HANDS-ON Microsoft® SQL Server® 2008 INTEGRATION SERVICES

Second Edition

ASHWANI NANDA

HANDS-ON

Microsoft® SQL Server® 2008

INTEGRATION SERVICES

Ashwani Nanda

McGraw Hill

New York Chicago San Francisco Lisbon
London Madrid Mexico City Milan
New Delhi San Juan Seoul Singapore
Sydney Toronto

The McGraw-Hill Companies

Copyright © 2011 by The McGraw-Hill Companies. All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

ISBN: 978-0-07-173641-1

MHID: 0-07-173641-7

The material in this eBook also appears in the print version of this title: ISBN: 978-0-07-173640-4,

MHID: 0-07-173640-9.

All trademarks are trademarks of their respective owners. Rather than put a trademark symbol after every occurrence of a trademarked name, we use names in an editorial fashion only, and to the benefit of the trademark owner, with no intention of infringement of the trademark. Where such designations appear in this book, they have been printed with initial caps.

McGraw-Hill eBooks are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. To contact a representative please e-mail us at bulksales@mcgraw-hill.com.

Information has been obtained by McGraw-Hill from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, McGraw-Hill, or others, McGraw-Hill does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

TERMS OF USE

This is a copyrighted work and The McGraw-Hill Companies, Inc. (“McGrawHill”) and its licensors reserve all rights in and to the work. Use of this work is subject to these terms. Except as permitted under the Copyright Act of 1976 and the right to store and retrieve one copy of the work, you may not decompile, disassemble, reverse engineer, reproduce, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish or sublicense the work or any part of it without McGraw-Hill’s prior consent. You may use the work for your own noncommercial and personal use; any other use of the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

THE WORK IS PROVIDED “AS IS.” McGRAW-HILL AND ITS LICENSORS MAKE NO GUARANTEES OR WARRANTIES AS TO THE ACCURACY, ADEQUACY OR COMPLETENESS OF OR RESULTS TO BE OBTAINED FROM USING THE WORK, INCLUDING ANY INFORMATION THAT CAN BE ACCESSED THROUGH THE WORK VIA HYPERLINK OR OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. McGraw-Hill and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither McGraw-Hill nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. McGraw-Hill has no responsibility for the content of any information accessed through the work. Under no circumstances shall McGraw-Hill and/or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

To my kids—With practice you can perform impossible tasks easily.

About the Author

Ashwani Nanda develops and maintains marketing, analytical, and CRM-based database systems for medium- to large-sized enterprises. His current area of specialization is data modeling, database development, business intelligence reporting, and analysis. He started his career on real-time mainframe systems back in the age when hard disks used to weigh a kilo per megabyte. Moving from system engineering for mainframes to networking and infrastructure support, then to database management and currently to business intelligence, he has worked for several major hardware and software vendors. He loves to integrate disparate technologies utilizing the best features of available applications. These days he is using his business intelligence skills to create data visualization reports. He can be reached at ananda@AffordingIT.co.uk.

About the Technical Editor

Allan Mitchell started his love affair with ETL in SQL Server 7 and Data Transformation Services. He moved on to SSIS with SQL Server 2005 and has traveled the world working with clients big and small who have complex data integration issues. He lives in the UK and is married to Lesley. They have a wonderful son, Ewan.

Contents at a Glance

Chapter 1	**Introducing SQL Server Integration Services**	**1**
Chapter 2	**Getting Started with Wizards**	**41**
Chapter 3	**Nuts and Bolts of the SSIS Workflow**	**69**
Chapter 4	**Integration Services Control Flow Containers**	**109**
Chapter 5	**Integration Services Control Flow Tasks**	**135**
Chapter 6	**Administering Integration Services**	**225**
Chapter 7	**Securing Integration Services Packages**	**269**
Chapter 8	**Advanced Features of Integration Services**	**291**
Chapter 9	**Data Flow Components**	**333**
Chapter 10	**Data Flow Transformations**	**377**
Chapter 11	**Programming Integration Services**	**481**
Chapter 12	**Data Warehousing and SQL Server 2008 Enhancements**	**535**
Chapter 13	**Deploying Integration Services Packages**	**567**
Chapter 14	**Migrating to Integration Services 2008**	**595**
Chapter 15	**Troubleshooting and Performance Enhancements**	**631**
Appendix	**How to Use the Provided Software**	**673**
	Index	**677**

