msdn

Microsoft.com Home | Site Map

MSDN Home | Developer Centers | Library | Downloads | How to Buy | Subscribers | Worldwide

Search for
All MSDN and KB
Go
Advanced Search

SQL Server Developer Center
Learn SQL Server 2005
Reference
Downloads
Support
Community
Product Information
SQL Server 2005 Mobile Edition
SQL Server Express
SQL Server 2005 TechCenter
SQL Server 2000

Microsoft® SQL Server Developer Center

MSDN Home > SQL Server Developer Center > Business Intelligence > Integration Services

See This in the MSDN Library

SQL Server 2005 Integration Services, Part 1: Lessons from Project REAL

Page Options

Richard Waymire

Len Wyatt

John H. Miller

Donald Farmer

Microsoft Corporation

Jyoti Jacob

Scalability Experts, Inc.

March 2005

Summary: In Project REAL we are using large volumes of real data and scenarios from real companies to implement business intelligence systems using early releases of Microsoft SQL Server 2005. In the process, best practices are being developed and potential problems uncovered. This article reports some of the lessons learned while working on the data extraction, transformation, and loading (ETL) portion of the first phase of Project REAL. (61 printed pages)

Contents

Introduction: Project REAL

Project REAL is an effort to discover best practices for creating business intelligence applications based on Microsoft SQL Server 2005 by creating reference implementations that are based on actual customer scenarios. This means that customer data is brought in-house and is used to work through the same issues that the customers face during deployment. These issues include:

- The design of schemas.
- The implementation of a data extraction, transformation, and loading (ETL) process.
- The sizing of systems for production.
- The management and maintenance of the systems on an ongoing basis.

By working with real deployment scenarios, we gain a complete understanding of how to work with the tools. Our goal is to attempt to address the full gamut of concerns that a large company would face during their own real-world deployment. This paper describes some of the lessons that were learned while working on the ETL portion of Phase 1 of Project REAL.

Project REAL uses data from two Microsoft business intelligence customers. Phase 1 of the project was modeled on a large electronics retailer that keeps sales and inventory data in a Microsoft SQL Server 2000 data warehouse. SQL Server 2000 Data Transformation Services (DTS) is used to manage the flow of data into the relational database, and from there into SQL Server 2000 Analysis Services cubes for reporting and interactive querying. This customer maintains approximately 200 GB of data in their relational store. All of this data is subsequently processed into Analysis Services cubes. Phase 1 implementation focuses primarily on the concerns that an existing SQL Server 2000 customer might have when carrying out a migration to SQL Server 2005. Our results largely represent the migration of existing functionality, with a few new capabilities used where appropriate. In the area of ETL, there was a substantial amount of work to do to create packages with SQL Server 2005 Integration Services (SSIS) based on the existing SQL Server 2000 DTS packages.

Note SQL Server Integration Services (SSIS) is the new name assigned to the component formerly branded DTS. The product was renamed after Beta 2; thus many screen shots of the new SSIS in this paper still use the old name.

Because SSIS is based on an entirely new architecture, many concepts and techniques from SQL Server 2000 DTS do not carry over. The process of dealing with these differences is one of the core focuses of this paper.

Phase 2 of Project REAL is based on a larger data set from a different customer, and exercises more of the new capabilities of SQL Server 2005 than does Phase 1. This is because Phase 2 is primarily a new implementation of a SQL Server 2005 solution. Look for more papers about Project REAL in the future.

Project REAL is a joint venture of Microsoft, Unisys, EMC, Scalability Experts, Panorama, Proclarity, and Intellinet. The work described in this paper was done by Microsoft and Scalability Experts.

Phase One Implementation

The Phase 1 customer has two primary sources of information that flow into the data warehouse. These are:

- TLOG files, which are the output of point-of-sale (POS) cash registers in stores (not to be confused with SQL Server transaction log files).
- Flat file extracts from a JDA system (a packaged retail software application), which are used to manage the business.

The overall flow of data, which is managed by SQL Server 2000 DTS, is shown in Figure 1.

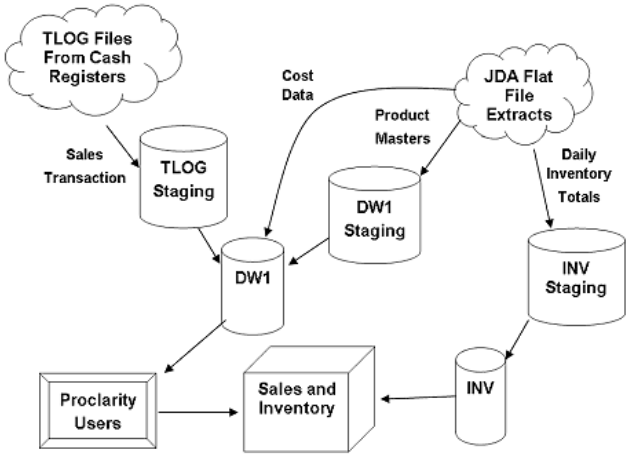

Figure 1

Core ETL Processing

The core ETL processing that is performed to load this customer's data warehouse is as follows:

1. TLOG files come in from point-of-sale (POS) cash registers in a special highly compressed format that must be decoded before they can be loaded into a database. The customer's application compresses these POS transactions into a packed decimal format based on predefined specifications, and stores each file in a separate directory according to the store number they came from.
2. The application assigns each file a name by using a numeric sequence. It stores the name in an appropriate directory according to the store number. This naming convention is necessary to support files that span days as well as to support situations where several files are written per day.
3. The customer uses Perl scripts to parse the TLOG binary files into multiple text files before loading the data into the database. The scripts unpack the data using a predefined template and then uncompress the data according to a set of rules as laid out in an .ini file.
4. The output from the scripts is stored in flat files. The flat files are then read by a DTS package to load the data into the database. This adds an extra step to parse the data as well as a step to load the flat file output before the data can be processed.

To migrate the ETL process to SSIS, Scalability Experts created a TLOG parser that runs in the SSIS pipeline. This effectively eliminates a costly extra step and helps reduce the storage requirements needed to process TLOG files. Each TLOG file, containing compressed data, is now read directly into the SSIS pipeline using a *custom source component* and parsed using a *custom transformation component*.

Note A separate white paper will document what we learned from writing the custom transformation that is required to implement the TLOG parser.

In order to fully exercise the system, the customer provided not only the initial state of the data warehouse, but also three months of daily incremental updates from both the sales and inventory systems. This allowed us to simulate a full processing cycle, as if run over time, including automated partition management in both the relational and OLAP databases.

In addition to the work on ETL described in this paper, the team executed various other activities in Phase 1 of Project REAL, including:

- Migrating the relational data from SQL Server 2000 to SQL Server 2005, preserving the schema precisely.
- Masking the customer's data to protect confidential information.
- Migrating the Analysis Services 2000 database to SQL Server Analysis Services (SSAS).
- Verifying client connectivity using the customer's preferred front-end tools (Excel and Proclarity).
- Creating sample reports against the new cubes using SQL Server 2005 Reporting Services (SSRS).
- Fully implementing a new Inventory cube in SSAS. The customer previously had encountered difficulties working with semi-additive measures because of the high volume of data. Because of this, the customer had stopped work in this area. New capabilities in Analysis Services 2005 now make working with additive measures feasible given larger data volumes.

The remainder of this paper documents what we learned during the process of moving the customer's ETL processing to SQL Server 2005 Integration Services. The paper also highlights numerous best practices.

Upgrade from SQL Server 2000 DTS

If you are an existing SQL Server 2000 DTS customer, one of your first questions might be how you would go about upgrading to SSIS. There are a couple of different options, including using the Migration Wizard, performing a manual migration, or even leaving your packages alone and running them using the Execute Package task. Each of these options is briefly explored in this section.

During Project REAL, we learned that customers are far more likely to be satisfied if they create a new implementation of their ETL process in SSIS, than if they attempt to migrate existing DTS packages. Although we successfully used some upgrade tools (the DTS Migration Wizard) in Project REAL, a majority of our effort involved rewriting various packages in SSIS to take advantage of the improved capabilities of the new software.

Using the Migration Wizard

The Migration Wizard is the most obvious method for migrating packages from DTS to SSIS. This section examines how to use the Migration Wizard, as well as key considerations for when to use or not use the wizard.

To use the Migration Wizard

1. When you create a new project in the BI Development Studio for SQL Server Integration Services (called a Data Transformation Project), a default new package named Package.dtsx is created for you. However, you can safely delete this new package by right-clicking on the package and selecting the **delete** option.
2. Right-click the **DTS Packages** folder in Solution Explorer and select the **Migrate DTS 2000 Package** option. This starts the **Integration Services Migration Wizard**.
3. Click **Next**, then select from where to get your original package(s) (as shown in Figure 2).

   Note You can also manually launch the Migration Wizard. By default, it is located in C:\Program Files\Microsoft Sql Server\90\Dts\binn\Dtsmigrationwizard.exe.

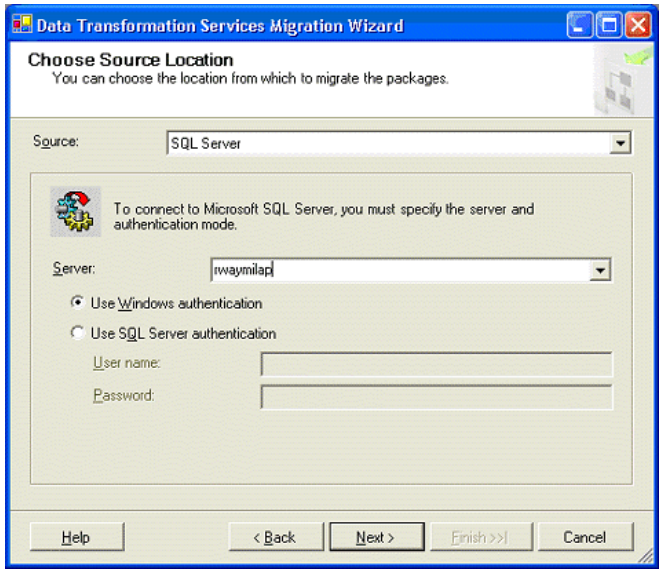

Figure 2

4. Designate a destination location to store the migrated package (as shown in Figure 3).

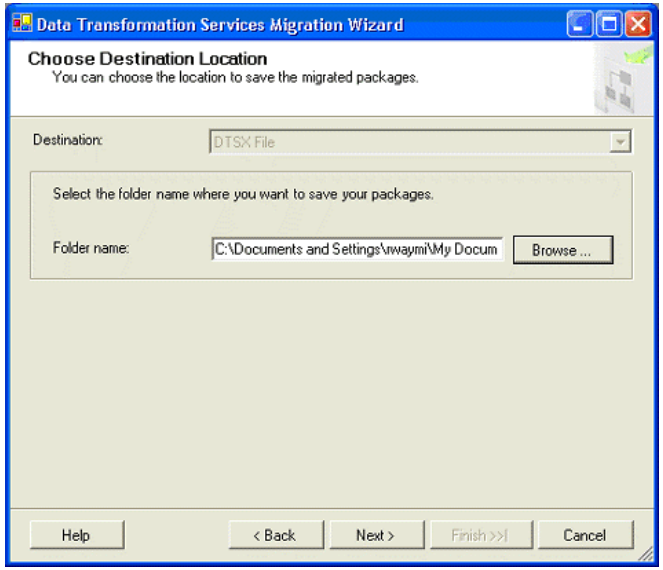

Figure 3

5. Select a DTS package (and version) to migrate (as shown in Figure 4). Note that a package can contain several versions, so you may choose to accept the default (the latest version) or you can upgrade an earlier version of the package.

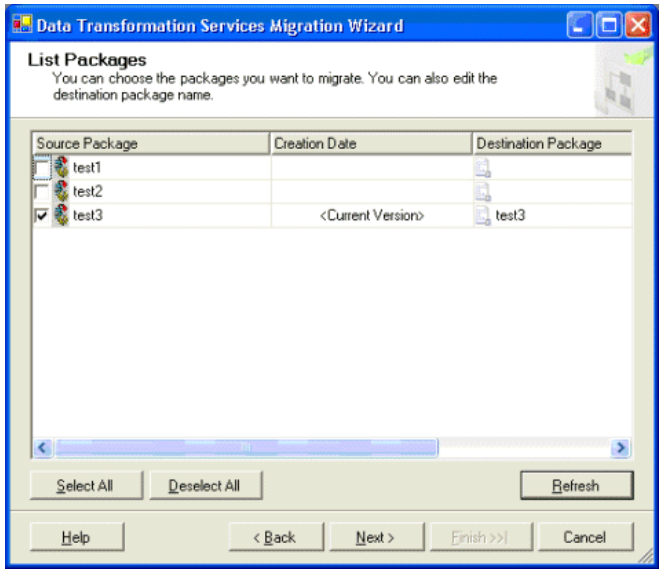

Figure 4

6. Assign a log file to capture the specifics of what the wizard changes during the migration. Review this log file, upon completion, to analyze any warnings or errors encountered during the migration.

