3

Overview: Understanding the Toolbox

In this chapter, we will review the different components of SQL Server Integration Services (SSIS). Our intent here is to go over the main components of SSIS, to give you a basis for understanding and working with the SSIS database-access solution. We won't explore the SSIS components in depth, as there is a lot to learn and understand about some of them. Once you get an overall idea of what these components are, it will be easier for you to pick up the deeper concepts later.

First, we will cover the most important components that you should understand to use SSIS and build your own SSIS-based solution. These are the Solution Explorer, the different views of a package, the SSIS Toolbox, properties, and variables. Next, we will step you through some of the most commonly used elements or tasks from the SSIS Toolbox component, which are used to create an SSIS workflow. Additionally, we will cover the basic uses of parameters and configuration files and how to create them. We will go into further details about configurations in chapter 13. Being familiar with the core components will help you understand the later chapters as we step through some code.

Note

If you are having a difficulty finding the Business Intelligence template, make sure that you have installed the SQL Server Data Tools for Visual Studio (SSDT), as mentioned in chapter 2.

Getting Started

Before we get started with SSIS, a quick explanation of the components is in order. A solution is like your main directory, which contains multiple sub-directories. A solution is designated with an .sln extension filename. Within a solution, the project is your main sub-directory and is designated with a .dtproj extension filename. A project is further split into multiple components: packages, connection managers, and so on. You can create multiple projects in a solution, and when you first create your new SSIS project, a new package will be created and presented to you by default. Packages are designated with a .dtsx extension filename. You can add as many packages as you'd like to your project.

Fire up Visual Studios if you haven't already done so, and let's create our first SSIS project! Let's start by creating a new project by selecting **File** > **New** > **Project...**, as shown in Figure 3.1.

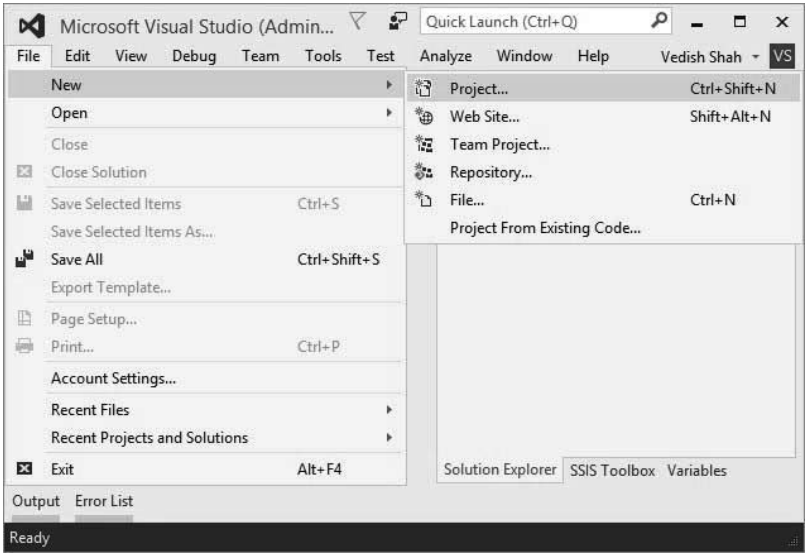

Figure 3.1: Creating a new project

This will display the wizard shown in Figure 3.2.

