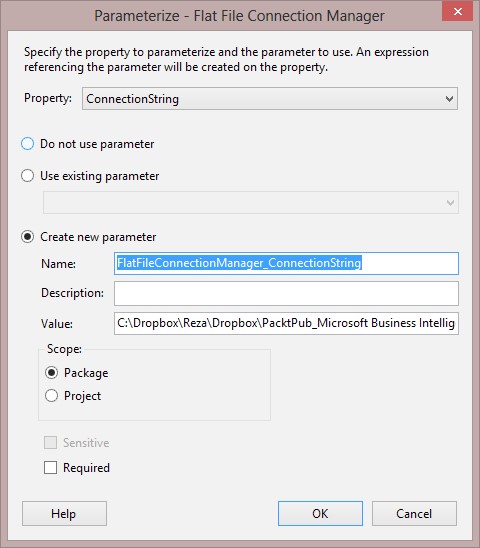
|  |
| --- |
| Time for action – looping through CSV files in a directory and |
|  |
| loading them into a database table |

In this example, we will use the **Foreach Loop** container to loop through files in a directory, and then we will set the connection manager of the source file dynamically in the flat file source, and load the content of the source file into the destination database table. Then, we will write a log entry into the database with the name of the file and load date time. For this example, we use packages and tasks that we developed in the last two examples:

1. Create a new SSIS package and name it **Master** **Package**. Drag-and-drop a Foreach Loop container in **Control** **Flow**.
2. Create a variable with a string type and name it **FilePath**.
3. Double-click on the **Foreach Loop Container**. In the **Foreach Loop Container Editor**, set **Enumerator** as **Foreach** **File** **Enumerator**.
4. In the **Enumerator** configuration, set the source directory to the path of the directory that contains Customers CSV files. Leave other settings as they are:



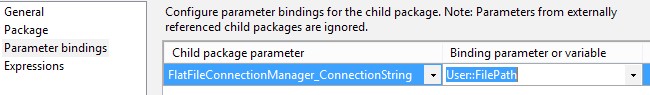
1. Go to the **Variable** **mappings** tab and choose the **User::FilePath** variable in the **Variable** column; the index automatically will be set to **0**.
2. Close the **Foreach** **Loop** **Container** **Editor**.
3. Go to the **Transfer Customer Data** package (from the previous example), right-click on the **Flat File Connection Manager** and choose **Parameterize**. In the **Parameterize** window, set **Property** as **ConnectionString**. Choose **Create a new parameter** and name it **FlatFileConnectionManager\_ConnectionString**. Leave other configurations as they are. Click on **OK**:



1. You will see that a small fx icon will appear beside the **Flat** **File** **Connection** **Manager** icon. This fx icon shows that one of the properties of this object will be resolved by an expression at run time, in this case, the **ConnectionString** will be set by the parameter. Save this package and close it.
2. Go to the **Archive Customer File** package, and then go to the **Parameters** tab.
3. Create a new parameter and name it **SourceFilePath**. Set the data type of this parameter as **string**.
4. Drag-and-drop an **Expression** task into the package designer and rename it to assign the parameter value to **Variable**. Write the following expression in **Expression Task Editor**:

**@[User::FilePath]= @[$Package::SourceFilePath]**

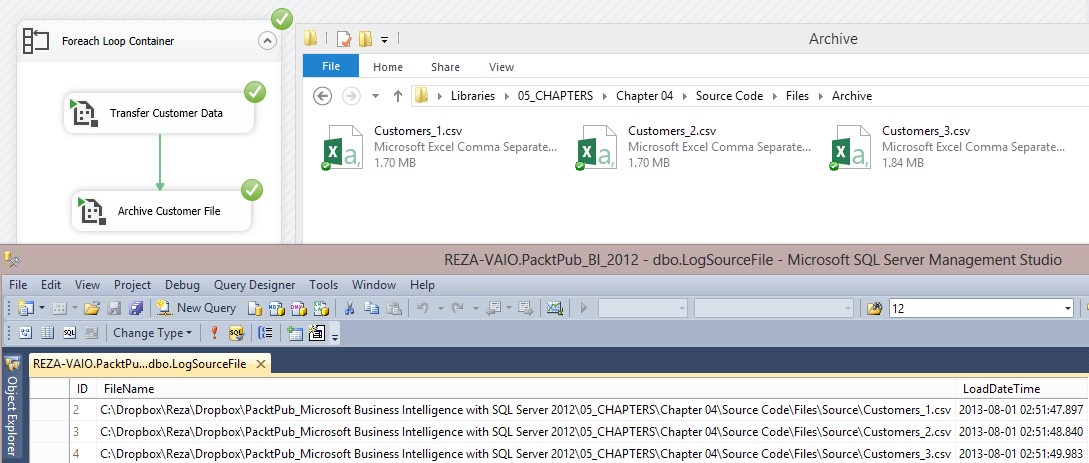
1. Connect the green precedence constraint from the **Expression** task to the **Archive** **File** task and save this package and close it.
2. Go back to **Master Package**. Add an **Execute Package Task** into the **Foreach Loop Container** and name it **Transfer Customer Data**.
3. In the **Execute Package Task Editor**, under the **Package** tab, verify whether **ReferenceType** is **Project** **Reference** and select the package's name as **Transfer Customer Data.dtsx**.
4. In the **Parameter** **Binding** tab, add a binding and bind the package parameter with the **User::FilePath** variable, as the following screenshot shows:



1. Close **Execute Package Task**. Drag-and-drop another **Execute Package Task**, and rename this one to **Archive Customer File**.
2. In the **Execute Package Task Editor** of this task, set **Package** as **Archive Customer**

**File**, and in the **Parameter** **Binding** tab, create a new binding to bind the **SourceFilePath** parameter of the child package with the **User::FilePath** variable.

1. Close the second the **Execute Package Task** editor. Run the **Master Package**.
2. You will see that the **Master Package** runs and executes both packages within each iteration of the loop. You can check the **Archive** folder after executing the package to see whether all the source files have been moved. Also, the **LogSourceFile** table will contain entries for all files that are processed. The **Customer** table also contains data rows of all imported customer files. The following screenshot shows the result of execution:



What just happened?

In this example, you learned how to work with a Foreach Loop container and create a dynamic connection using parameters and expressions. You also used **Execute** **Package** **Task** to execute other packages. Working with connections dynamically within a loop structure is one of the most common scenarios in data transfer.

**The Foreach Loop container**

SSIS utilizes three containers: Sequence, For Loop, and Foreach Loop. For Loop is a simple loop structure that uses initialization, conditions for evaluation, and assignment. Foreach Loop is used for looping through items of an object collection, array, files, XML content's node and attributes, SQL Server objects, and so on. There are seven types of enumerators in the Foreach Loop container. Each enumerator type works with a specific kind of object collections.

In this example, we used the Foreach Loop container in step 3, and we used **File Enumerator** in step 4 to loop through files in the source directory. An enumerator configuration for the **File** enumerator allows you to traverse subfolders. You can also use file masks to filter only those files that follow a template; for example, **Customers\*.csv** will only pick those files with **Customers** at the beginning of the filename and **.csv** as the extension. You can specify whether you want to pick the full path of the file or just the filename (with or without the extension).

In step 5, we specified a **FilePath Variable** to be loaded with the fully qualified path fetched in each iteration of the loop. The index **0** indicates the first column in the looping dataset. As the file enumerator only has one column, the index **0** will only pick the output column, which is a fully qualified name in this case.

**Parameterize**

In step 7, we used parameters to set the flat file connection string dynamically. As you've seen, the **Parameterize** GUI helps to create the connection string dynamically. It assigns the value of the specified property with the selected (or created) parameter. The **Parameterize** GUI utilizes the **expression** property in the code behind; in other words, if you check the **Expression** property of the connection string, you will see that an expression sets the connection string property's value with the parameter's value.

In step 9, we created a **package** parameter. This parameter is used as an incoming argument to the package and is good for package-level abstraction. We created the **SourceFilePath** parameter. Then, with an **Expression** **Task**, we write the parameter's value into the package variable (step 11). The best practice is to always use parameters as an input gate and variables for interaction between tasks inside the package. This will end up with the package having a good level of abstraction.

In the expression language, variables will be identified with the **@[<user or system>::<variable name>]** signature. However, parameters have a slightly different signature as **@[$<package or project level>::<parameter name>]**. You've seen an example of this expression signature in step 11.

**Execute Package Task**

Calling other packages is one of the vital requirements in an ETL scenario. Using **Execute Package Task**, you can execute other packages, and as a result, you can fortify the design of data transfer in such a way that each package performs a main task. As you can see, in the previous example, we used one package to transfer data between CSV files and database tables, and another package to archive a file and write a log entry. Then, we used two execute package tasks (steps 13 to 17) to run each package.