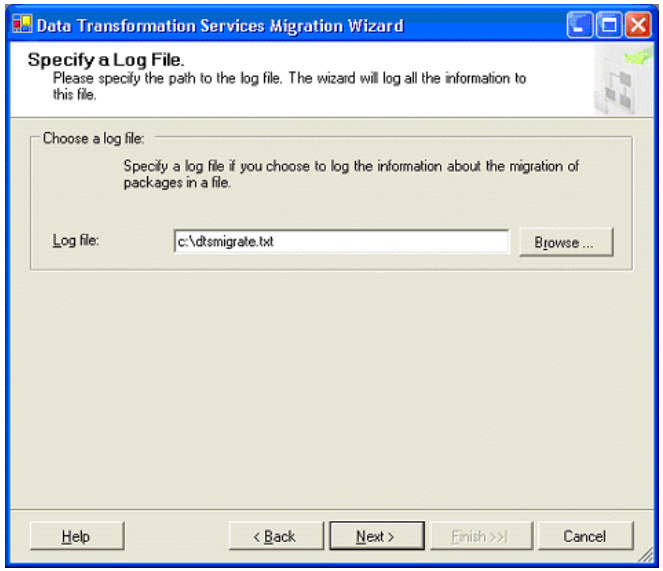

Figure 5

7. Click **Next**, review the summary, and then select **Finish**. You should see a dialog box similar to that in Figure 6, showing that the package has been migrated.

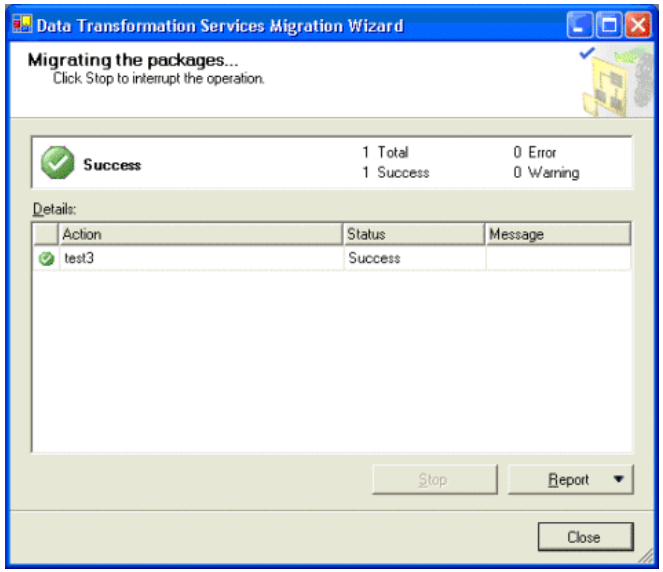

Figure 6

8. Close the wizard and expand the DTS Packages folder in Solution Explorer. Double-click the migrated package to open it in the editor.

   Our upgraded package is shown in Figure 7, and the original package is shown in the SQL Server 2000 DTS designer in Figure 8.

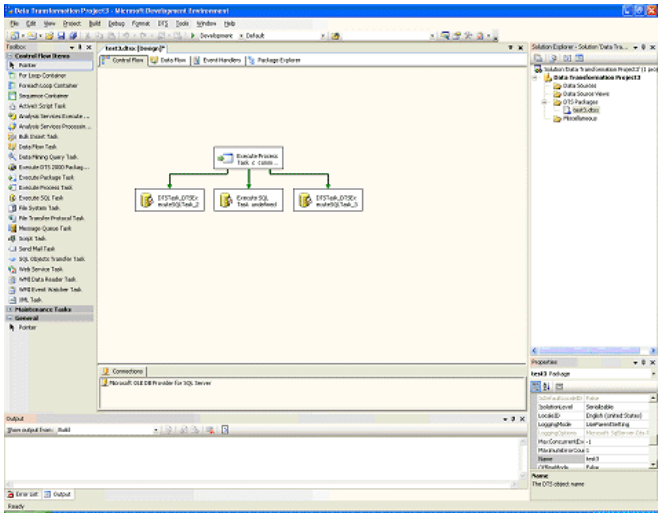

Figure 7

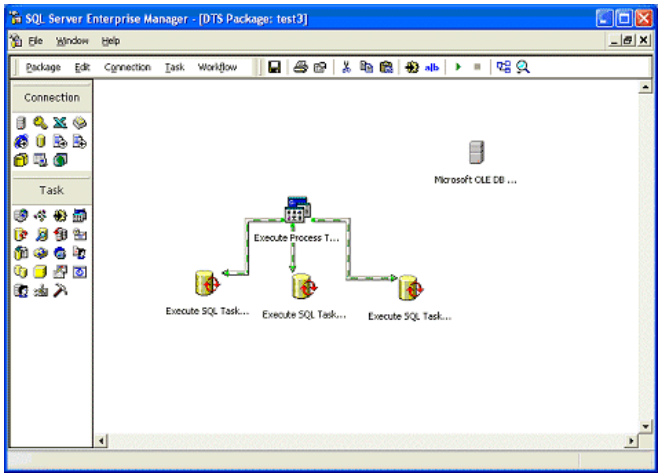

Figure 8

Migration Wizard Considerations

As you saw in the previous section, using the Migration Wizard is not a difficult process. However, there are some additional considerations to keep in mind when using the wizard.

SQL Server 2005 Integration Services is a completely new product. While it is able to continue running SQL Server 2000 DTS packages (we will examine this in more detail later in this paper), SSIS has a new design surface, a new object model, and a new internal design. There isn't necessarily an equivalent or direct upgrade path available for all packages that could be designed in SQL Server 2000 DTS. The Migration Wizard represents a best-effort migration attempt.

If your package contains some of the tasks included in the categories described in this section, you are likely to see errors and failures in the Migration Wizard. These are to be expected, and you will have to rewrite those packages that the wizard is unable to successfully migrate.

SQL Server 2000 DTS tasks can be classified into three categories. Your migration experience will depend on which category your tasks fall into.

Category 1: Simple Tasks

These tasks are a straight port into SQL Server 2005 Integration Services. They include:

- Execute SQL task
- Bulk Insert task
- File Transfer Protocol task

- Execute Process task
- Send Mail task
- Copy Objects task
- Execute Package task

If your package only includes these tasks you should have a good migration experience.

Category 2: Opaque Tasks

These tasks will migrate, but might not work once they have been migrated. This is especially true of the ActiveX and Dynamic Properties tasks. These tasks typically interact with the SQL Server 2000 DTS object model and the SSIS object model is not backwards-compatible with SQL Server 2000 DTS. Opaque tasks include:

- ActiveX Script task
- Dynamic Properties task
- Analysis Services DTS Processing task

You will almost certainly find that migrating packages with these tasks requires the new development of at least these task components, and may entail a deeper review of your package design.

Category 3: Encapsulated Tasks

These tasks are not migrated. When the Migration Wizard is complete, it will have created a new DTS 2000 package that contains these tasks, and then use the Execute DTS 2000 Package task to call into the newly created packages to perform these SQL Server 2000 DTS tasks. They include:

- Custom tasks
- Data Pump tasks
- Data Driven Query task
- Transform Data task
- Parallel Data Pump task
- Copy Database Wizard tasks

For this category of tasks, you must either rewrite the components in SSIS if you want them migrated, or continue to call them using the Execute DTS 2000 Package task.

Be aware that if you plan to run SQL Server 2000 DTS packages on a new server installation with SQL Server 2005, you must choose the **Advanced** option during setup to install the DTS 2000 Run-time engine. If the SQL Server 2000 administrative tools (in particular, Enterprise Manager) are not already installed on the server, this feature is treated as an *optional* component during setup. You must change the setup, as shown in Figure 9, to install the feature.

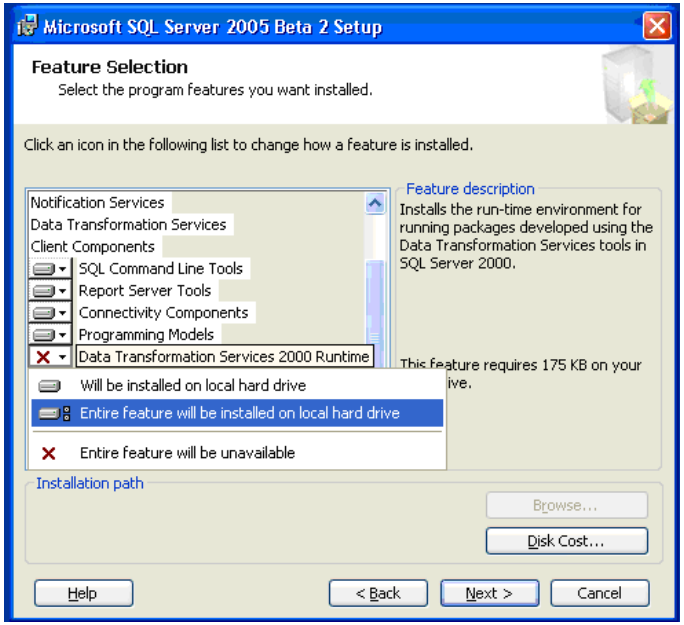

Figure 9

Beta 2 and Community Preview Release Migration Considerations

For Beta 2 and the community preview release (IDW9), there are still some issues with the Migration Wizard:

- A package that contains either the Analysis Services Processing task or the Data Mining Prediction task will not upgrade.
  This should be resolved for Beta 3.
- Code in ActiveX Script tasks are not validated during migration.
  Code used to iterate over objects cannot be converted to loops, which are available in SSIS.
- Transactions are moved from connections to containers.
  No mapping is possible—you must code this yourself in SSIS.
- Custom Transforms should migrate to SSIS.
  If the custom transforms use the DTS object model, they will migrate by means of the Migration Wizard. However, they will not function properly in SSIS.

Performing a Manual Upgrade

Rather than using the DTS Migration Wizard, you may choose to manually upgrade the package yourself. This is, in fact, the preferred mechanism for moving from SQL Server 2000 DTS to SQL Server 2005 Integration Services. Although you can choose to use the Migration Wizard and upgrade some of your existing packages, you will not receive the full benefits of SSIS until you rewrite your packages to take advantage of the new services and capabilities.

This is the path taken in project REAL. Although we chose to keep some of the business logic in stored procedures, and hence superficially the packages in some respects look very similar, in other cases we moved significant amounts of logic out of stored procedure calls and into the SSIS package.

One example of this was the process used by our Phase 1 customer to load sales data. The data was originally loaded using a dynamic Bulk Insert task as one big text string, which was then converted by a select-insert statement into a valid set of typed data. At the time of the conversion, records were filtered out of the loaded data. In some cases, almost 50 percent of the records were filtered out. In SSIS, we opened the data using a flat file connection, and then filtered the data using the same logic as was done originally following the dynamic bulk insert. The data was then directly loaded (properly typed and filtered) into the destination table. This was possible only because of the enhanced capabilities in SSIS.

Running SQL Server 2000 DTS Packages Under SQL Server 2005 Integration Services

You can keep your packages in SQL Server 2000 DTS format, and simply upgrade your installation to SQL Server 2005. In this scenario, your SQL Server 2000 DTS packages continue to function as before. They are run from a controlling SSIS package that calls the Execute SQL Server 2000 DTS Package task.

Figure 10 shows a sample connection in the package task to a DTS 2000 package. You can even edit these packages in the SQL Server 2005 SSIS editor (at least as of Beta 2) by selecting the **Edit Package** button in the task editor.

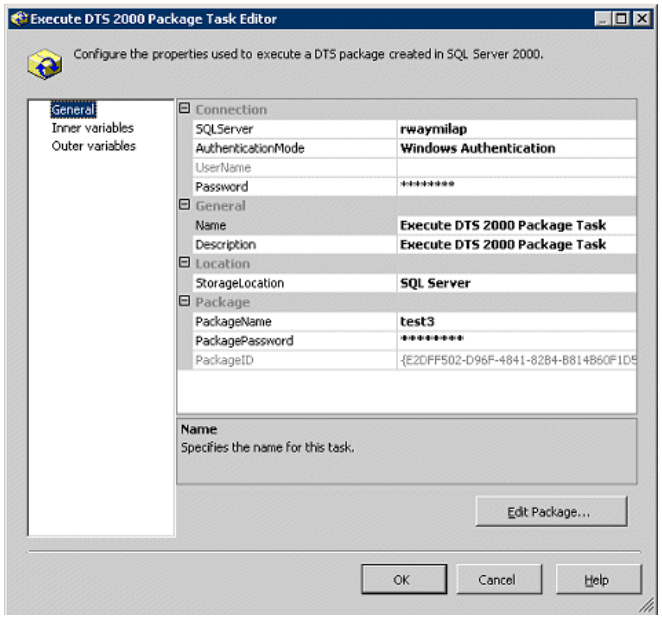

Figure 10

This loads the SQL Server 2000 DTS editor for the package (see Figure 11). It is important to note that this only works on a server that previously had the SQL Server 2000 Enterprise Manager installed. The SQL Server 2005 Setup program does not have an option to install the DTS 2000 editor.