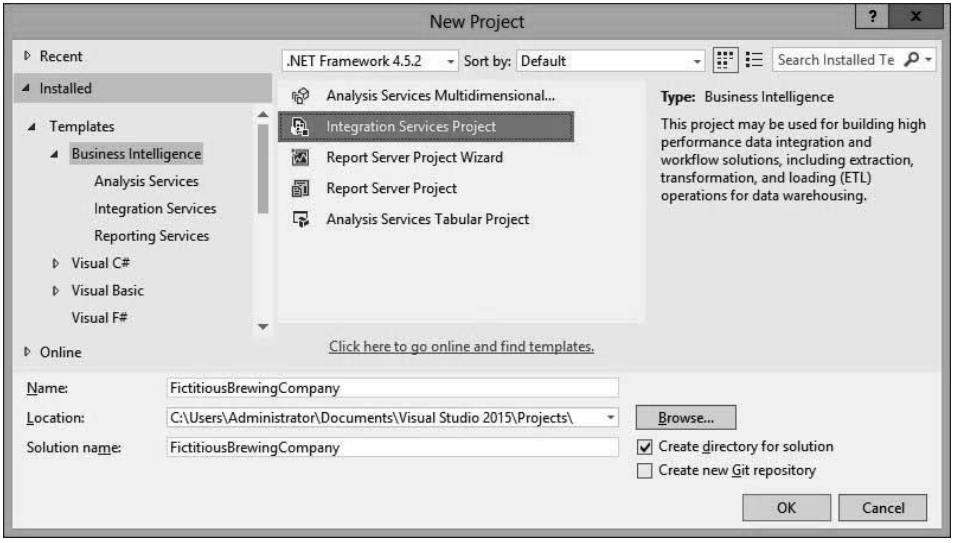

Figure 3.2: Naming the project and its save location

We will call our new Integration Services project *FictitiousBrewingCompany*.

Once you have successfully created your new project, you will be at the landing page (Figure 3.3), where you can see most of the components. This is your design window and is where you will create your SSIS package workflow.

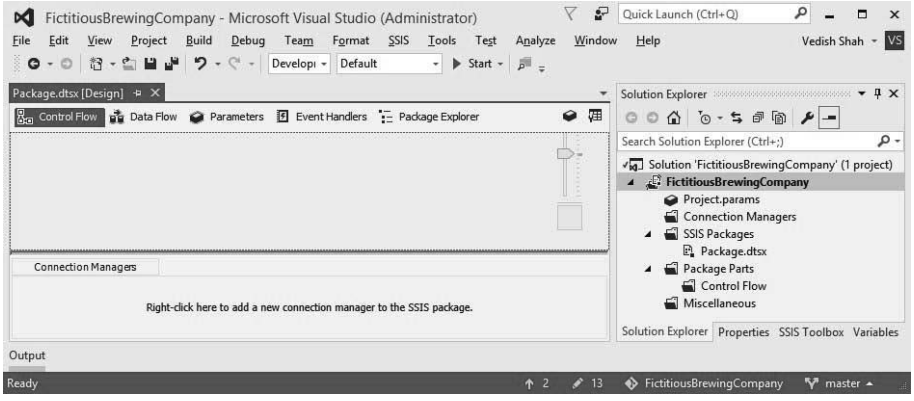

Figure 3.3: Overview of a Project's landing page after its initial creation

Before we delve into the different components and tasks, let's take a quick look at the importance of how you build your SSIS solution. How you build the solution depends on the type of deployment model you use for your project. These models also dictate the type of deployment utility that will be generated by the solution when it is built.

Deployment Models and Utilities

Every SSIS solution can be built on two types of deployment models: project deployment, which generates an .ispac file, and package deployment, which generates a manifest.

Project Deployment

This is a newer deployment model that was introduced with the version of SQL Server Integration Services in SQL Server 2012. By default, new projects are automatically generated using the project deployment model. With this deployment type, a user can deploy the whole project as a single entity via an .ispac file. You treat an .ispac file as a zipped folder. You can use any folder unzip program to extract files from the .ispac file and work on the extracted files using Visual Studio's SSDT.

Package Deployment

This is the legacy deployment model of SSIS that allows a user to deploy packages individually on file systems and integration servers. In this model, a package is the entity to be deployed with its dependent files. A deployment utility called a *manifest* is created with this model. This model, in our view, is the best deployment model for SSIS packages as it gives a user more freedom when deploying such packages. We will talk about deployments in more detail in chapter 13.

