SQL Server 2005 - SQL Server Integration Services - Part 4 - Variables

In the previous article of this series dedicated to the SQL Server 2005 Integration Services (SSIS), we started exploring features, which provide the ability to reuse packages (without modifying their design) in situations where some of their parameters need to be changed (frequently, this is required as the result of switching from development or testing environments to production). The first method we introduced involved the use of configurations, implemented through Package Configuration Organizer, included in Business Intelligence Development Studio. With its intuitive Wizard-driven graphical interface, it is straightforward to specify variables or properties of SSIS components, which values can then be assigned using external sources of data (such as XML configuration files, registry entries, environment variables, or variables defined in parent SSIS packages). We also described similar functionality available when executing packages via the DTExec command line utility. In this article, we will focus on variables, which capabilities have been considerably enhanced compared with their implementation in SQL Server 2000 Data Transformation Services. We will also look at using expressions - both in the context of variables and component properties - which further expand the dynamic nature of SQL Server Integration Services.

We already have provided an overview of the concept and basic characteristics of SSIS variables. The most significant ones we mentioned were the existence of two main categories (system and user) as well as scope, namespace membership, and modifiable properties associated with each. The only modifiable option of system variables is their ability to raise an event when their value changes (which can be either enabled or disabled). This is one of the primary distinctions that separates them from user variables, for which majority of properties can be arbitrarily changed, (the other is the fact that all system variables are predefined, while all user variables are custom created). Variables can be assigned either a literal value (as in SQL Server 2000 DTS) or an expression (new feature in SQL Server 2005), but in either case, they must posses a specific data type. As with package or component properties, values of variables are set directly within a package, but they might also be assigned through configurations or command line switches of the DTExec utility, at the time of package invocation (allowing you to alter package parameters and the outcome of its execution, without direct modifications of its content). This, in turn, significantly reduces potential maintenance and version control issues, simplifying package development, deployment, and management.

When executing packages from the command line with the DTExec utility, you need to reference variables using the syntax we described briefly in our previous article. Always starting with the "\Package" string, the notation consists of one or more additional elements preceded by the backslash (indicating that the next element is a container or a task name), a period (specifying that what follows is a component characteristic - in our case this is Variables collection), or square brackets (enclosing the index within the Variables collection - this role is handled by a value of variable Name property). In order to assign value to a variable, such string would be followed by the ".Value" suffix, separated by the semicolon from the last part, containing the actual value. For example, in order to set the value of the CampaignDonations variable (for the sake of simplicity, let's assume in this case that the variable has been defined on the package level) to the value "Kenny Lay", when executing the package called ToShredder.dtsx, you would type at the Command Prompt:

```
DTExec /FILE ToShredder.dtsx /SET
\Package.Variables[CampaignDonations].Value;"Kenny Lay"
```

When referencing a variable on the level of a package component, you would specify its path after the \Package entry, following the rules outlined previously (e.g. `Package\RunMe\Variables[CampaignDonations].Value;"Kenny Lay"` would allow you to accomplish the same goal but for a RunMe task level variable). Keep in mind, though, that with both methods (configurations and command line switches), values are assigned prior to package execution (rather than when its tasks are running). To overcome this limitation, you can take advantage of expressions, by assigning them to variables or component properties (more about this shortly).

There are various ways to create user variables, which depend on their scope and purpose, as well as the current context within the Business Intelligence Development Studio. The most convenient method is provided by the Variables window, which is activated from the View -> Other Windows menu or from the same item in the SSIS menu (just ensure that your Designer window is active). Within its interface, you can view, create, and modify properties of variables for the container that is currently selected in the Designer window (with Control Flow tab active, this also includes all of the package level variables) for both User and System namespaces. For example, to define variables for Data Flow Tasks, click on the Data Flow tab in the Designer interface and switch to the Variables window (by selecting Variables from the Other Windows or SSIS menu). Every new variable created at this point, will, by default, have the Data Flow Task scope. Icons in the toolbar of the Variables window allow you to display the content of the System namespace and show all of the variables defined within the package. You can also specify which variable characteristics should be displayed. By default, they include, besides the variable name, its scope, data type, and value, but you can choose to add the name of Namespace and indicate whether an event is raised when the variable's value changes (we will look into this functionality later in this series, when discussing event handlers). For user variables, you can alter their names or values directly in the Variables window as well as modify any of non-Read Only entries in the Properties window.