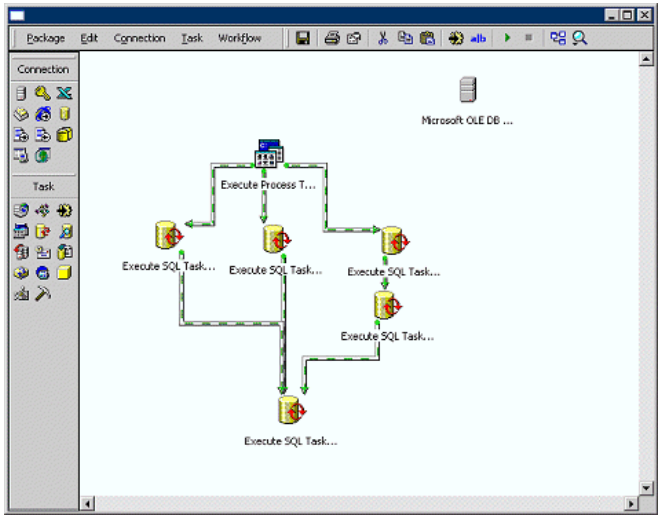

Figure 11

This may be the best short-term migration strategy for many customers. This enables you to upgrade your SQL Server installation to SQL Server 2005 and continue running your existing DTS 2000 packages. Over time, you can migrate each package (either using the Migration Wizard or by rewriting them) to take advantage of enhanced SQL Server 2005 capabilities.

Lessons Learned While Developing SQL Server 2005 Integration Services Packages

As we worked through the first phase of Project REAL we discovered several useful best practices, as well as a few problems with the product. These are documented here.

Implementation Best Practices

These represent the best practices we discovered while working on Phase 1 of Project REAL.

Adding Logging to Your Packages

Using logging within your SSIS packages is considered a best practice. So, how do you add logging? In SQL Server 2005, this is trivial.

To add logging to packages

1. Right-click anywhere on the control flow design surface and select **Logging** as shown in Figure 12. This launches the **Configure DTS Logs** dialog box.

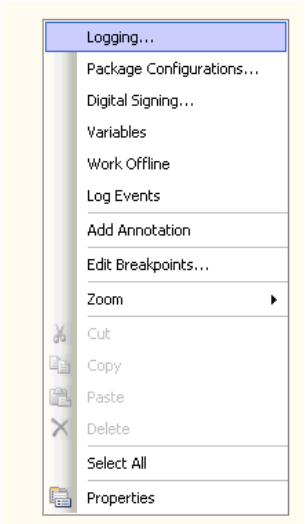

Figure 12

Note In the community preview release (IDW9), you must check the package in the tree control on the left side of the screen. Then, click **Add** (next to **provider:**) to add a log file.

Log File Types

In addition to logging results to a basic text file, you can optionally choose to log to:

- A SQL Server Profiler
- a SQL Server table
- the Windows 2000 Server Event Log
- an XML file

Which type of logging should you use?

- The SQL Server Profiler log allows you to open the log file in the SQL Server Profiler GUI. That, combined with the ability to load performance monitor counters, could give you a powerful analysis tool in some situations, such as if your run-times are unexpectedly long.
- A SQL Server table allows you the full power of the relational language to examine the logs, and allows you to use SQL statements to take actions that are based on log entries.
- The Windows Event Log may be your best choice if you are using operations management software such as Microsoft Operations Manager to monitor your servers. You can have alerting and actions taken based on log entries from your packages. This may be a good option to consider using once you've moved the application into production.
- The XML file format is handy if you want to browse the log visually or, should you choose, you can write an XSLT transformation to enable viewing the log as a Web page. XML also offers a nice intermediate format if a need exists to share results with external parties, or should you want to consolidate log files from various data sources, including SQL Server.
- A text file is simple to view, simple to configure, and simple to define, particularly if you're simply doing the basic testing phase of a package.

2. After clicking the **Add** button, check the DTS log provider (as seen in Figure 13). Click in the **Configuration** column and a list opens. Select **<New Connection>** in the list, as shown in Figure 13.

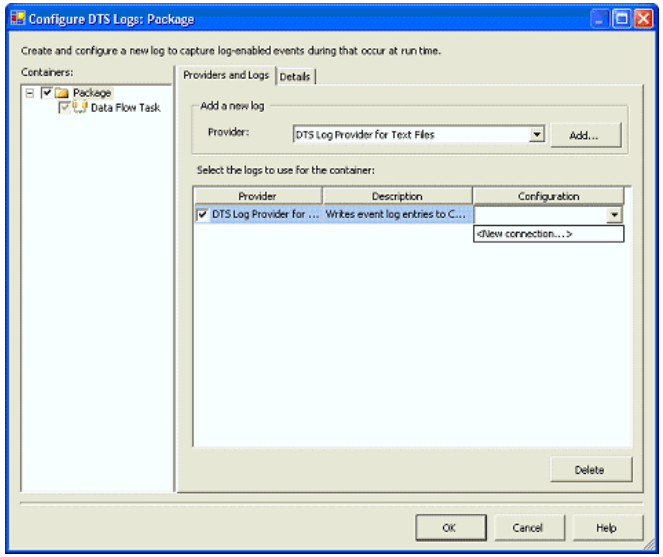

Figure 13

This opens the **File Connection Manager Editor** dialog box as shown in Figure 14.

3. In this example, I opened the **Usage type:** list, selected **Create File**, and input a file name to act as my log (see Figure 14).

   Notice that this shares a common graphical interface with SQL Server Profiler because both are based on a common working infrastructure. This makes logging easier to understand and more consistent throughout SQL Server.

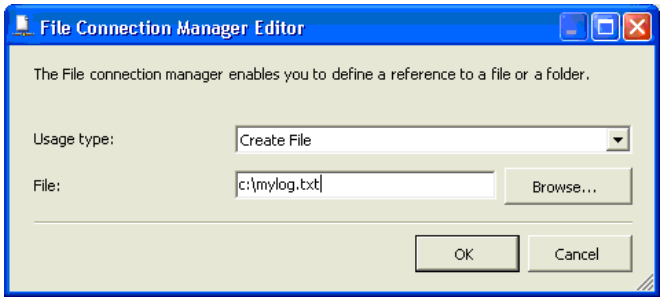

Figure 14

4. Click **OK**. You've successfully added logging to your SSIS package and will be able to review the log after the package has run.
5. To configure the details you want logged, click **OK**, then click the **Details** tab, as shown in Figure 15. This lets you choose which event types to capture in your log.

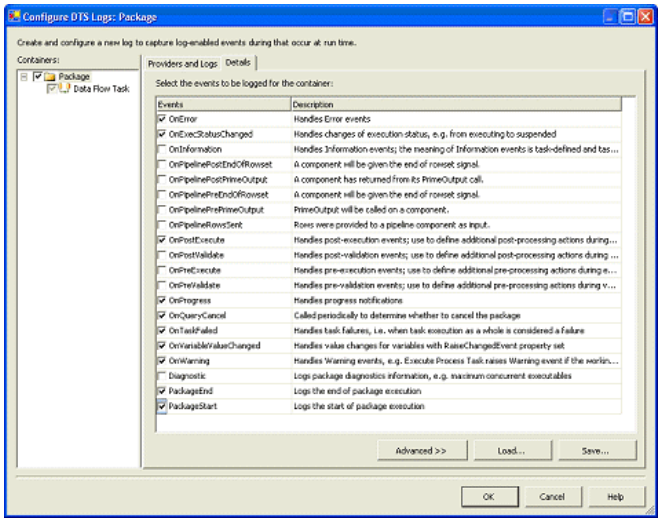

Figure 15

Note if you click the **Advanced** button, you can get very granular in the events you choose to log (as shown in Figure 16).

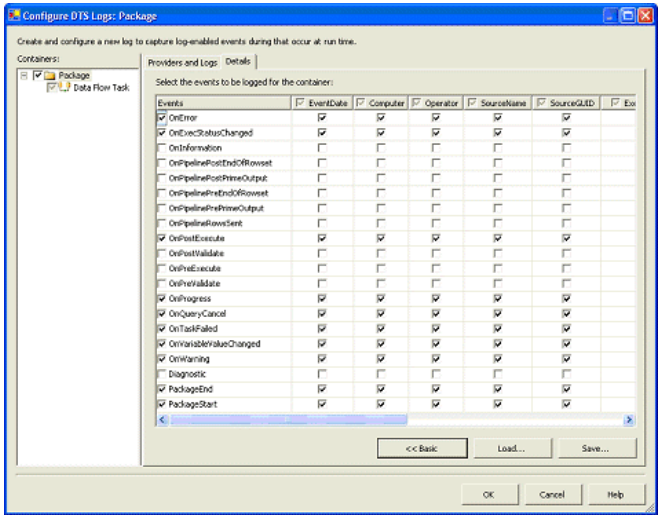

Figure 16

Using the Execute SQL Task

The Execute SQL task allows you to set parameter values in the query and return a single row, multiple rows, or output to XML.

To ensure good performance while using this task, you should optimize your queries by using traditional query optimization tools, as SQL Server Integration Services will not optimize your queries for you. This is true even when extracting data from SQL Server. To optimize performance, return only the columns that are absolutely necessary in your queries. Queries such as **Select * from <sometable>** should be avoided because unnecessary data will be returned.

You can specify parameterized queries in the Execute SQL task. Input variables for stored procedures and queries are mapped to the value of **?** at run time.

```
Select EmployeeName from dbo.EmpTable where EmpID = ?
```

Parameters are order-dependent. In the case of a stored procedure with several parameters, the SSIS engine follows the order in which input variables are assigned and will map them accordingly. So your first input variable is mapped to the first **?**.

If a complete result set (more than one row) is returned by the query, set the **Result Name** to **0** to assign the value to a variable, as shown in Figure 17.

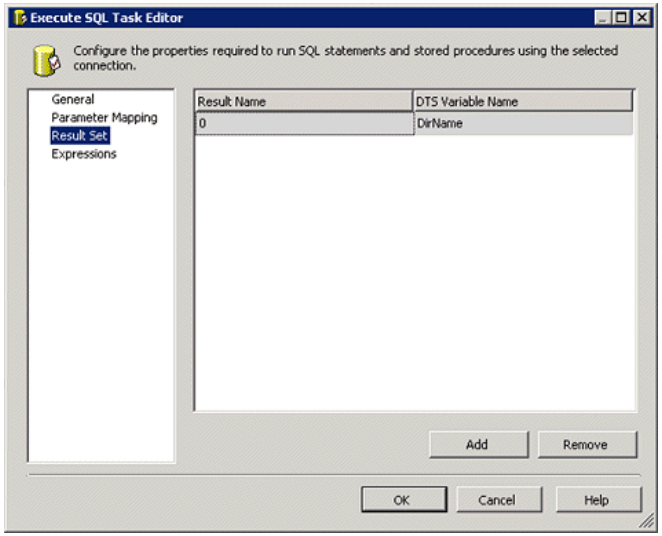

Figure 17

In the case of single row return, set the value of **Result Name** to the name of the column(s) returned.

For error handling that involves stored procedures, map the **ReturnValue** direction in **Parameter Mapping** to a variable, so that you can check the return code following execution of the stored procedure.

Working with Files That Have a Variable Number of Columns on the "Right" Side

Some files, such as the text files we are using as part of Project REAL, have a variable number of columns at the end of the file. Sometimes optional values are present at the end of a data file row, and sometimes they are not. SSIS can work with this type of file by using the Flat File connection manager, with the RaggedRight format. However, there is a limitation when using this file format. Only the LAST column is allowed to be variable. In other words, you can't have multiple columns defined in your file format that are optional.

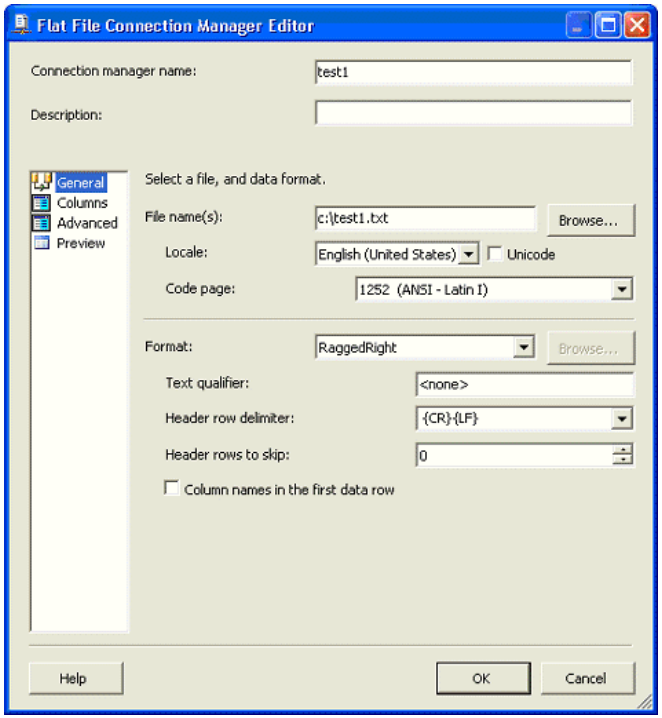

Figure 18

There is, however, a straightforward way to work around this—you can define the last column as the maximum length of all optional columns, then use the Derived Column editor to create the column definitions of the optional columns for further use in your package. Figure 18 shows a definition of a Flat File connection manager connection to a file, and in Figure 19 you can see that we have defined the last column, **variablecol**, as 50 bytes.