Why bring up deployment models this early? SSIS packages are built to be deployed right off the bat. Each deployment model has a subtly but noticeably different look and folder structure. So, it is easier to explain the components based on the deployment models. Throughout this chapter, we will be using the default model, project deployment, to explain the different components of an SSIS package.

Then what about the package deployment model? Microsoft has done a great job of keeping the functionalities of both deployment models very similar, to avoid confusing their users. When we explain one component of the project deployment model, we will also mention its counterpart from the package deployment model, if there is one.

At the end of this chapter, we will provide the steps to convert your project from the project deployment model to the package deployment model and vice versa. It is up to you to decide which model you prefer to use as the deployment model for your future builds. Now, let's get started!

Solution Explorer

Solution Explorer allows a user to view and control the different parts of a solution. Depending on the type of project model you are working with, the Solution Explorer will be presented differently. For a project deployment model, it is split into four subfolders: Connection Managers, SSIS Packages, Package Parts, and Miscellaneous, as shown in Figure 3.4.

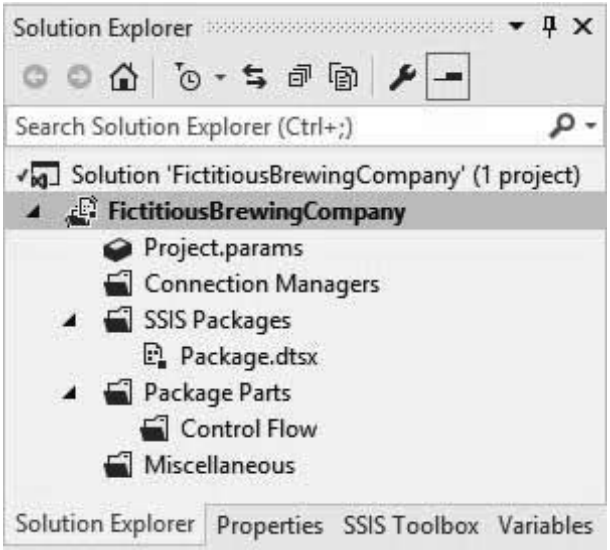

Figure 3.4: Solution Explorer

The **Connection Managers** folder is used to hold connections for your project data. It can be shared across your project packages so that you don't have to create multiple instances of the same connection. To add a new connection manager, right-click the folder and select the **New Connection Manager…** option. This will open a wizard that will step you through your connection creation. There are several types of connections that can be used for your project—for example, Flat File, Database, and Excel. A connection manager is designated by the .conmgr extension. The package deployment model, on the other hand, has a **Data Sources** folder. This folder strictly holds connections to your data and no other connection type.

The SSIS Packages folder is used to hold all the packages of your project. To add a new package, right-click the folder and select the **New SSIS Package** option. This will create and display the new package for you with the default name (`package.dtsx`). A number will be appended to the package name if you already have another package in your project with its default name. To add an existing package, right-click the folder and select the **Add Existing Package** option, then follow the steps provided by the wizard. A package is designated by the .dtsx extension. This folder is also called **SSIS Packages** in the package deployment model.

The **Package Parts** folder is used to hold prebuilt tasks that can be referenced by any packages. A *package part* is reusable code, which is a set of controls that can contain multiple tasks and/or containers. Package parts can have their own connection managers, variables, properties, and logging but cannot have event handlers, parameters, or nested packages. A package part is designated with a .dtsxp extension. One important thing to note about package parts is that you can only have one container or task per part. This folder is also called **Package Parts** in the package deployment model.

The **Miscellaneous** folder holds a file that is neither a package nor a data source. This could include your configuration file(s) of your project package(s). This folder is also called **Miscellaneous** in the package deployment model.

Package Views

An SSIS package is divided into multiple views, which include the **Control Flow**, **Data Flow**, **Parameters**, **Event Handlers**, and **Package Explorer** views (Figure 3.5). These views let you work with the different functionalities of a package. The views are fixed, tabbed windows that cannot be removed from the design window. Additionally, you cannot add any other tabbed windows as a new view. There are two additional views that are only visible either during execution or once a package has completed execution: Progress view and Execution Results view.