Another comprehensive view of variables within a package and its components is provided by the Package Explorer tab of the Designer interface. As you would expect, based on the naming convention, the display has a Windows Explorer-like structure, with the Package node as the root. The Variables (containing all package-level variables), Precedence Constraints, Event Handlers, Connection Managers, Log Providers, and Executables are second level folders. The last second level folder, Executables, consists of all of the tasks included in the Control Flow, with Event Handlers and Variables (containing task level variables) subfolders underneath (Data Flow Task contains also Components folder). Once you display the content of any of the Variables folders, you can choose any of its entries, representing individual System and User variables, and use the Properties window to review and manage their values.

Variables can also be managed from the SSIS Designer interface in the context of the container, which constitutes their scope. For example, Execute Process Task Editor (which is part of our sample project, presented in the second article of this series, includes the ability to create new variables when setting Standard Input, Standard Output, and Standard Error entries (although, it is also possible to use any of existing variables for this purpose). If you decide to do so (and select the New Variable option), from the Add Variable dialog box, you set the container, name, namespace, value type, and value, as well as specify whether the variable is read-only. Similarly, in the Editor interface of the File System Task, there exists IsDestinationPathVariable and IsSourcePathVariable properties, along with corresponding DestinationVariable and SourceVariable properties (respectively), allowing you to create new variables (or use existing ones) to store the path to source and destination files. Another example is the Execute SQL Task, where one of the general properties, called SQLSourceType, determines whether a SQL Statement is specified through direct input (in this case, SQLStatement property is used), file connection (which requires connection to a file containing T-SQL code), or variable (which automatically makes the SourceVariable property available, giving you the ability to define a new variable or reuse an existing one).

An SSIS expression is a combination of variables, column identifiers, literals, functions, and operators that all together evaluate to a single data value. Expressions can be used in a number of scenarios, such as determining package control flow at run-time (as part of conditional splits, precedence constraints, or ForLoop containters), controlling data transformations and updates, or assigning values to variables and component properties dynamically. Expressions offer more flexibility than direct values or even non-expression based variables, since they are evaluated at the time of execution of a control-flow task with which they are associated. For example, Precedence Constraint Editor allows you to set the evaluation operation to be a constraint (such as Success, Failure, or Completion), an expression (involving any number of system or user variables), as well as logical AND or OR of both. Note, however, that expressions cannot be used to change properties of Data Flow task or any of its components.

If you intend to assign an expression to a variable (instead of setting its value directly), you need to set its EvaluateAsExpression property (accessible from the Properties window) to True. This will automatically enable its Expression property (and make the content of the Value property irrelevant). Conversely, in order to reference variables in expressions, it is necessary to use specific notation which consists of the @ prefix, followed by namespace and variable name, separated by a pair of semicolons and enclosed within square brackets. For example, when including a variable called Donations in an expression, you would need to type in `@[User::Donations]`.

As already mentioned, expressions cannot only be used to set a value of a variable dynamically - it is also possible to assign them to properties of package components. The simplest way to accomplish this is via the component's Editor interface. For example, when working with Execute SQL Task, you would select the Expressions entry appearing on the left hand side of the Editor dialog box, highlight Expressions listed under "Misc" label on the right hand side, and then click on the ellipsis (...) button. This would bring up the Property Expressions Editor dialog box, with two columns - Property and Expression. An entry in the first one is implemented as a list box from which you can select any of the available properties of the Execute SQL Task component. The second column allows you to either type in an expression that will determine the value of the relevant property or to create it with help of Expression Builder (which you can invoke by clicking on the ellipsis button to the right of the Expressions entry). Expression Builder provides a fairly intuitive interface, which includes a display of available Variables, Mathematical, String, Date/Time, and NULL functions, as well as Type Casts and Operators. Once an expression is constructed, you have an option to evaluate its value with the Evaluate Expression button. The Expressions feature is available for each task within the Designer interface (with the exception of the Data Flow task).

Variables and expressions play a significant role in SSIS For Loop and ForEach Loop containers (which have been introduced in SSIS). The SSIS For Loop implements iterative execution of its subcomponents until an arbitrarily assigned condition, formed by expressions including variables - InitExpression (determining initial value for the loop), EvalExpression (defining evaluation condition), and AssignExpression (which contains assignment expression) - evaluates to False. The ForEach Loop container is more versatile since it can derive a number of iterations based on several types of enumerators, such as rows in tables (via For Each ADO enumerator), files in a file system folder (via For Each File enumerator), objects in a collection referenced by a variable (via For Each From Variable enumerator), XML Path expression (via For Each Nodelist enumerator), or SQL Server Management objects of a specific type (via For Each SMO enumerator). We will be reviewing these containers in more details in the next article of this series.