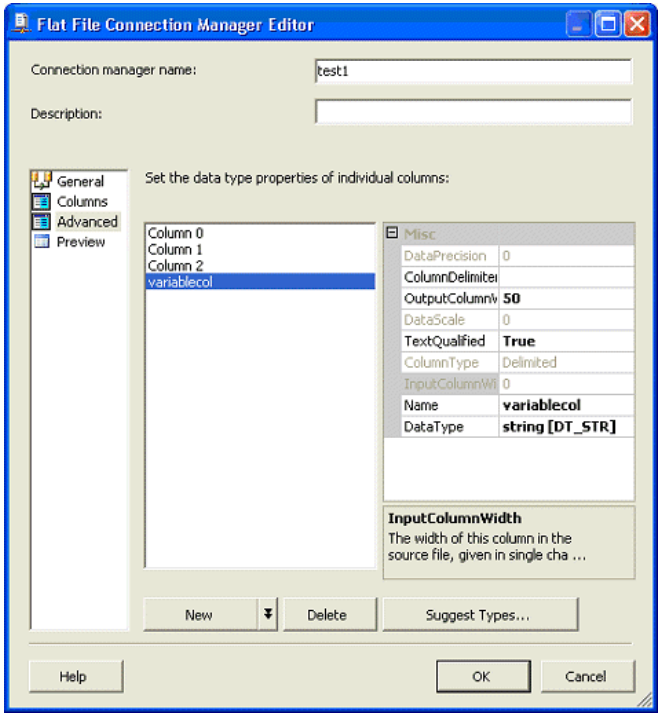

Figure 19

Now we'll add a Flat File source to reference the flat file connection manager connection we have just created in the **connection manager** list. Next, we need to add a **Derived Column** Data Flow Transformation, and connect the flat file source to it. Double-click the **Derived Column** transformation, and then define what your optional columns will look like if they exist, as shown in Figure 20.

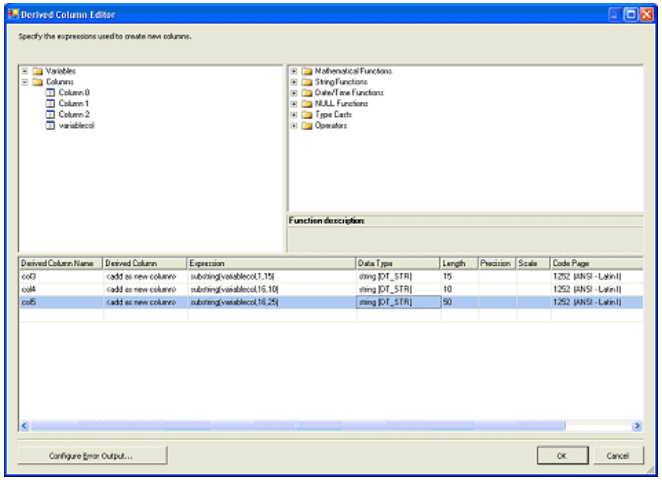

Figure 20

Next, add a destination, such as an SQL Server destination. You can now reference all of your columns, even those that may or may not be present in every single row of the file. The finished data flow will look something like Figure 21.

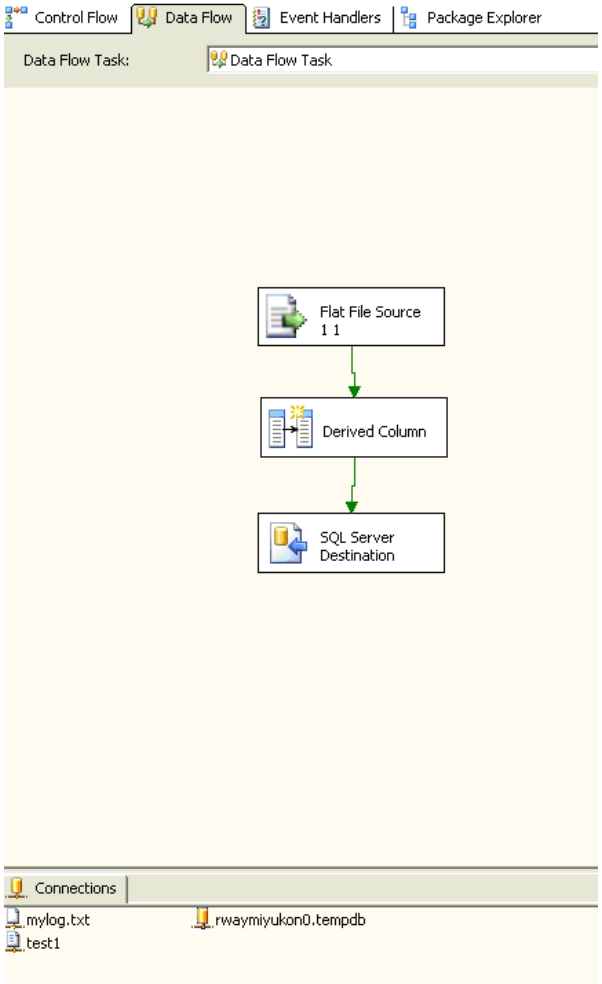

Figure 21

Transforming Data Using the Derived Column Transformation

A Derived Column transformation allows the user to create new columns by applying expressions on the input. The expressions can be applied on both input and variables. SSIS has its own set of expression grammar defined to allow the user to transformation input. This transformation is very useful for the following tasks:

- If the source contains variable numbers of columns and column lengths. As discussed above, the user can read the source data as a single column, using the ragged-right option in the Flat File source, and then apply a Derived Column transformation to extract multiple columns from a single large column.
- When new columns need to be derived based on one or more source columns to support downstream processing.
- During ETL processing, it is not uncommon that external data will need to be cleansed before inserting it into the database. Examples:

  String column: leading null characters need to be trimmed; only a substring of the source column needs to be extracted.

  -or-

  Int column: The data needs to be an absolute value; the data needs to be a rounded value.

This can be achieved easily by using the various string or mathematical functions in a Derived Column transformation. Hence, Derived Column provides a convenient way to replace complex stored procedure logic traditionally used to perform data cleansing or transformation.

Derived column

As shown in Figure 22, a number of complex expressions can easily be applied using multiple functions defined within a single transformation. For instance, our second expression employs a **Case / Switch** statement that is dependent on the results of a

Substring function to cast results into the desired data type.

Note The expression syntax used in this and other tasks within SSIS is a custom implementation that is not based on either Transact-SQL or Visual Basic. See SQL Server Books Online for full documentation on the expression language.

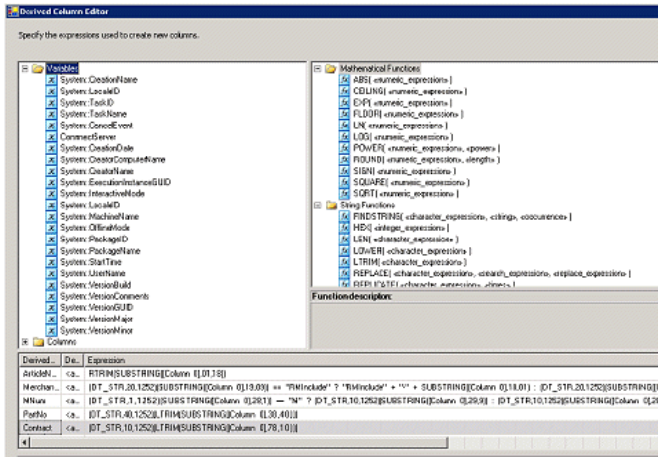

Figure 22

To use a Derived Column transformation

1. Drag the Derived Column transformation to the Data Flow window and double click to open it.
2. Add the desired derived column and define an expression to assign a value to the derived column. The expression can be written on the input or on a variable.
3. In the **Derived Column** list, define whether to **<add a new column>** or **<replace the old column>**.
4. For new columns, specify the appropriate data types with the length, precision, scale, and code page.

The Derived Column transformation allows the user to configure how the row-level component should be handled in case of an error or truncation of data.

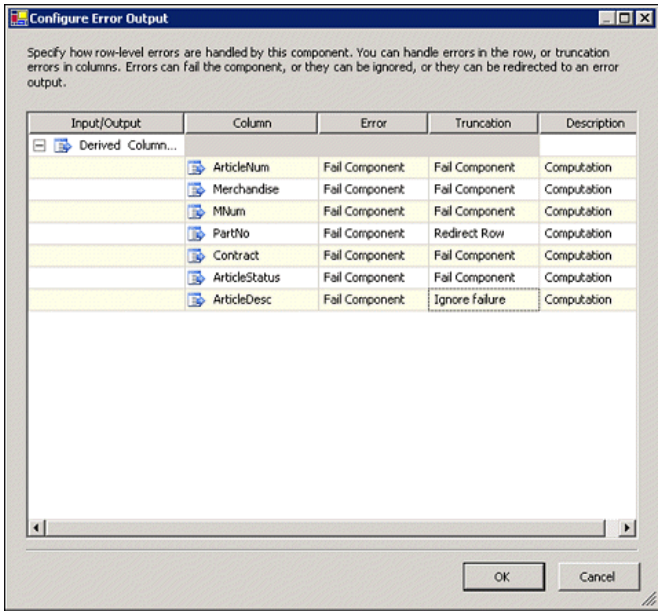

Figure 23

As shown in Figure 23, the user can define different actions in case of an error or the truncation of the column data for each column. For **ArticleDesc** we choose to ignore the failure in case of truncation, while for **PartNo** we decided to redirect the row to an error file. For the rest of the columns, the component will fail if either an error is encountered or the data is truncated. An output **Error** column is generated; this can be mapped to an error-handling component downstream for processing.

Using Sequence Containers

Multiple tasks that run in parallel, as shown in Figure 24, are commonly seen in SQL Server 2000 DTS packages.

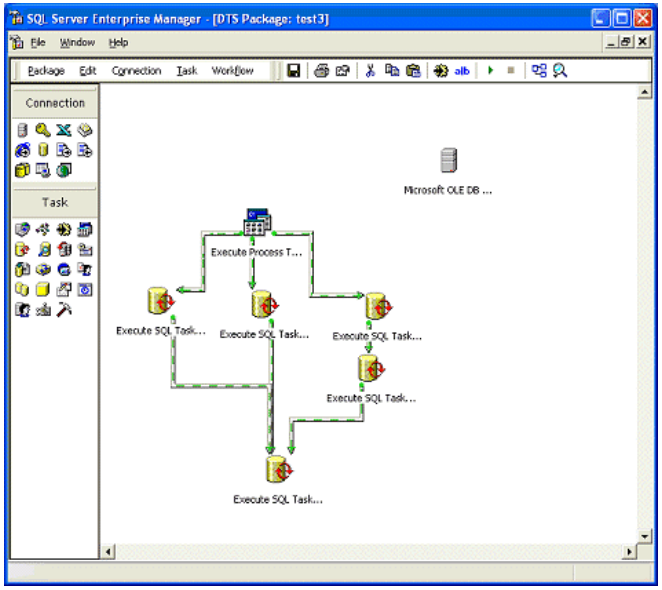

Figure 24

With SQL Server 2005 Integration Services, you can wrap these tasks together into a *Sequence container* so that you don't have so many workflow connections in the designer. Additionally, you can have tasks that have dependencies within the Sequence container, as well as make tasks further downstream in the workflow depend on everything that happens in the Sequence container (see Figure 25 for an example of using a Sequence container). The Sequence container will act as a single task in terms of input, output, and error handling.

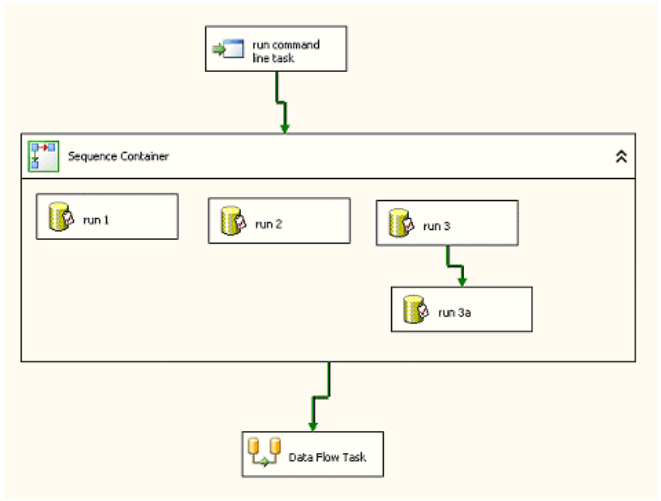

Figure 25

Looping In SSIS

In DTS 2000, to enable looping through the task, the developer had to implement what amounted to a "hack" to fool the runtime engine into thinking the task was still waiting following completion of the task. Thankfully, in SSIS, no such "hack" is required to enable looping. In fact, several looping tasks, ForEach Loop and For Loop, have been introduced.

For Loop

The For Loop container defines an iterative workflow in a package by using the for iteration statement. The For Loop container evaluates an expression and repeats its workflow until the expression evaluates to **False**. When the task is iterating, multiple tasks can be executed in a loop as shown in the following figure.

