SQL Server 2005 - SQL Server Integration Services - Part 3

This is the third article of our series discussing SQL Server 2005 Integration Services (SSIS), which provides Extraction, Transformation, and Loading features, replacing Data Transformation Services (DTS) available in SQL Server 2000 and 7.0. We have already presented an overview of the basic concepts necessary to design and implement SSIS-based projects. We have also stepped through the creation of one such project using Business Intelligence Development Studio. Our sample SSIS package, described in our previous article, delivered the basic functionality we needed (running an external process and loading the outcome of its execution stored temporarily in a text file into a database), however, it had one major shortcoming - a lack of support for reusability. If the external process, temporary file, or destination table in the target database had to be changed, it would be necessary to directly modify relevant components in the package. This is cumbersome as well as error prone, and makes code maintenance or versioning difficult. In this article, we will begin discussion of SSIS features, which simplify package maintenance and increase their flexibility.

Before we start looking into these features, let's briefly summarize the ones covered so far, pointing out the differences between SSIS and its predecessor - Data Transformation Services. The most significant one, from the package design point of view, is separation of control flow and data flow, as well as decoupling of the connection information from both. Control flow defines the sequence of package processing, with additional logic dictated by the outcome of execution of its individual components and configuration of precedence constraints. Data flow determines movement (and most typically transformation) of data between its source and destination. Every SSIS package consists of a single control flow, which can contain any number of Data Flow units. Connections, implemented via connection managers, represent various data stores and can be used simultaneously by multiple control and data flow elements. Variables provide runtime location for values utilized by SSIS components, such as containers, tasks, event handlers and precedence constraints, as well as the packages themselves. They facilitate the use of properties which might change from one package execution to another, eliminate the need for using file system or databases for temporary storage, serve as loop counters, parameters of stored procedures, or parts of more complex expressions (used for example by Derived Column Transformation).

In general, there are two types of variables - System and User (which occupy System and User namespaces, respectively). System variables are predefined and their list cannot be modified; user variables, on the other hand, can be created on as needed basis and are modifiable. Both types are limited to the scope of an SSIS component, which hosts their definition (the scope includes automatically all of its subcomponents). This means that while System and User variables on the package level are accessible from any of its components, the ones that have been created within container, task, event handler, or precedence constraint, are not visible outside of it. In cases where variable names conflict, the variabale with more localized scope takes precedence. You can minimize the possibility of such occurrences by introducing custom namespaces (other than the System and User), but only for User variables.

The flexibility, which would allow you to change the properties of the components described above, in order to accommodate changes in external conditions, (e.g. when switching from a development to production environment or when deploying the same package against multiple data stores) can be accomplished though a number of different techniques. However, we will start with the most

straightforward technique, based on the package configurations. Configurations are to some extent similar to the Dynamic Properties Task in SQL Server 2000 Data Transformation Services, since both are capable of modifying package components using external data sources (such as text files or environment variables). The same mechanism can be used to assign values to package variables, further increasing the degree of flexibility. Note, however, that unlike the Dynamic Properties Tasks, SSIS configurations are applied prior to package invocation, so their dynamic changes have no immediate impact (they affect only subsequent package executions).

Configurations are implemented using Business Intelligence Development Studio. With the Package Designer workplace open, select Configurations... item from the DTS menu or from the context sensitive menu of the Control Flow area. This will trigger the Package Configuration Organizer, where you can enable package configurations by selecting the checkbox in its upper left corner. Clicking on the Add... command button will launch the DTS Configuration Wizard, guiding you through the creation of configurations. On the Select Configuration Type page, you need to decide which configuration type to use (XML Configuration File, Environment Variable, Registry Entry, or Parent Package Variable). Your selection determines other parameters necessary to complete this step (e.g. in case of XML configuration file, registry entry, or parent package variables, you need to provide either their location and name directly, or use an indirect approach and specify the name of an environment variable where this information will be stored). File or registry entry is a convenient choice in case the location remains the same across all package deployments, otherwise, the use of environment variable would be a more appropriate option. Storing configuration parameters of properties and variables outside of a package allows you to modify them without the need to use Business Intelligence Development Studio.