This page intentionally left blank

Contents

Acknowledgments xix
Introduction xxi

Chapter 1 **Introducing SQL Server Integration Services 1**

Integration Services: Features and Uses 4
Integration Services Architecture 4
Integration Services Designer and Management Tools 4
Data Warehousing Loading 5
Standardizing and Enhancing Data Quality Features 6
Converting Data into Meaningful Information 7
Data Consolidation 7
Package Security Features 7
Service-Oriented Architecture 8
SSIS Package as a Data Source 8
Programmability 8
Scripting 9
Easy Management of SSIS Packages 9
Automating Administrative Tasks 10
Easy Deployment Features 10
Legacy Support Features 10
What's New in Integration Services 2008 10
Better Lookup 11
Improved ADO NET Components 12
Powerful Scripting 12
Extended Import and Export Wizard 12
Ability to Profile Your Data 12
Optimized Thread Allocation 13
SSIS Package Upgrade Wizard 13
Taking Advantage of Change Data Capture 13
Benefiting from T-SQL Merge Statement 13
Enhanced Debugging 14
Inclusion of New Date and Time Data Types 14
Where Is DTS in SQL Server 2008? 14

Integration Services in SQL Server 2008 Editions 15
32-Bit Editions vs. 64-Bit Editions 17
Integration Services Architecture 18
Integration Services Service 19
Integration Services Object Model 20
Integration Services Run Time 20
Integration Services Data Flow 20
Installing Integration Services 21
Installing Integration Services on a Clean System 21
Hands-On: Installing SQL Server 2008 Integration Services 22
Installing Integration Services from the Command Prompt 25
Installing Side by Side 26
Upgrading to SQL Server 2008 Integration Services 27
Business Intelligence Development Studio 28
Hands-On: Creating a Blank Integration Services Project 29
SQL Server Management Studio 37
Hands-On: Connecting to Integration Services Service 37
Summary 39

Chapter 2 **Getting Started with Wizards 41**

Starting SQL Server Import and Export Wizard 43
Hands-On: Importing a Flat File into SQL Server 2008 43
Using Business Intelligence Development Studio 53
Hands-On: Exploring an SQL Server Import and Export Wizard Package Using BIDS 53
Integration Services Connections Project Wizard 63
Analyzing Data Quality with the Data Profiling Task 63
Single-Column Profiles 64
Multiple-Column Profiles 65
Hands-On: Using Data Profiling Task 66
Summary 68

Chapter 3 **Nuts and Bolts of the SSIS Workflow 69**

Integration Services Objects 70
Solutions and Projects 71
File Formats 73
Fixed Width 73
Delimited 73
Ragged Right 74

Connection Managers 74
ADO Connection Manager 75
ADO.NET Connection Manager 75
Cache Connection Manager 76
Excel Connection Manager 76
File Connection Manager 77
Flat File Connection Manager 78
FTP Connection Manager 78
HTTP Connection Manager 78
MSMQ Connection Manager 78
Analysis Services Connection Manager 78
Multiple Files Connection Manager 79
Multiple Flat Files Connection Manager 79
ODBC Connection Manager 80
OLE DB Connection Manager 80
SMO Connection Manager 80
SMTP Connection Manager 81
SQL Server Compact Edition Connection Manager 81
WMI Connection Manager 81
Microsoft Connector 1.0 for SAP BI 81
Microsoft Connector for Oracle by Attunity 82
Microsoft Connector for Teradata by Attunity 83
Data Sources and Data Source Views 84
Data Sources 84
Data Source View 85
SSIS Variables 86
System Variables 87
Hands-On: Using System Variables to Create Custom Logs 87
User-Defined Variables 93
Hands-On: Creating a Directory with User-Defined Variables 95
Precedence Constraints 99
Constraint Options 100
Multiple Constraints 102
Integration Services Expressions 102
Hands-On: Using Expressions to Update Properties at Run Time 105
Summary 107

Chapter 4 **Integration Services Control Flow Containers 109**

Integration Services Package 110
Foreach Loop Container 111
Hands-On: Contacting Opportunities 113
For Loop Container 123
Hands-On: Deleting Data Month by Month After Archiving 124
Sequence Container 131
Task Host Container 133
Summary 133