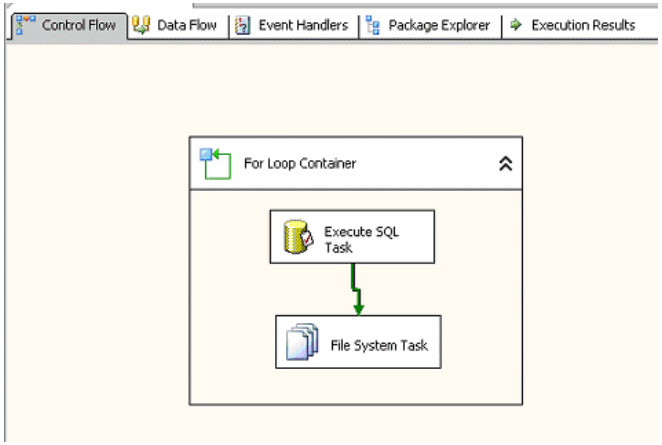

Figure 26

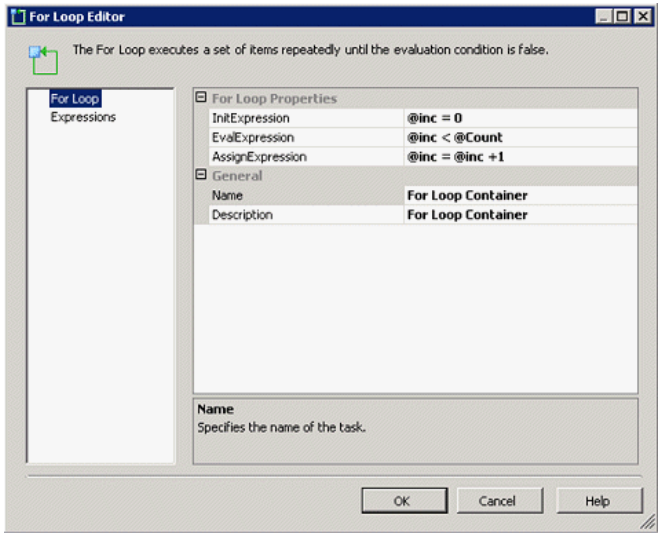

Figure 27

ForEach Loop

The Foreach Loop container defines an iterative workflow in a package by using the foreach iteration statement. DTS provides the following enumerator types:

- For Each ADO enumerator, which enumerates rows in tables.
- For Each File enumerator, which enumerates files in a folder.
- For Each From Variable enumerator, which enumerates the enumerable object that a specified variable contains.
- For Each SMO enumerator, which enumerates SQL Server Management Objects (SMOs).
- For Each Nodelist enumerator, which enumerates the result set of an XML Path Language (XPath) expression.

The For Each Directory enumerator will be added to the above list by Beta 3. This enumerator was not planned initially. One of the requirements in project REAL was to enumerate over store directories to access TLOG files for processing and insert the data into the database. This requirement uncovered a real-world scenario using the ForEach Directory enumerator we had missed. Fortunately, it was considered a common enough scenario that the decision was made to include it in SSIS by Beta 3. Until then, a For Each Directory enumerator custom task is available in the DTS samples.

It is important to note that SSIS was designed in such a way that if a task that you desperately need is not available, it's relatively easy for you to create a custom task and include it for use in constructing SSIS packages. Such extensibility is encouraged and

it's probable that you may see third parties create custom tasks to augment the base set supplied by Microsoft.

Using the Deployment Utility and Configurations to Ease Movement of Packages Between Servers

Another issue we encountered was the challenge of moving packages between environments (for example, moving from development to test to production). Connection strings to servers, hard-coded file locations, and others are often dependent upon the physical server or network that is used in the package development environment. When you move packages to a different server or network, these connection strings may no longer be valid. There is a great new capability in SSIS to help with this, but getting started with it is a little tricky.

First, make sure that all the files you would want to have along with your actual package (dtsx) files are part of the project in Solution Explorer (you can add extra files to the Miscellaneous folder in your project). Once that's done, use the SSIS configuration utility to allow parts of your package to be configured by an input source (perhaps an XML file). Then, you run the SSIS deployment utility to build an executable setup to install the package, including your updated configurations, onto a new server.

To Access the SSIS Configuration Utility

1. After you have your package built, right-click the design surface of the control flow and select **Package Configurations**....
2. Make sure that **Enable Package Configurations** is checked, and then click **Add**. You should see something like Figure 28.

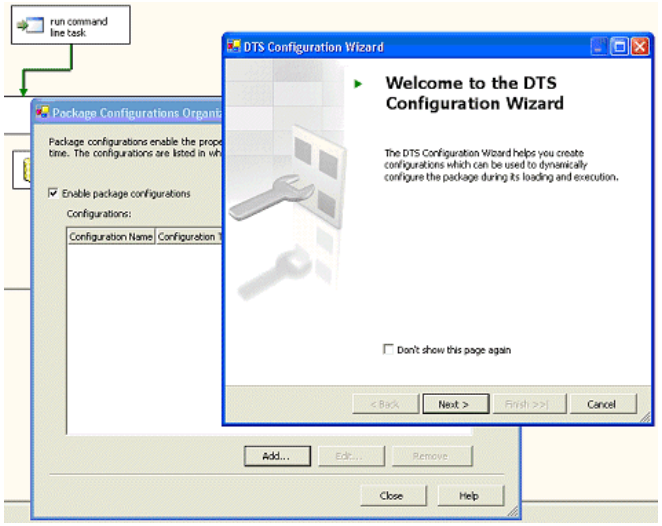

Figure 28

3. Click **Next** and select the type of configuration to use (we'll use an XML file, the default). Specify the correct information for the configuration type you've selected (in our case, a file path to the XML file containing the configuration details). You should see a dialog box similar to that shown in Figure 29.

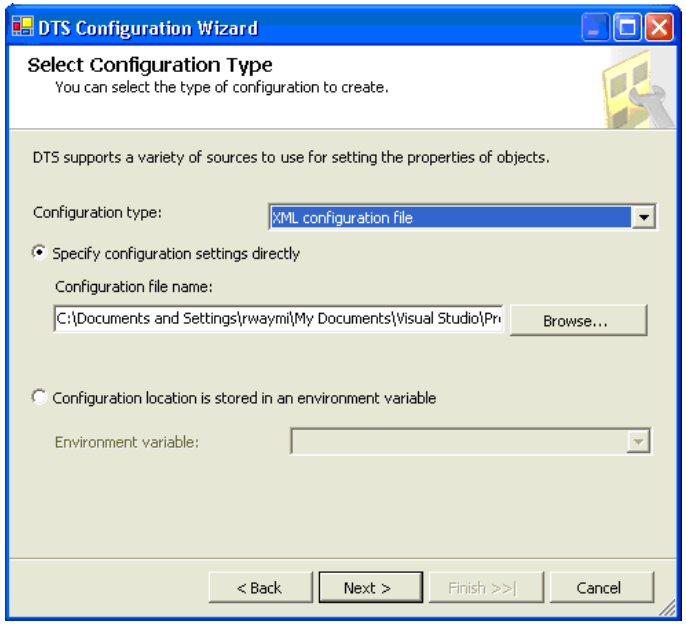

Figure 29

4. Click **Next**, and the **Select Properties to Export** dialog box shown in Figure 30 is displayed.
5. Set the properties in the **Select Properties to Export** dialog box as necessary. This is the most confusing part of this process. You have to know what things are likely to change when moving from one server to another. I selected the connections to my files and SQL Server, as they're likely to be different between test and production systems. As you scroll through the list of available objects, I think you'll be impressed by how much you can configure at install time. The most likely items you will want to configure (and the ones selected for this configuration) are the connections to files and databases.

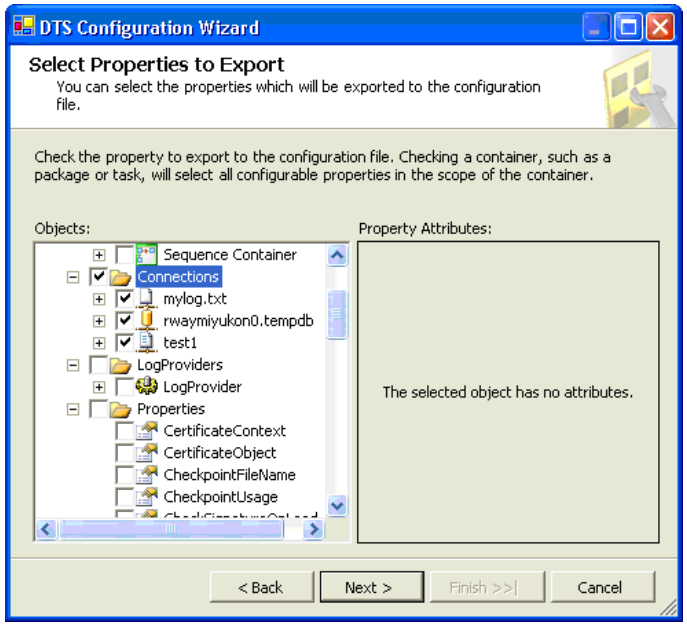

Figure 30

6. For this example, click **Next**, then **Finish**, then **Close the Package Configurations Organizer**. Save your project.
7. Right-click the project properties in Solution Explorer, and select **Properties**. Click the **Deployment Utility** node of the tree control on the left, and you will see a dialog box similar to the one in Figure 31.

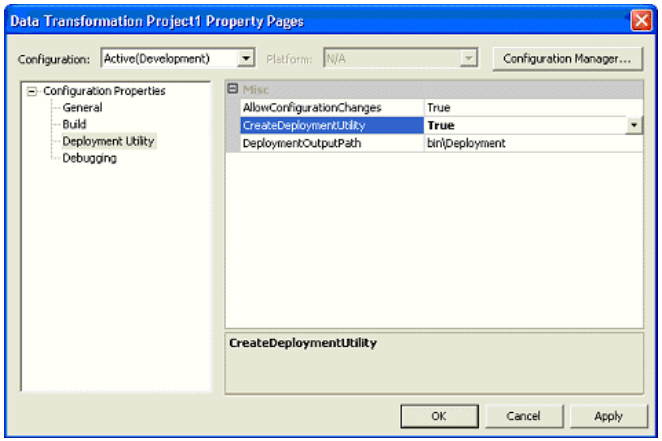

Figure 31

8. In this dialog box, change the **CreateDeploymentUtility** option to **true**. Make sure that the **AllowConfigurationChanges** property is set to **true** as well, then click **OK**.
9. Select **Build** → **Build Solution** from the menu. Make sure the package builds successfully.
10. Navigate in Windows Explorer to the directory where your package was built, then to the \bin\deployment directory below it (in my project it was in **My Documents** → **Visual Studio** → **Projects** → Data **Transformation Project 1** → **Data Transformation Project 1**), as shown in Figure 32. Notice the configuration file along with your package and an installer executable.

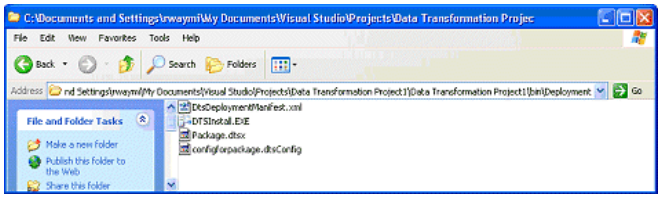

Figure 32

11. Copy the contents of this entire directory to the server you intend to move your package to.
12. Double-click **DTSInstall.exe**. The DTS package installer starts.
13. Click **Next**, and you see the **DTS Package Installer** dialog box as shown in Figure 33.

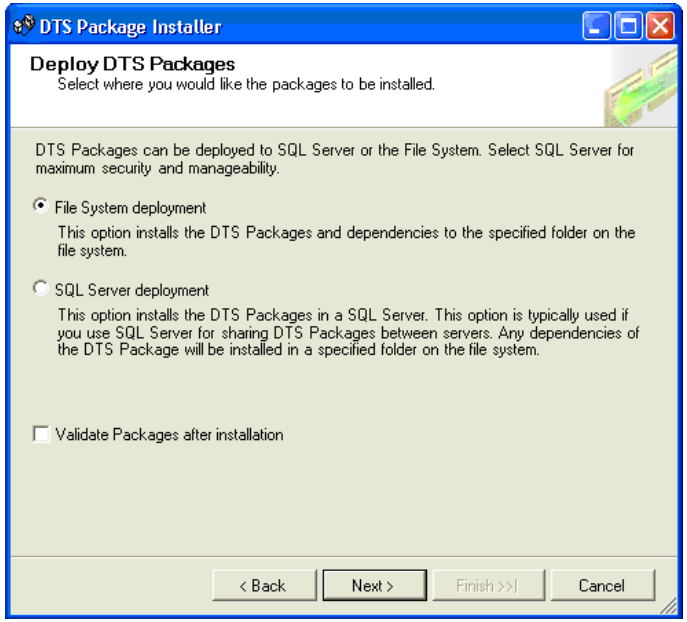

Figure 33

14. Select the type of deployment (copy the package to the file system or to a server running SQL Server). For this example, we'll select **File System deployment**.
15. Click **Next**, and then select the path to deploy to, as shown in Figure 34.