Figure 3.5: Package views

Control Flow is the main view of a package where much of work is implemented. It contains different elements that you can use to build your package workflow. These elements consist of:

- *Containers* that provide a structural support for other elements
- *Tasks* that provide a functional support
- *Constraints* that act as connectors/constraints between elements in the flow controlling their execution during runtime

The **Data Flow** view is used to control the data transformation elements of the package. Components of this view contain:

- *Sources*, which are used to pull data for manipulation
- *Destinations*, which are used to store data after manipulations or before further processing
- *Transformers*, which can be used individually and do not require an actual data flow
- Other common functions that are used for conversions and other tasks

The **Parameters** view is used to add, update, or delete parameters related to the project. This view is visible for both types of deployment models but is disabled for the package deployment model.

The **Event Handlers** view is used to capture events raised during the package's runtime. They can use the same elements that are available for the **Control Flow** view. Twelve different events can be captured and processed during package runtime.

The **Package Explorer** view lets you indirectly access a breakdown of the different views of your SSIS package.

The **Progress** view is a hidden tab and is visible only when a user enters the Debug mode. You can enter Debug mode by pressing **F5**. This will start to execute your package, and you can watch the output in this view.

The **Execution Results** view is only visible once a package has completed execution. This view will show the overall steps of the package execution.

SSIS Toolbox

The SSIS Toolbox (Figure 3.6) holds the different elements used to construct an SSIS package workflow. The Toolbox displays components in a categorized form and will display only those tools available for the view you are in. You cannot create your own categories, but you can move tools around and maximize and minimize categories for ease of use. If you choose, you can install additional third-party tools in the Toolbox. Once they are installed, you can right-click inside the Toolbox and click the **Refresh Toolbox** option to see your newly installed tools.

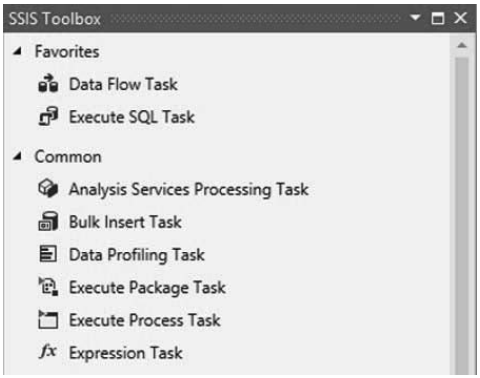

Figure 3.6: SSIS Toolbox

By default, the Toolbox is displayed automatically either when a new package is created or when you open an existing package. If you have a difficulty finding the toolbox, you can simply click the **SSIS Toolbox** button located at the top right of the design window, as shown in Figure 3.7.

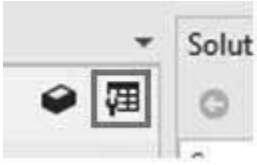

Figure 3.7: SSIS Toolbox access button

Properties

When you first create your SSIS package, you can set up the different properties of the package and its components using the Properties window (Figure 3.8, page 42). In the default form, the Properties window is in a categorized and alphabetic state. You also can view the properties fully alphabetically without any categorization. In a categorized form, the Properties window is split into following categories: **Checkpoints**, **Execution**, **Forced Execution Value**, **Identification**, **Misc**, **Security**, **Transactions**, and **Version**.

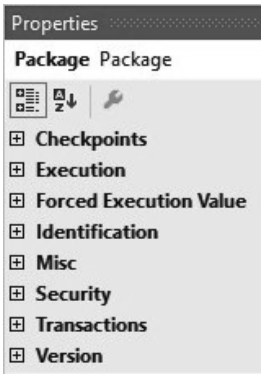

Figure 3.8: Properties window in a categorized form

- **Checkpoints** allow you to set up package-restart options. This property will restart the process from a specified point in the package instead of rerunning the whole package from the beginning.
- **Execution** lets you set the runtime behavior of an SSIS package.
- **Forced Execution Value** lets you force a value to be returned by the package.
- **Identification** lets you define an identity of the package (e.g., creation date, name, ID).
- **Misc** holds the properties of package configurations, element expressions, and package logging.
- **Security** lets you password-protect your package.
- **Transactions** lets you set the isolation level and transaction options for your package.
- **Version** lets you set your package's versioning.