Chapter 5 **Integration Services Control Flow Tasks 135**

Categories of Control Flow Tasks 136
Data Flow Task 137
Data Preparation Tasks 137
Workflow Tasks 138
SQL Server Tasks 138
Scripting Tasks 139
Analysis Services Tasks 139
Transfer Tasks 139
Maintenance Tasks 140
Backward Compatibility Tasks 141
Custom Tasks 141
Control Flow Tasks in Detail 142
FTP Task 142
Preparations for the Hands-On Exercises in This Chapter 143
Hands-On: Downloading Zipped Files 144
Execute Process Task 148
Hands-On: Expanding Downloaded Files 149
File System Task 155
Hands-On: Archiving Downloaded Files 156
Web Service Task 160
XML Task 161
Input Section 162
Second Operand Section 162
Output Section 162
Operation Options Section 162
Execute SQL Task 165
General Page 166

Parameter Mapping Page 168
Result Set Page 168
Expressions Page 168
Bulk Insert Task 169
Message Queue Task 169
Hands-On: Importing Expanded Files 173
Execute Package Task 187
Hands-On: Consolidating Workflow Packages 189
Send Mail Task 195
WMI Data Reader Task 196
Hands-On: Reading the Application Log 197
WMI Event Watcher Task 200
Transfer Database Task 202
Transfer Error Messages Task 203
Transfer Jobs Task 205
Transfer Logins Task 206
Transfer Master Stored Procedures Task 208
Transfer SQL Server Objects Task 210
Back Up Database Task 212
Check Database Integrity Task 213
Execute SQL Server Agent Job Task 214
Execute T-SQL Statement Task 215
History Cleanup Task 216
Maintenance Cleanup Task 217
Notify Operator Task 218
Rebuild Index Task 220
Reorganize Index Task 221
Shrink Database Task 222
Update Statistics Task 223
Summary 224

Chapter 6 **Administering Integration Services 225**

Connecting to Integration Services Service 226
Managing Packages with Default Settings 227
Managing Packages Saved on a Remote Server 227
Managing Packages on an Instance of an SQL Server 228
Connecting to Integration Services on a Remote Server 228

Managing SSIS Packages 229
Hands-On: Working with Integration Services Storage Folders 229
dtutil Utility 235
Hands-On: Using dtutil 237
Running SSIS Packages 244
SQL Server Import and Export Wizard 244
BIDS 244
Execute Package Utility (DTExecUI) 245
Hands-On: Running an SSIS Package Using the Execute Package Utility 246
DTExec Utility 250
SQL Server Agent 256
Hands-On: Automating Running an SSIS Package with SQL Server Agent 257
Executing SSIS Packages Programmatically 266
Summary 267

Chapter 7 **Securing Integration Services Packages 269**
Digitally Signing the Package 270
Excluding Sensitive Information from the Package 271
Encrypting Sensitive Information in the Package 272
Encrypting All the Information in the Package 273
Hands-On: Working with Package Protection Levels 273
Using Integration Services Fixed Database-Level Roles 282
Fixed Database-Level Roles and Their Permissions 282
Hands-On: Control Access to a Package with User-Defined Roles 283
Considerations for Different Storage Areas 288
Considerations for Saving to SQL Server 289
Considerations for Saving to the File System 290
Summary 290

Chapter 8 **Advanced Features of Integration Services 291**
Logging and Log Providers in SSIS 292
Hands-On: Configuring Logging in a Package 295
Transactions in Integration Services Packages 300
Hands-On: Maintaining Data Integrity with Transactions 301
Restarting Packages with Checkpoints 311
Hands-On: Restarting a Failed Package Using Checkpoints 313

Expressions and Variables 317
Hands-On: Extending the Contacting Opportunities Package with Property Expressions 318
Handling Events at Package Run Time 323
Hands-On: Creating Event Handlers in an SSIS Package 325
As a Data Source for Reporting Services Report 328
Enable SSIS as a Data Source 328
Using SSIS as a Data Source 330
Summary 332

Chapter 9 **Data Flow Components 333**

From Control Flow to Data Flow 334
Data Flow Component Interfaces 335
External Metadata 336
Inputs 336
Outputs 337
Error Outputs 337
Considerations for Bringing Data into Data Flow 337
Data Flow Sources 342
ADO NET Source 343
Excel Source 345
Flat File Source 346
OLE DB Source 348
Raw File Source 349
Script Component Source 350
XML Source 351
Data Flow Transformations 351
Business Intelligence Transformations 352
Row Transformations 353
Rowset Transformations 354
Split and Join Transformations 354
Auditing Transformations 355
Data Flow Destinations 355
ADO NET Destination 357
Data Mining Model Training Destination 357
DataReader Destination 358