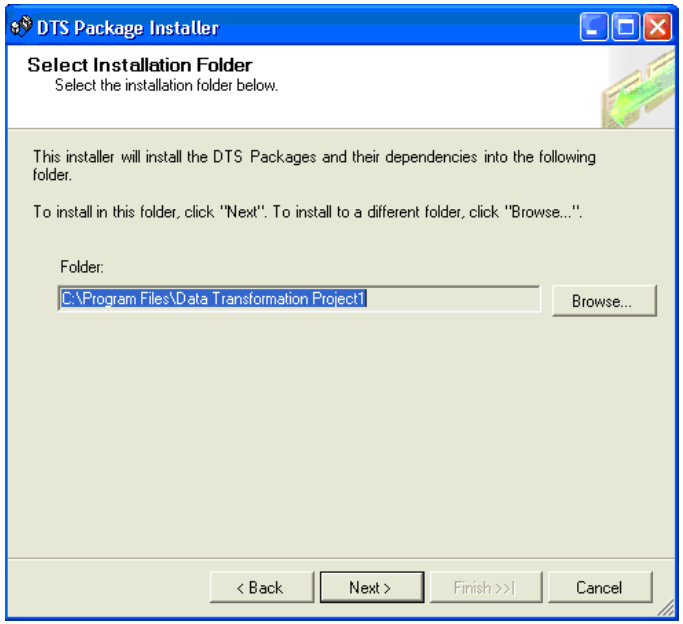

Figure 34

16. Click **Next**. In this dialog box you can change configuration options, such as your connection strings to your SQL Server installation. Notice, for example, in Figure 35 that I've changed the SQL Server data source name from rwaymiyukon0 to FRED.

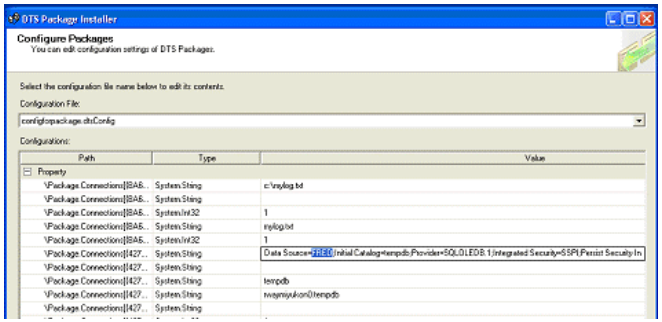

Figure 35

17. Click **Next**, then **Finish**.

    Your package will be deployed along with the configuration modifications you've requested the DTS package installer to make to your package.

Property Expressions

Configurations are loaded and applied to the package once, at the beginning of execution. It's important to note that configurations are applied before execution, not dynamically while the package is running. Sometimes you want to dynamically change properties during execution. Property Expressions have been introduced to allow a user to dynamically set component properties at run time using variables.

A Property Expression allows the user to set the variable values at run time with minimum effort and to pass the information from component to component using these variables. This allows easy manipulation of variables, even for non-developers. Each task contains the Expression feature that allows the user to set the properties of the component using different expressions. The UI for the Property Expression is similar to the Derived Column task UI (see Figure 36).

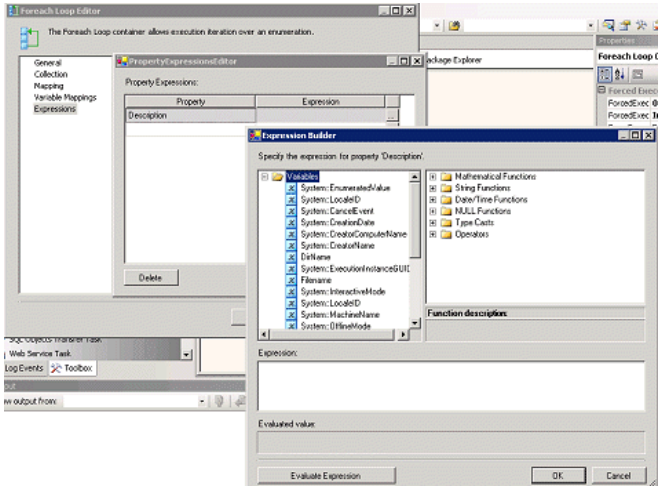

Figure 36

In our SSIS package, for the Foreach File enumerator, we wanted to set the folder name property dynamically at run-time but could not find these properties in the **Expressions** property list. Later we found out that the enumeration configuration properties can be set in the Property Expression Editor found in the **Collection**.

Figure 37 shows an example of setting the **DestinationTableName** property of a Bulk Insert task.

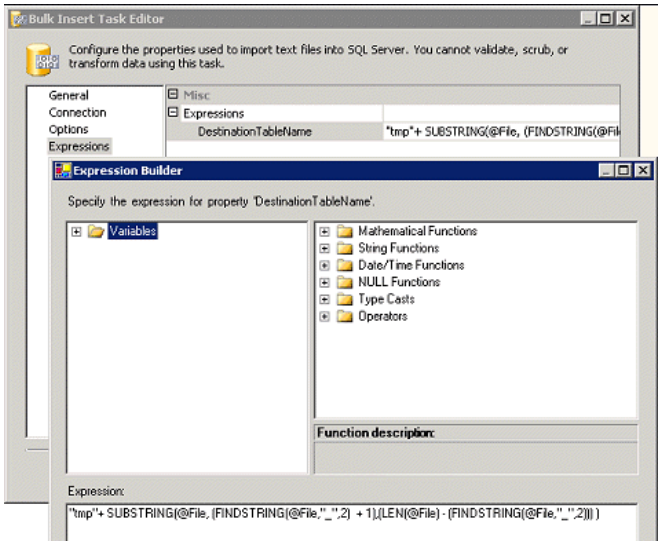

Figure 37

For die-hard fans of the Dynamic Properties task in SQL Server 2000 DTS, the Property Expression feature may be your answer for SSIS.

The variables in SSIS have set scopes within the container in which they are defined. In SQL Server 2000 DTS, all the variables were global variables, which is not the case in SSIS. Hence, when a variable is assigned to a property in a Property Expression, the variables defined in the child containers cannot be used. Only variables that are defined at the container level or above can be used to set the property expression. This is true even in error handling. Another important criterion to take into consideration is the data type of the variable. The variable needs to be cast to the same data type as the property before assigning.

Avoid the Pains of a Restart After Failure

When we started testing the package, the frustrating part came when the data was loaded successfully in the staging database and then the package failed. The Checkpoint feature came as a blessing to resolve this problem. When the Checkpoint feature is enabled, SSIS starts logging the information into a checkpoint file up to the point of failure. When the package is rerun, the Runtime engine uses the information in the checkpoint file to restart the file from the point of failure.

To enable the restart of packages from the point of failure, you need to set the following properties:

- In the Custom Flow task window, set the **SaveCheckpoint** property to **True**.
- Specify the location of the checkpoint file in the **CheckpointFileName** property.
- Set the **CheckpointUsage** property to one of the two values:

  Always: Restarts the package from the checkpoint always.

  IfExists: Restarts the package from the checkpoint if it exists.

- Configure the tasks and containers from which the package can restart.

  On the task or container, set the **FailPackageOnFailure** property to **True**.

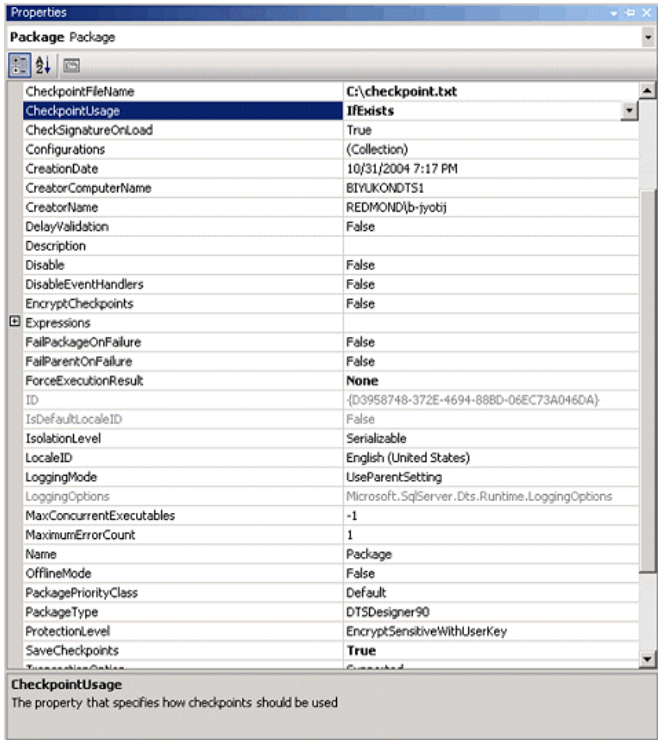

Figure 38

Something to note is that the Foreach Loop container and any transacted containers are considered as atomic units of work. Any child containers in the Foreach loop are not recorded in the checkpoint file. Hence, if a package is restarted on the Foreach task, all the child containers in the Foreach loop task are also restarted, even if they may have completed successfully before failure. For example, consider a package that extracts the file names from a table by using the Execute SQL task, and uses the ForEach File enumerator task to loop over the files, and implements a bulk insert task to insert the extracted file data into table. If the package fails on the Execute SQL task and if it is set as a task to be restarted, the package starts at this task. But if, after looping over a few files using the ForEach File enumerator and loading data using the Bulk Insert task, the task fails, on restart the package executes the ForEach loop container and also the Bulk Insert task, contained inside the ForEach loop task.

Precedence Constraint Editor

The Precedence Constraint editor controls the flow of data to the next component by setting different conditions and constraints on the precedence constraint. This is useful in the case of conditional flow, when you want a particular task to execute only if a particular condition is satisfied before some other task should execute. As shown in Figure 39, we wanted the next task in the flow to execute only if the previous task execution result was successful and the value of the variable **@CountPartitions** (set dynamically during run time) was equal to zero. This was achieved by setting the constraints as shown in Figure 39.

If multiple precedence constraints are set on a task, the user can specify whether all the constraints need to be to be met before the next task can execute or if satisfying any one condition is fine. In the example below, we set the property such that all the precedence constraints must evaluate to **True** to execute.

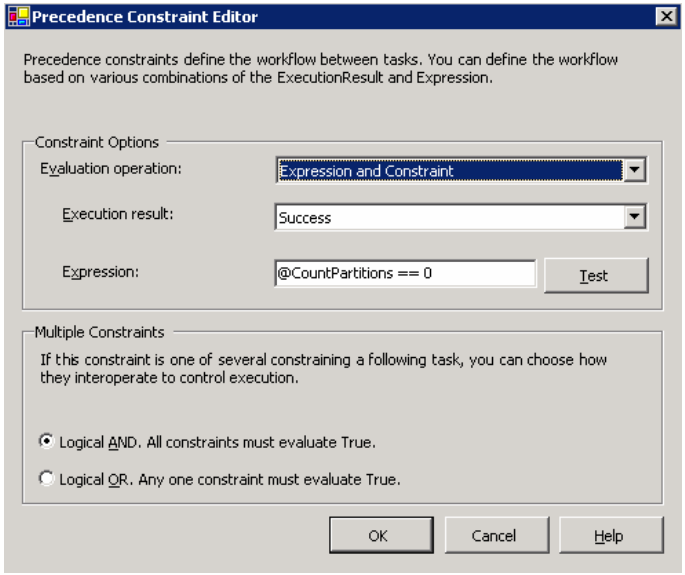

Figure 39

Package Execution

At run time, the DTS Runtime engine executes each task in the package in the order specified by the package workflow. DTS packages can be run from DTS Designer in Business Intelligence Development Studio, in the DTS Import/Export Wizard, and by using the DTS package execution utilities.

While running the package in the Business Intelligence Development Studio, you can also debug the package. BI Development Studio offers an environment similar to that of Microsoft Visual Studio for debugging the package. While testing the package, I was able to resolve many of the errors by putting breakpoints at specific locations, and monitoring variables by setting watch on the variables. The breakpoints can be set at the following events:

- OnPreExecute
- OnPostExecute
- OnError
- OnWarning
- OnInformation
- OnTaskFailed
- OnProgress
- OnQueryCancel
- OnVariableValueChanged
- OnCustomEvent

Adding Breakpoints

Right-click the task to set the breakpoints. Then select **Edit breakpoints** to select the events (or press F9, which quickly adds a Pre-execute breakpoint on the selected object). When the package is run, it will break at the selected event, as shown in Figure 40.

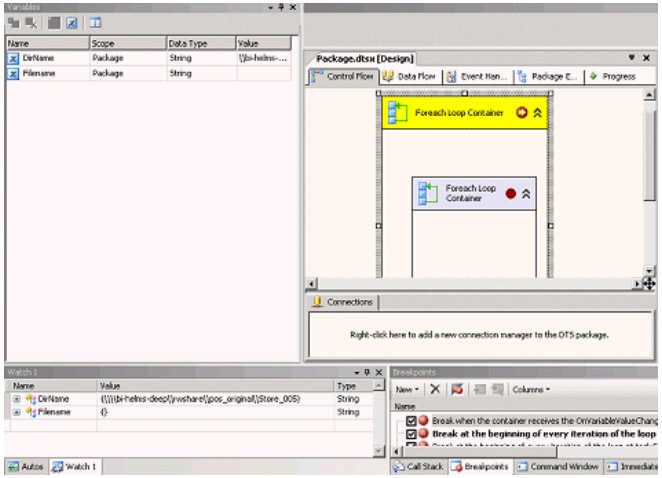

Figure 40

You should also enable logging as explained in Adding Logging to Your Package earlier in this paper to record the events on package execution.

The DTExec utility can be used to execute the package from the command line. The DTExec utility provides access to all package configuration and execution features, such as connections, properties, variables, logging, and progress indicators. DTExec also provides the ability to load packages from three sources: a Microsoft SQL Server database, the DTS service, and the file system.