Variables

As in any programming language, a *variable* is an object that holds a value provided to it, which can change at runtime. Within SSIS, there are two types of variables: system variables and user-defined variables.

System variables are variables predefined by SSIS that represent certain properties of a solution—for example, `PackageName`, `ErrorCode`, or `CreationTime`. As a user, you cannot create system variables, but you can best use them for error handling, logging, or just informational purposes. The best use of system variables we have found is to use them for error handling. Chapter 11 explains more about the use of this variable type.

User-defined variables are the types of variables that you define when you create your package's workflow. You can create as many user-defined variables as you'd like for a package. User-defined variables are distinguished with a simple box-like icon, as shown in Figure 3.9.

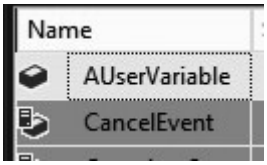

Figure 3.9: A user-defined variable (AUserVariable) and a system variable (CancelEvent)

Each variable type can be accessed by every element of a package workflow with one exception: *variable scope*. The scope is the accessibility of a variable within a package. Variables that are created with a package-level scope act as global variables that can be accessed by every element and in every view of a package. Variables created with specific scopes—container level, element level, or event level—can only be accessed by elements in that scope type. In SSIS, variables can play an integral role in the execution of a package. They can be used in many ways, as described in the following sections.

Expression Builder

At runtime, the Expression Builder evaluates and calculates the value of an expression to be used by other elements of a package. Expression Builder expressions are enclosed in double quotes (" "), and variables passed into the expression are required to be character strings. To use expressions in any of your tasks, you can use the task's properties and create expressions there. Alternatively, you can open the task, click **Expressions**, then click the Expressions drop-down menu and click **Ellipsis**. Then select **property** and click its ellipsis. You will see a screen like that in Figure 3.10 (page 44).