Dimension Processing Destination 358
Excel Destination 360
Flat File Destination 360
OLE DB Destination 361
Partition Processing Destination 363
Raw File Destination 364
Recordset Destination 365
Script Component Destination 365
SQL Server Compact Destination 365
SQL Server Destination 366
Data Flow Paths 367
Hands-On: An Introduction to the Data Flow Task 368
Summary 375

Chapter 10 **Data Flow Transformations 377**

Row Transformations 378
Copy Column Transformation 378
Character Map Transformation 379
Data Conversion Transformation 381
Derived Column Transformation 382
Export Column Transformation 385
Import Column Transformation 386
Script Component 386
OLE DB Command Transformation 387
Split and Join Transformations 389
Conditional Split Transformation 389
Multicast Transformation 390
Union All Transformation 391
Merge Transformation 391
Merge Join Transformation 392
Cache Transform 394
Lookup Transformation 395
Hands-On: Updating PersonContact Data 400
Rowset Transformations 415
Sort Transformation 415
Percentage Sampling Transformation 417
Row Sampling Transformation 418

Pivot Transformation 419
Hands-On: Pivoting Sales Order Records in an Excel Worksheet 420
Unpivot Transformation 427
Aggregate Transformation 429
Hands-On: Aggregating SalesOrders 430
Audit Transformations 436
Audit Transformation 436
Row Count Transformation 437
Business Intelligence Transformations 439
Slowly Changing Dimension Transformation 439
Hands-On: Loading a Slowly Changing Dimension 443
Data Mining Query Transformation 455
Term Lookup Transformation 456
Term Extraction Transformation 457
Fuzzy Grouping Transformation 460
Fuzzy Lookup Transformation 463
Hands-On: Removing Duplicates from Owners Data 468
Summary 480

Chapter 11 **Programming Integration Services 481**

The Two Engines of Integration Services 482
Programming Options 483
Scripting 484
Developing Custom Objects from Scratch 485
Building Packages Programmatically 486
Extending Packages with Scripting 486
The Legacy Scripting Task: ActiveX Script Task 486
Script Task 488
Hands-On: Scripting the Handshake Functionality 488
Script Component 505
Script Task vs. Script Component 506
Hands-On: Extending Data Flow with the Script Component 507
Script Component as a Data Source 507
Script Component as a Transformation 515
Script Component as a Destination 525
Debugging Techniques for Script Component 530
Summary 534

Chapter 12 **Data Warehousing and SQL Server 2008 Enhancements** 535

The Need for a Data Warehouse 536
Data Warehouse Design Approaches 538
Top-Down Design Approach 538
Bottom-Up Design Approach 539
Data Warehouse Architecture 539
Centralized EDW with Dependent Data Marts 539
Distributed Independent Data Marts 540
Data Warehouse Data Models 541
Entity-Relationship Model 541
Dimensional Model 542
Dimension Types 544
Loading a Dimension Using a Slowly Changing Dimension 545
Data Model Schema 547
Star Schema 547
Snowflake Model 548
Building a Star Schema 549
SQL Server 2008 R2 Features and Enhancements 551
SQL Server 2008 R2 Data Warehouse Editions 551
SQL Server 2008 R2 Datacenter 551
SQL Server 2008 R2 Parallel Data Warehouse 552
SQL Server 2008 R2 Data Warehouse Solutions 552
Fast Track Data Warehouse 553
Parallel Data Warehouse 554
SQL Server 2008 R2 Data Warehouse Enhancements 557
Backup Compression 557
MERGE Statement 559
GROUP BY Extensions 561
Star Join Query Processing Enhancement 562
Change Data Capture 562
Partitioned Table Parallelism 564
Summary 566

Chapter 13 **Deploying Integration Services Packages** 567

Package Configurations 568
Types of Package Configurations 569
Hands-On: Applying Configurations to Contacting Opportunities 572

Direct and Indirect Configurations 577
Hands-On: Using Indirect Configurations 578
Deployment Utility 587
Deploying Integration Services Projects 589
Hands-On: Deploying an Integration Services Project 589
Custom Deployment 592
Summary 594