Analysis Services Partition Cloning

The SQL Server 2000 DTS packages that we received for Phase 1 had code that connected to SQL Server and determined whether, based on the date of the data being added, a new monthly partition needed to be created in the Analysis Services cubes. The queries were issued to SQL Server as part of an ActiveX Scripting task using SQL-DMO. Then, if a new monthly partition needed to be created, Decision Support Objects (DSO) code was run to make a corresponding partition in Analysis Services. The DSO code made a "clone" of an existing partition and gave it a name, setting the slice value based on the new year and month of the data being loaded.

Moving this to SSIS required rethinking how to perform equivalent tasks. Some questions that needed answers included:

- Leave it as is in SQL-DMO and DSO, or upgrade it to use a .NET Scripting task?
- Use .NET Scripting or use native tasks in SSIS?
- Use the existing programming models, or the SQL Server 2005 equivalents? (SQL Server Management Objects (SMO) / Analysis Services Management Objects (AMO))

We made several decisions very quickly. The first was to use a .NET Scripting task. The second was to upgrade the code from the ActiveX Scripting task. The first part involved code that called SQL-DMO to determine whether a new partition should be created. Since the management object model shouldn't be used for query access, we rewrote the data access code using ADO.NET. This was very easy since we were able to simply cut and paste a code sample from the Microsoft Developer Network (MSDN) to get things running very quickly. The code looks something like this:

```
Dim sPartition As String Dim conn As SqlConnection = New SqlConnection("Data Source=servername;" & "Integrated Security=SSPI;Initial Catalog=DW1") Dim partCMD As SqlCommand = conn.CreateCommand() partCMD.CommandText =
"select partitionname from partitions where status = 'N'" conn.Open() Dim myReader As SqlDataReader = partCMD.ExecuteReader() myReader.Read() 'if this is not a new month, exit out If myReader.HasRows = False Then
'update DTS global variable Dts.VariableDispenser.LockOneForWrite("ItemSalesXML", vars) vars.Item("ItemSalesXML").Value = " " Exit Sub 'it is a new month, set the partition variable value Else sPartition = myReader
("partitionname").ToString() End If
```

The next decision for us was whether to upgrade the DSO code to AMO, or to use another option. We chose another way. We used the graphical interface in SQL Server Management Studio to script out a partition in the Analysis Services database (in the XML for Analysis scripting language) in SQL Server Management Studio, and then dynamically modify the partition name in our .NET Scripting task. The .NET Scripting task then assigned the newly formed XML for Analysis string to an SSIS global variable. We then had a follow-on task (the new Analysis Services Execute DDL task) that ran our XML for Analysis code as defined in the SSIS global variable.

Another option here would have been to write the updated code in the AMO programming model. However, this approach takes more learning time. The XML for Analysis approach, while technically less robust, was faster to implement. Perhaps not a best practice, but definitely a truism—when under a crunch developers often take a path of least resistance.

Extend the DTS Data Flow Tasks Using Custom Source and Transformation Components

As described in Core ETL Processing, the sales data to be loaded comes from TLOG data generated by the customer's point-of-sales (POS) cash registers.

Migrating the package to SSIS required rethinking their design for loading the TLOG data into the database. We were unsure whether to continue using Perl scripts to parse the data or to convert their parser into custom data flow components that would parse the data within the pipeline. The latter approach would perform better and offer streamlined, efficient ETL. The big attraction was that, by doing so, we could avoid the costly step of copying data from the TLOG files into flat files and then into the database. This approach is also more manageable. Hence, it was decided that we would implement a custom task to load the TLOG data.

The TLOG files contained compressed packed decimal data, where each row could either be delimited by a new line or a comma. Hence, we could not simply use the available data flow sources to read the data files. Before the data could be parsed, it was necessary to first unpack the packed decimal data. For simplicity and manageability, it was decided to separate the logic of unpacking the binary file from that of parsing the data based on the template file. Hence, we ended up implementing two custom

pipeline components:

- A custom source component to read the data.
- A custom transformation component to parse the data.

Both the components derive from the **PipelineComponent** base class and override the appropriate methods. The source component connects to an external TLOG file using the File connection. It accepts a template file as the input. Based on the template, the source component unpacks the packed decimal bytes into appropriate Hexadecimal and ASCII characters. A sample template format looks like:

```
"02" => ["H2","H10","H6","H2"],

"99-Data" => ["H2","A4","A99"]
```

The source component unpacks the data based on the template and outputs a row of single column of type **DT_TEXT**.

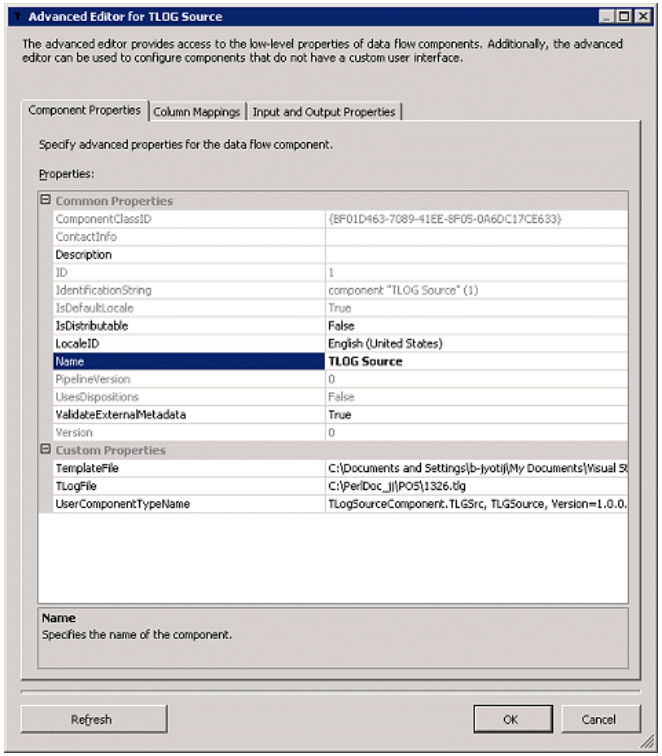

Figure 41

Figure 41 shows the component properties of the source component.

The custom component reads the unpacked data from the upstream data flow, parses the data based on the .ini file, and generates multiple outputs downstream. The custom transformation accepts two inputs:

Config File: path of the .ini file

Store Name: store number of the TLOG file to be processed

A sample part of the .ini file is of this format:

```
[01] Filename="Item" DelimiterCharacter="," OutputFields="%store,%line,1-0,%term,%tran,%datetime,1-1,1-2,1-3,1-4
```

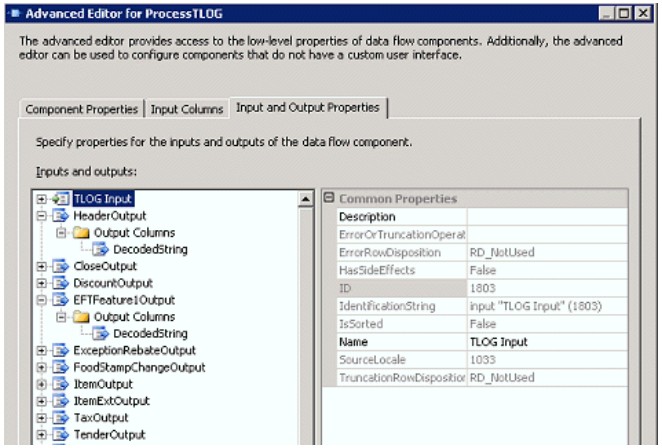

Figure 42

The custom transformation generates multiple output of type DT_STR as shown in Figure 42 based on the set of rules defined in the .ini file.

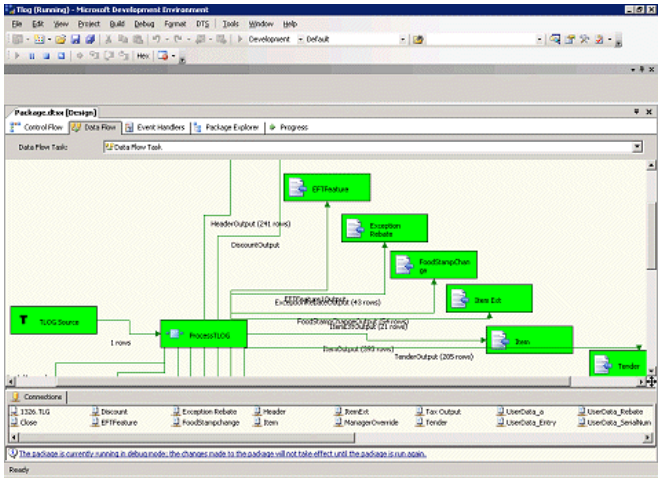

Figure 43

Figure 43 shows a package that incorporates the TLOG source and transformation component.

More details on the design of the custom dataflow components, along with sample code, will be provided in a separate follow-up white paper.

Advanced Editors

When you edit a package, you typically double-click a task or a data flow item to edit the properties of that item. However, for some objects there are two editors—a default editor and an advanced editor. When an advanced editor exists, it usually provides a significantly enhanced set of properties that you can change for a given object. You should explore the advanced editor for each kind of object you are working with in SSIS to determine when to use the advanced editor versus the default editor. To reach the advanced editor, right-click the task, source, and so on, and select the **Show Advanced Editor** option (as shown in Figure 44).

Exercise caution when working with the advanced editors, just as you would have exercised caution when working in Disconnected Edit mode in SQL Server 2000 DTS. You have direct access to a number of properties that, if incorrectly set, could have serious negative consequences on your package. However, sometimes this is the best way to correct issues in your package (see the section titled Changes to a file connection object aren't always picked up by a flat file source later in this paper for an example).

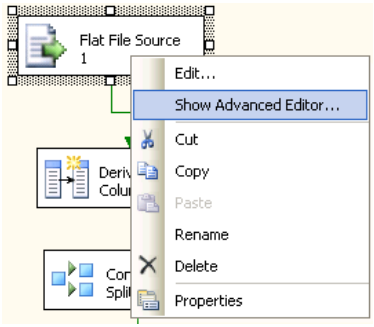

Figure 44

Performance Tips and Tricks

- If the package is running on the same server as your destination SQL Server 2005 relational database instance, use the SQL Server Destination component instead of the OLE DB connection for SQL Server. This component runs in-process and thus avoids the overhead of a connection. It can be up to 25 percent faster than the OLE DB connection.
- Remove unnecessary columns from the data flow to improve performance. The data flow engine warns the user about the output columns not used. Removing these columns saves the engine from allocating space and processing data that is not used.
- The **EngineThreads** property on a task sets the number of threads used by the task. The default value is **5** but on a multi-processor server, this value can be set at a higher value to improve performance. The number of threads for optimal performance can be determined on the basis of testing.

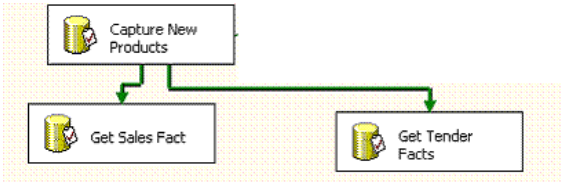

Figure 45

- On a multi-processor server, execute independent tasks in parallel. For example, for a Data Warehouse project, once the dimension tables in Data Warehouse are loaded, if the fact tables access different dimension tables and can be loaded simultaneously, you might consider loading the data in parallel as shown in Figure 45.
- Though the correct number of tasks executed in parallel for optimal performance can be determined only after testing, you can start by setting the number of parallel tasks equal to the number of processors.

Issues Encountered

As is to be expected in a product that is still under development, we discovered a number of technical issues. If you are using the community preview release (IDW9) of SQL Server 2005, you may encounter some of these issues, as well. Our expectation is that these issues will have been resolved for the Beta 3 release of SQL Server 2005.

SQL Server Destination Doesn't Work Against Named Instances

When creating a data flow task, the most efficient way to load an SQL Server table is to use the SQL Server destination. This destination is a highly optimized in-memory connection to an SQL Server instance, and whenever possible should be used as the preferred connection to your SQL Server instances (it can be up to 25 percent faster than the method shown in Figure 46, which uses an OLE DB fast load destination to SQL Server).

The downside of this destination is that it requires that the SQL Server instance be on the same server as the SSIS package. This includes at development time, so you must develop on a computer with an installation of SQL Server to target to add this destination type to your data flow. If you must develop on another server, you should use the OLE DB destination, point to an instance of SQL Server, and make sure to set the **fast load** option in the **Data access mode:** list as shown in Figure 46**.**

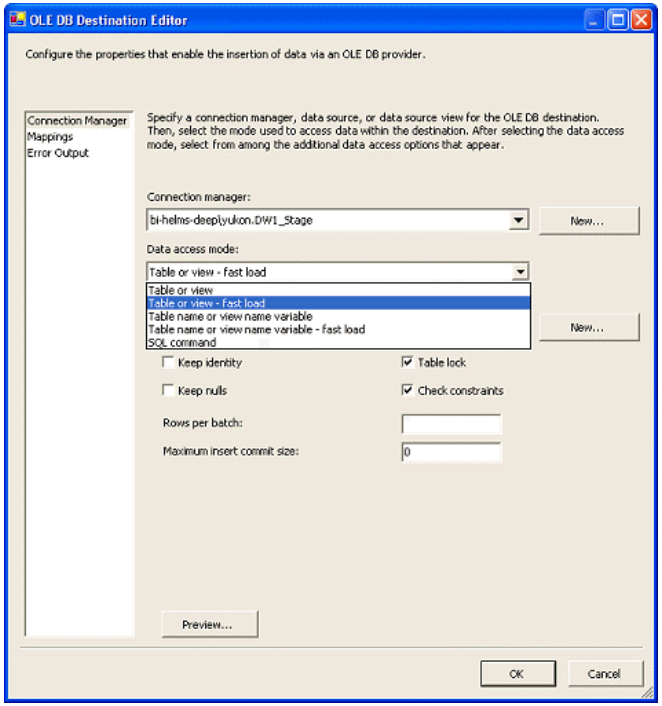

Figure 46

Unfortunately, in the IDW9 community preview release (the release used for development of this project) the SQL Server connection wasn't working against named instances (bug #323570). This issue will certainly be fixed by the next public update of SQL Server 2005.