Once you choose which configuration type you intend to use, the DTS Configuration Wizard presents you with a list of package components and their manageable properties (its content is the same, regardless of your choice) on the Select Objects to Export page. The list is extensive, covering all Package-level Variables and Properties, its Executables (such as Execute Process Task and Data Flow Task in our example) with their Variables and Properties, Connections with their Properties - all provided in an easy to work with graphical interface. To demonstrate this functionality, we will increase the flexibility of our package by selecting the following (obviously, the selection could be considerably larger, including other properties and variables):

- Executable and Arguments properties of the Execute Process Task (residing under Package\Executables node), which determine the external process to be executed and its arguments,
- ConnectionString property of the Flat File Connection (residing under Package\Connections node), which determine the location of the file where input data resides,
- ConnectionString, Initial Catalog, and ServerName properties of the OLE DB Connection (residing under Package\Connections node), which determine the location of the destination table where output data gets loaded.

After this step is completed, finish the wizard by assigning a descriptive configuration name. You can review the results by analyzing the content of an XML file, registry entry, or environment variable you designated as the configuration store. A package can have multiple configurations, which are loaded in the order listed in the Package Configuration Organizer dialog box (i.e. entries lower in the list take precedence over the ones above them - in case they contain conflicting entries). In order to change the configuration, you can edit its content (using

appropriate editor, depending on storage type) and alter values of properties or variables according to your preferences. For example, if you need to execute the same package in development, test, and production environments, you can create three XML-based configuration files, each containing different values for server, database, and security settings, and copy each to the same local file system location on development, test, and production computers. This way, your package will execute in the manner matching its intended environment, without having any changes applied to it. In addition, the same configuration file can be shared across multiple packages, further simplifying their maintenance.

Another alternative providing the desired flexibility involves package execution using the DTExec command line utility with appropriate switches or using its graphical interface equivalent DTExecUI (both executables reside, by default, in the Program Files\Microsoft SQL Server\90\DTS\Binn folder). DTExec is a replacement for the Data Transformation Services-based DTSRun, providing the ability to execute packages directly from the Command Prompt interface. To access packages stored in the file system, you need to use the /FILE switch (/SQL option is required when dealing with packages residing in SQL Server databases). The /SET option is intended for assigning values to properties of package components and variables at the time of package invocation. Both properties and variables are identifiable by their paths, formed according to specific syntax rules. In particular, the notation starts with the \Package prefix followed by additional elements, pointing either to the target variable or property to be set. The backslash character ('\') is a container separator, period ('.') delimits properties, and members of collections (such as Variables) can be determined by their index, enclosed in square brackets ([]). Finally, the semicolon is placed between the property path and its value. For example, in order to assign the value of "tempdb" to the InitialCatalog property of our OLE DB Connection to the target database, you would need to type the following on the command line (assuming that our package has been saved in the file called SamplePackage.dtsx in the current directory):

```
DTExec /FILE SamplePackage.dtsx /SET
\Package.Connections[ConnectionID].InitialCatalog;tempdb
```

The *ConnectionID* represents the GUID value identifying the OLE DB Connection component from the moment of its creation (you can find this value by checking the content of the Properties Window of the connection). When dealing with variables, the syntax of the path parameter will depend on their scope. For example, if your intention were to assign the value "Thievery Corporation" to a User variable called "Group" defined on the package level and residing in the file Electronica.dtsx, you would run the following from the Command Prompt:

```
DTExec /FILE Electronica.dtsx /SET \Package.Variables[Group].Value;"Thievery
Corporation"
```

When setting a variable on a component level, the path would need to be modified in order to reflect its location within the package hierarchy. For example, assuming that a User type variable named Album, which existed within the Execute Process Task called RunMe in the same package, should be set to "The Mirror Conspiracy," you would be able to accomplish it by altering the path property and value as below:

```
DTExec /FILE Electronica.dtsx /SET \Package\RunMe.Variables[Album].Value;"The
Mirror Conspiracy"
```

In our next article, we will discuss in more detail the use of variables as well as present another method of modifying package properties through Property Expressions.