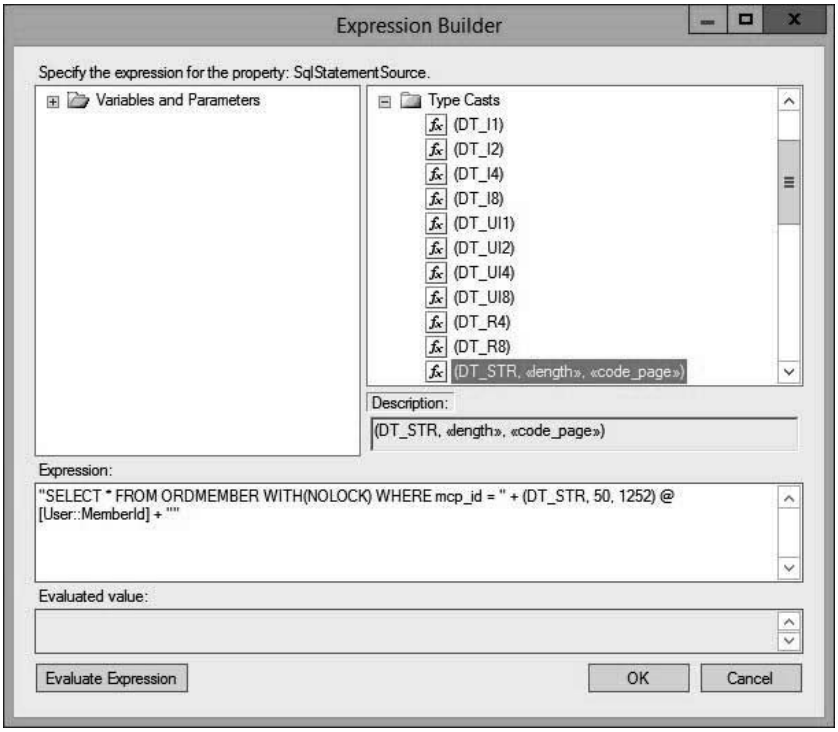

Figure 3.10: Expression Builder for an execute SQL task

If a variable isn't a string type variable, the Expression Builder will throw an "incompatible data type" exception when you evaluate it (see Figure 3.11). To avoid such exceptions, just cast your variable as a character string (`DT_WSTR` or `DT_STR`) within your expression, as shown in Figure 3.10. `DT_WSTR` is a Unicode character string, and `DT_STR` is a non-Unicode character string.

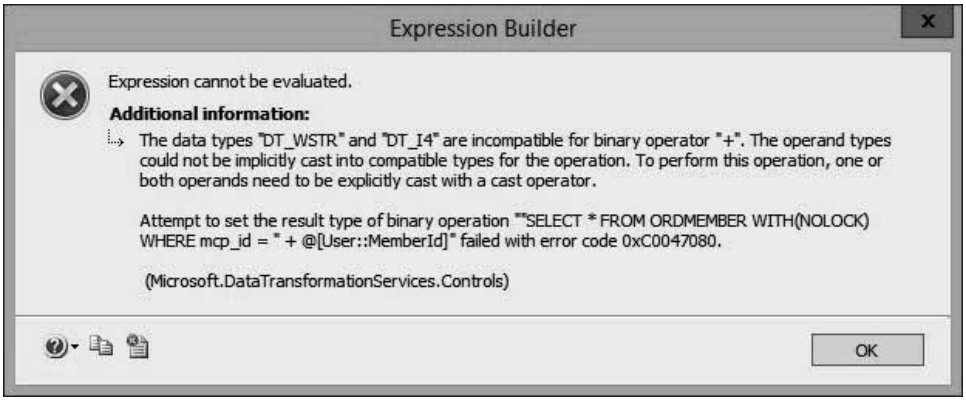

Figure 3.11: Incompatible data type error

Package Properties

Specific properties can be set for the elements that can allow changing their behaviors during runtime. For example, you can set the **Retain Same Connection** property of a Connection Manager to True/False to allow the usage of runtime-created temp tables in a loop.

Record Sets

Data can be pulled into a variable of the Object data type that can be used by a for each loop to enumerate through a data set, as shown in Figure 3.12 (page 46).