**Execute Package Task** is a simple task that asks for the child package to be run. If the child package is protected with a password, you can enter the password in the **General** tab of **Execute** **Package** **Task**. The **Parameter** **Bindings** tab (step 15) will be used for passing the package variable's value to the child package's parameter.

Using **Execute Package Task**, you can utilize a master package and run other packages in the master package. So, logging and troubleshooting of packages will be much easier, and the maintenance cost will be much lower because each package will be responsible only for a specific task.

Deploying and executing

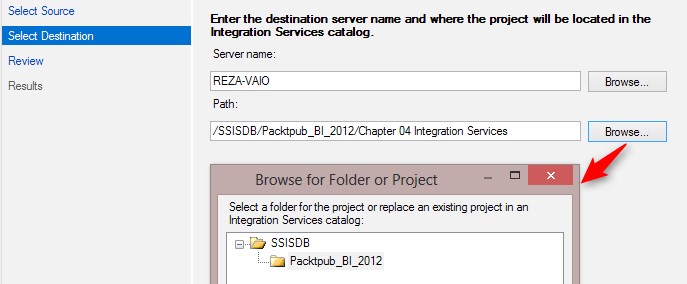
SSIS packages should be deployed in the production environment. SSIS 2012 introduced a new database as a repository for deployed SSIS projects and packages; this database is called SSIS catalog. SSIS catalog can be accessed through SSMS, which is handy for database administrators. SSIS catalog stores projects, packages, relationships of variables, execution log, and many other types of deployment and execution-related information that is useful in production environments.

Usually, there is no SSDT or development tool in the production environment. So, SSIS packages should be executed using the DTEXEC utility, either through SSIS catalog or from a SQL Server Agent Job. Calling SSIS projects from SQL Server Agent Job is one of the most popular methods in real-world scenarios because an ETL process usually needs to run on a scheduled basis.

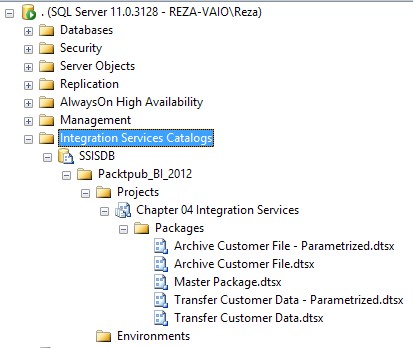
Time for action – deploying an SSIS project

In this example, we will first create the SSIS Catalog. Then, we will deploy the SSIS project that we created in the previous examples. For this example, you need to have SSIS 2012 or higher versions installed, because SSIS Catalog is not available for earlier versions of SSIS. Perform the following steps to deploy an SSIS project:

1. Open **SQL Server Management Studio** (**SSMS**) and connect to the database engine.
2. In the object explorer, expand the server name and then right-click on the **Integration** **Services** **Catalogs** and then click on **Create** **Catalog**. In the **Create Catalog** dialog box, just set a password and click on **OK**.
3. SSISDB catalog will be added under the **Integration** **Services** **Catalogs** node in SSMS. The SSISDB name cannot be changed for this version as well as the earlier versions of SSIS.
4. Go to SSDT and right-click on the **Project** file in **Solution** **Explorer** and select **Deploy**.
5. Now, the **Integration** **Services** **Deployment** **Wizard** will appear. In the **Select** **Source** step, leave the configurations as is (in which the project deployment file is present) and then go to the next step.
6. In the **Select** **Destination** step, browse the database engine instance under which you've created the SSISDB catalog. Then, select **Path** and create a new folder for **Packtpub\_BI\_2014** as the following screenshot shows:



1. Continue the wizard and deploy the project. After successful deployment, go back to SSMS and refresh the **Integration** **Services** **Catalogs** node. Now, you will see the new project deployed under the **Packtpub\_BI\_2014** folder as the following screenshot shows:



What just happened?

In this example, we first created the SSIS Catalog (steps 1 to 3). SSIS Catalog is an internal database repository for SQL Server Integration Services projects. You won't be able to deploy an SSIS project in the project deployment model without having an SSIS catalog installed.