Changes to a File Connection Object Aren't Always Picked Up by a Flat File Source

When you create a connection to a file using the Flat File connection manager, a common thing to do is to define columns and their layouts. This information gets picked up when you use that connection as a flat file source in a data flow. However, when you make changes to the column layout, it doesn't necessarily get picked up by the flat file source. To implement such changes, you must edit the flat file source in the data flow and update such column definitions manually. We saw this particularly when we attempted to update the length property of a column. Additionally, we had to use the **Advanced Edit** property on the flat file source to make the change. Double-clicking the flat file source brings up the default editor as illustrated in Figure 47.

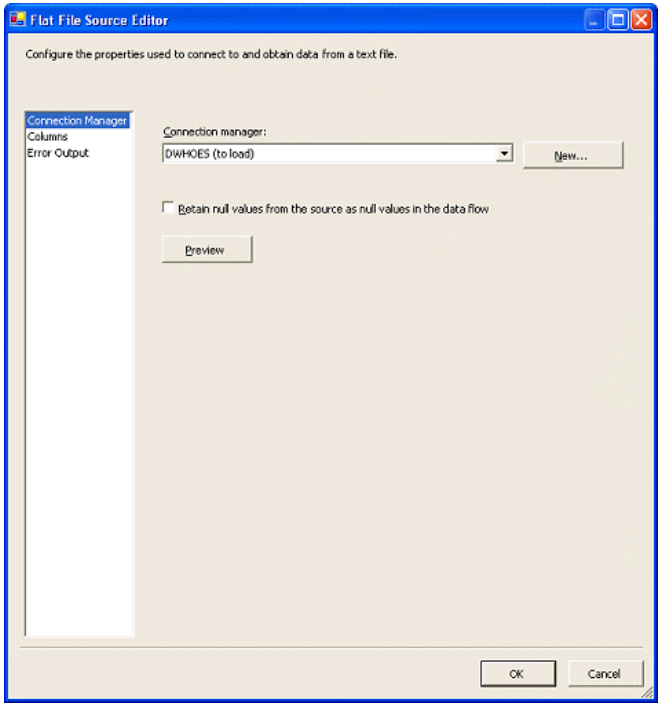

Figure 47

If you right-click the flat file source, you see an option for **Show Advanced Editor**.... Select this to bring up the advanced editor for the flat file source. This allows you to configure individual column properties to make the necessary changes not visible from the default editor. For example, you are able to manage the mapping of columns and various detailed properties for each of the columns, as illustrated in Figure 48.

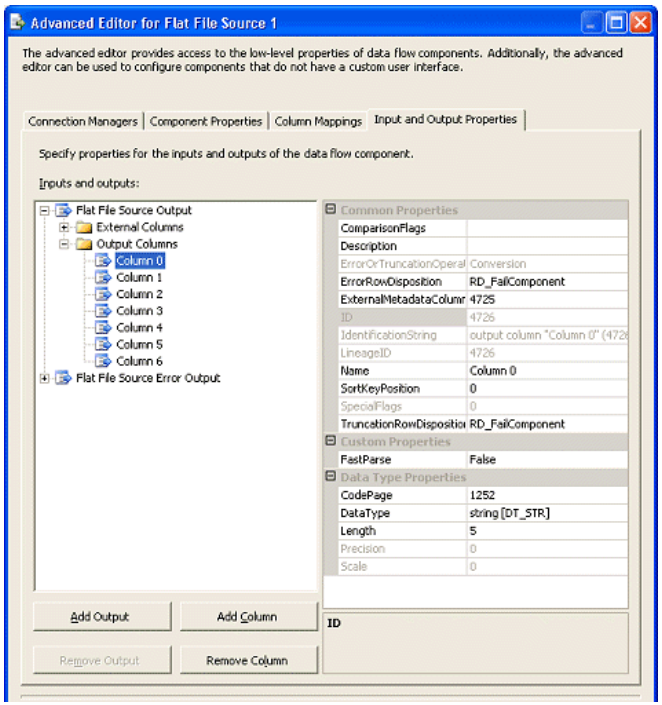

Figure 48

Minimizing the BI Development Studio While Debugging an SSIS Package May Cause the GUI to Disappear

The graphical interface in BI Development Studio disappeared on us several times during debugging. There is no consistent way to reproduce this, but the effects are so dramatic it's worth pointing out. Sometimes while debugging a package, when an attempt was made to minimize the debugger, the development environment simply disappeared. It was not visible on the Windows taskbar that is typically at the bottom of the screen and no visible GUI was shown—it simply seemed to disappear. However, when we checked Task Manager, BIENV.EXE (the executable for BI Developer Studio) was still listed as running on the system. The solution was to ALT-TAB to the designer (it is still in the ALT-TAB list even though it's no longer in the Windows taskbar). This returns the focus to within the development environment and also restores the development environment to the Windows taskbar.

Product Enhancement Requests

Several new product enhancement requests were generated during Phase 1 work on SSIS. These are enumerated in this section.

The "Partial Row" Problem While Loading a Flat File

During attempts to perform file loads, we encountered an error that had a "partial row" in it. The "partial row" was not visible while browsing the data in Notepad. When troubleshooting this, we found a handy feature in Visual Studio we'll also point out. When opening a file (click **File**, then **Open**, then **File**), it's possible to click on a down-arrow that appears next to the **Open** button as illustrated in Figure 49.

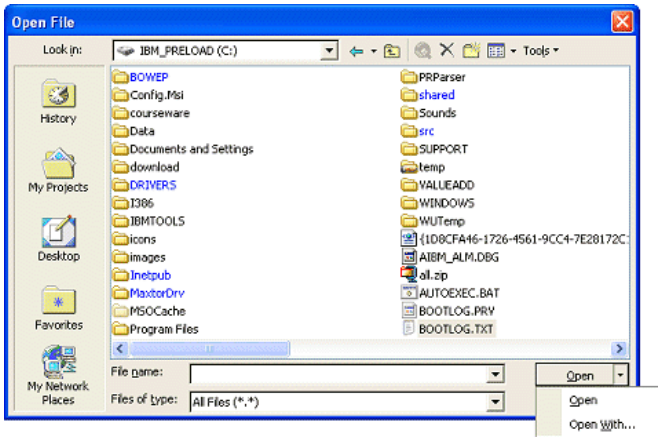

Figure 49

Selecting the **Open With** option enables you to select a Binary Editor, as illustrated in Figure 50.

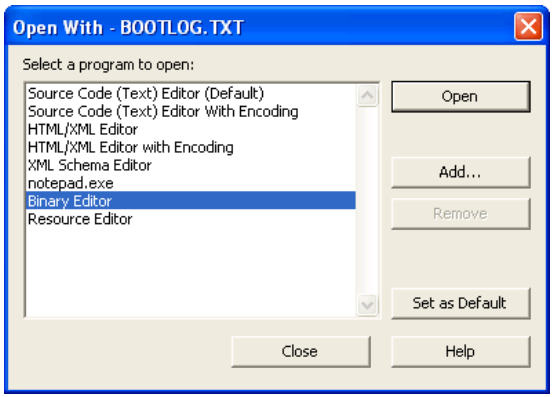

Figure 50

Notice the hex "1A" that appears at the end of this file (Figure 51). This is the end-of-file marker in Windows and, at least in Windows Server 2003, an ordinary copy command inserts this value at the end of the file. At the time of this writing, SSIS sees this as a new row in the data file, but treats it as an incomplete row. Unfortunately, SSIS stops reading the file in this situation and sets the task state to failed.

```
012af960  53 54 55 56 57 58 59 5A  41 42 43 44 45 46 47 48  STUVWXYZABCDEFGH
012af970  49 4A 4B 4C 4D 4E 4F 50  51 52 53 54 55 56 57 58  IJKLMNOPQRSTUVWX
012af980  59 5A 41 42 43 44 45 46  47 48 49 4A 4B 4C 4D 4E  YZABCDEFGHIJKLMN
012af990  4F 50 51 52 53 54 55 56  57 58 59 5A 41 42 43 44  OPQRSTUVWXYZABCD
012af9a0  45 46 47 48 49 4A 4B 4C  4D 4E 4F 50 51 52 53 54  EFGHIJKLMNOPQRST
012af9b0  55 56 57 58 59 5A 0D 1A                           UVWXYZ..
```

Figure 51

A change request was filed (#323638) to at least have an option to allow this end of file marker be ignored. The workaround for this project is to change the copy command that is being executed to include the **/B** option. When this option is added to the copy command, the file type is set to binary and the file is not modified.

The Derived Column Editor May Change Data Types When Modifying Column Definitions

When creating a Derived Column transformation, the need arose to change a column definition. When we made simple modifications, the data type that was previously designed as **String (DT_STR)** was changed to **Unicode String (DT_WSTR)**. The change we had made was to a substring statement on an existing column that was used to define a new column. The change was simple in that we modified it from **substring(optional, 1, 8)** to **substring(optional, 1, 9)**. One would not anticipate that such a small change would necessarily cause our data type for the target derived column to be reset. As of the time of this writing, this is being considered as "by design," because technically it's a modification to the target column that causes the target data type to be reset to the default definition. Unless addressed, this is definitely something to watch out for.

Flat File Connections Can't Be Used as a Source for the BULK INSERT Task

When opening the file to perform filtering of records, we used a "Flat File Connection Manager" connection. That connection enabled us to define the column layout and file location. After filtering our text file, we saved the file and subsequently attempted to reuse the same connection to feed a BULK INSERT task to load our file. We discovered the BULK INSERT task only accepts connections from the "File Connection Manager" (note the subtle but critical difference in naming). To perform the BULK INSERT operation, we had to define a redundant connection to the file using the "File Connection Manager." We filed a design change request asking that the BULK INSERT task be enabled to source directly from a "Flat File Connection Manager" object as well. The product should allow Flat File Connection Manager objects to be used more widely in Beta 3.

Wildcards Can't Be Specified in the File System Task

Some operations that were required for Project REAL required copying and moving files while looping through the directory structure. An example was copying multiple files into a single text file, or moving files from a "to process" to a "processed" directory once they had been loaded. In SQL Server 2000 DTS these tasks are typically done using an Execute Process task that makes calls to batch files. We hoped to be able to use the File System task to do this rather than to continue to depend on the batch files.

Unfortunately, the File System task works on all files in a directory or in a set of directories. Further complicating things, most of our directories had multiple types of files within them and we wanted to work against a subset of the files (for example, just *.txt files). The file system task wasn't capable of filtering by file extension, so we ended up using an Execute Process task. A design change request was filed to add the capability to use extensions for filtering.

Analysis Services Processing Task Progress Information Is not Adequate

The Analysis Services Processing task, when processing a complex cube, puts out many progress messages with a % complete message in the **Progress** tab. Unfortunately, nowhere in the message will you find what particular object against which the progress is being reported. That makes it frustrating if you trying to gauge the progress of your processing. A design change request has been filed to add more detail to this logging. As of the writing of this paper, no decision has been made on our request.

Conclusion

As is evident from our discussion, many differences exist between SQL Server 2000 DTS and the new Integration Services in SQL Server 2005. While Microsoft put significant effort into enabling customers to continue running their existing SQL Server 2000 DTS packages, it should be obvious from this brief tour that a clean, easy migration path is not always going to be readily achieved. Given the dramatic architectural changes and enhancements introduced in SQL Server 2005 Integration Services, the key to successfully migrating existing DTS 2000 packages rests on knowing which tasks can be migrated versus which need to be rewritten. Once you assess each package that you intend to migrate, you should plan enough time to either rewrite those tasks that can't be migrated or else plan to run them under the DTS 2000 Runtime engine until you can rewrite them.

Also, Integration Services in SQL Server 2005 introduces many new concepts. As such, it represents a paradigm shift that existing DTS 2000 programmers will need to embrace in order to be effective. It's definitely worth taking some time to go through the supplied samples in order to gain familiarity with core new features of the product. Developers who are familiar with Visual Studio (VS) languages such as Visual Basic, Visual C#, and others will likely find the new SQL Server 2005 BI Development Studio friendly and familiar. Those unfamiliar with VS might benefit from some Visual Studio training or by working through some of the VS online tutorials.

For more information:

To learn more about SQL Server 2005, including Integration Services, visit the Microsoft SQL Server 2005 home page.

Manage Your Profile |Legal |Contact Us |MSDN Flash Newsletter

© 2005 Microsoft Corporation. All rights reserved. Terms of Use |Trademarks |
Privacy Statement