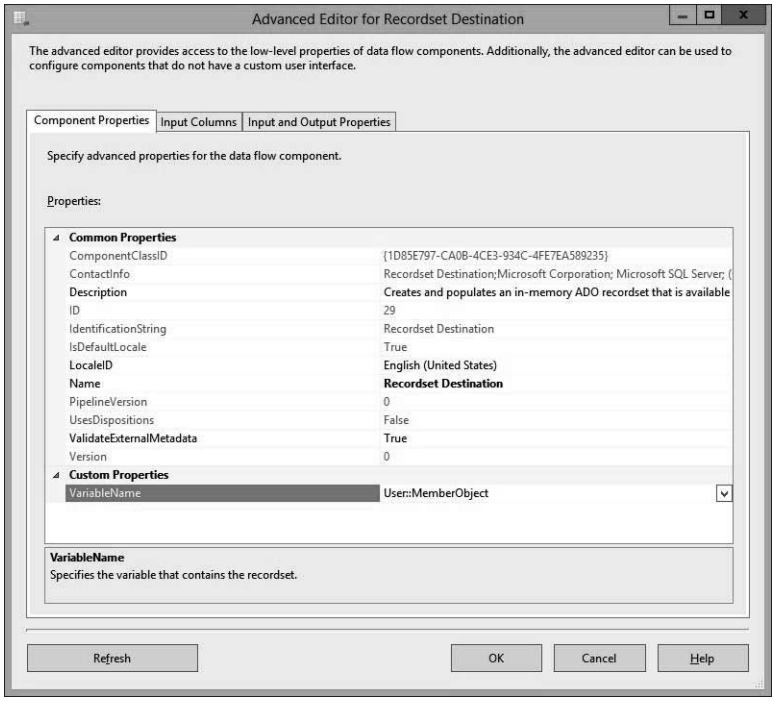

Figure 3.12: Record set destination: MemberObject variable of Object data type

Parameters and Return Values

SQL statements can be executed with input parameters and return values that can be stored during runtime for further use, as shown in Figure 3.13.

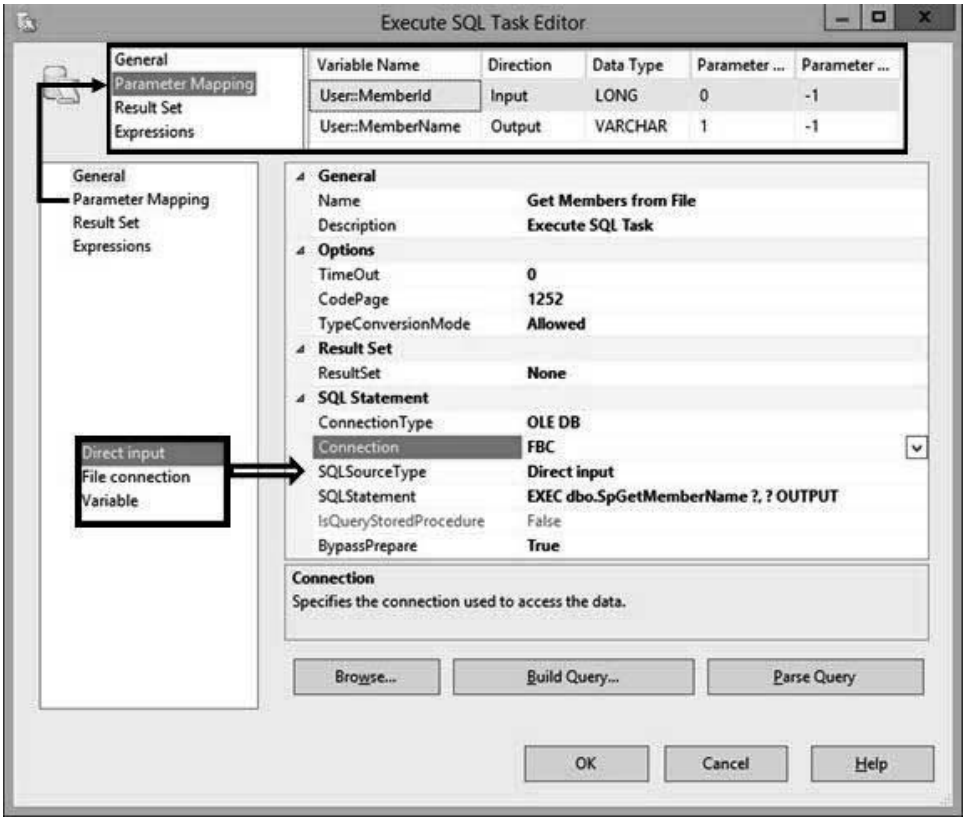

Figure 3.13: Execute SQL task: calling a stored procedure with variables mapped

Variables can also be used as expressions that can be evaluated during runtime or prior to runtime. This allows a user to, for example, store SQL statements that can be dynamically created during package runtime. To achieve this, you can change a variable's `EvaluateAsExpression` property value to True. If variables are to be evaluated during runtime, you can change a variable's `DelayValidation` property to True, as well. By default, every element's `DelayValidation` property is set to False, and these elements will be evaluated prior to the package execution. This allows a package to catch validation errors before reaching those specific execution points. Setting the `DelayValidation` property to True will allow the package to skip initial validation and validate the element once the specific point in the package is reached where such element is used.