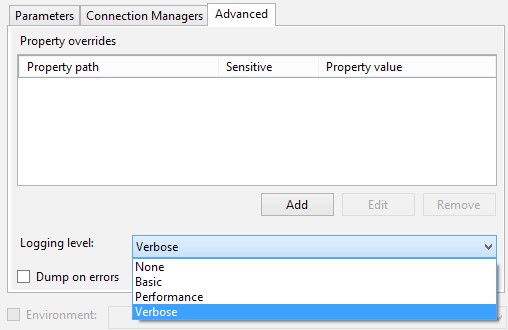
In steps 4 to 7, we deployed the SSIS project into the SSIS Catalog. As you've seen in the example, we can create folders in the catalog and then deploy one or more projects under each folder. After deployment, the catalog shows folders under SSISDB, and then under these folders, the projects will be listed, and each project may contain one or more packages. Environments that are listed under each project are for variable configurations.

Time for action – executing an SSIS package from a catalog

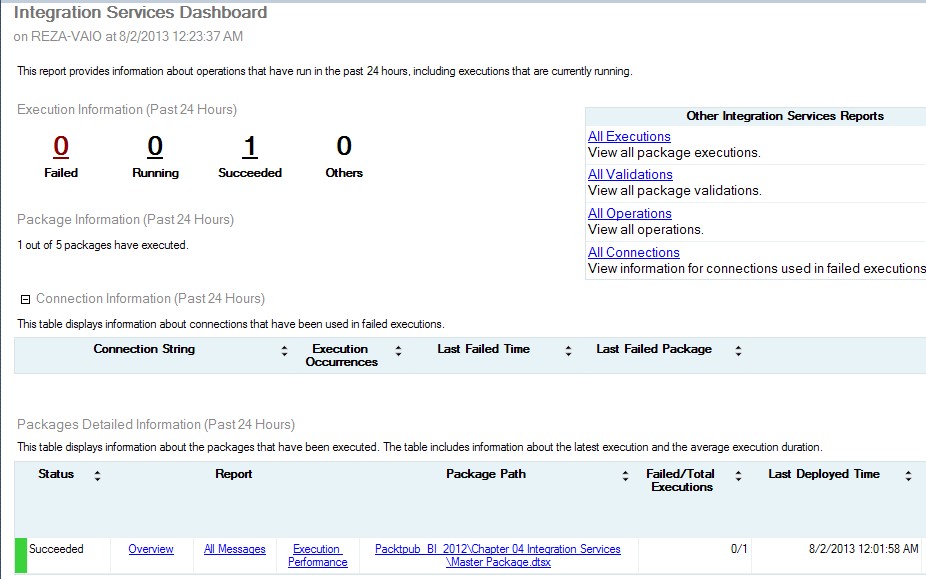
After deploying the project, we can execute the **Master** **Package**. Before executing the

**Master** **Package**, revert all the changes, which means move customer CSV files from the **Archive** folder back to the source folder, and truncate the customer and log database tables. To get this sample working, you will need to have the project deployed to SSIS Catalog, as illustrated in the previous sample, and perform the following steps:

1. In the SSMS, under **Integration Services Catalogs**, under **SSISDB**, expand the **Packtpub\_BI\_2014** folder and then expand the packages under the **Chapter 04 Integration Services** project. Then, right-click on the **Master** **Package** and click on **Execute**.
2. In the **Execute** **Package** window, go to the **Connection** **Managers** tab. You will see all the connection managers here and you will be able to change the properties of the connection managers from this GUI tool. Just leave the configuration as is.
3. Go to the **Advanced** tab and change the **Logging** **level** to **Verbose** (each logging level provides logging facilities different events. Verbose is the most detailed logging level):



1. Now, click on **OK** to execute the package. You will see that the execution starts. A dialog box appears that asks you to open **Overview Report**. Click on **No**.
2. Right-click on the **SSISDB** node, and from the pop-up menu, choose **Reports** and then **Standard** **Reports**, and then click on **Integration Services Dashboards**.
3. You will see that the SSRS dashboard shows the execution summary of packages. You can also drill down to the execution details of each package and check the information and error messages for each package at the task level. The following screenshot shows the **Integrated Services Dashboard** window:



What just happened?

You've seen a very simple example of executing packages from SSIS catalog. As you've seen in this example, the **Execute Package** window provides a useful GUI to assign connection managers and configure log settings before executing the package.

You've also seen that SSIS catalog provides extensive logging as a part of the execution. There are also some SSRS reports that show a summary of high-level information about packages and detailed information about each task and error messages (if there is any).