Chapter 14 **Migrating to Integration Services 2008 . . . . . 595**

Upgrade Advisor 596
Hands-On: Analyzing DTS 2000 Packages with SQL Server 2008 Upgrade Advisor 597
Migrating Data Transformation Services Packages 600
Migration Options 600
Installing DTS 2000 Support Components 601
Running DTS 2000 Packages As-Is with Run-Time Support 604
Hands-On: Executing a DTS 2000 Package 605
Embedding DTS 2000 Packages in Integration Services Packages 609
Execute DTS 2000 Package Task 610
Hands-On: Executing Importing Contacts Using the Execute DTS 2000 Package Task . . . 610
Migrating DTS 2000 Packages to Integration Services 614
Package Migration Wizard 617
Hands-On: Migrating Importing Contacts to Integration Services 617
Upgrading Integration Services 2005 622
Same-Server Installation 623
Different Server Installation 625
Upgrading SSIS 2005 Packages 626
Summary 630

Chapter 15 **Troubleshooting and Performance Enhancements 631**

Troubleshooting Integration Services Packages 632
Debugging Features Available by Default 632
Debugging Tools Requiring Configuration 634
Hands-On: Setting Breakpoints to See Variables Values 634
Performance Enhancements 640
It's All About Memory 640
Architecture of the Data Flow 644
Synchronous and Asynchronous Transformations 645

Classifying Data Flow Transformations 647
Optimization Techniques 650
Performance Monitoring Tools 658
Performance Counters 659
SQL Server Profiler 660
Logging 661
Execution Trees 662
Hands-On: Monitoring Log Events in a Pipeline 663
Using Parallel Processing 668
Running Parallel Tasks in the Control Flow 669
Creating Multiple Data Flows 670
Enhancing EngineThreads 670
Summary 670

Appendix **How to Use the Provided Software 673**
Downloaded Software 674
Attaching Campaign Database 675

Index 677

Acknowledgments

I would like to thank my wife Sarita for being supportive and doing the major work of maintaining the household while I was busy with the book revision work. My kids Toozy, Ritzy, and Himnish also deserve acknowledgments here for being co-operative and patient with me. Special thanks go to the readers of the first edition—especially the ones who have taken time to give their valuable feedback. The feedback coming in through various channels has actually inspired me to revise the first edition to this new edition in line with the release of SQL Server 2008 R2. In the process of writing and editing the book, technical editor Allan Mitchell has contributed his ideas and constructive comments that not only have helped to remove some of the mistakes in the compilation but also have greatly improved the content. I would like to thank all my colleagues at Avis-Europe, where I learn most while working with them and for giving their valuable inputs as well. Finally, special thanks to the team at McGraw-Hill for their hard work and support during this project.

This page intentionally left blank

Introduction

Hands-On Microsoft SQL Server 2008 Integration Services is a revised edition of its predecessor, which was based around SQL Server 2005 Integration Services. I have taken the opportunity to enhance the content wherever I felt could benefit readers more. The feedback I have received on the previous edition has been instrumental to these enhancements. Though not many new features have been packed in this release of Integration Services, I think this book has gone steps beyond its previous release. Not only does it contain improved content and lots of relevant examples and exercises, now it has two new chapters to cover topics such as programming and scripting SSIS and data warehouse practices. These chapters enable readers to extend their SSIS packages with programming and scripting and also show how SSIS can be put to use in large-scale data warehouse implementations, thus extending the reach of the tool and the developer.

This book has been targeted to reduce the learning curve of the readers and, hence, has a unique style of presenting the subject matter. Each topic includes a theoretical introduction, but care has been taken not to present too much information at that stage. More details about the tasks and components are then included in the relevant Hands-on exercises that immediately follow the theoretical introduction. Lots of examples have been included to cover most commonly used business scenarios. All the files and code used in the examples have been provided for you to download from McGraw-Hill site. The appendix of this book contains more details on how to download and use the code. And finally, the chapters have been organized in a way better suited to learning than a sectionalized approach.

Chapters 1 and 2 cover the basic concepts, an introduction to Integration Services, new features in this release, and the Import and Export Wizard.

Chapters 3, 4, and 5 walk through connection managers, control flow tasks and containers, and use of variables and Integration Services expressions.

Chapters 6 and 7 take you deep inside of administration of SSIS packages and the security that you can build around packages.

Chapter 8 demonstrates for you the advanced features of Integration Services.

Chapters 9 and 10 enable you to work with the pipeline components and data flow paths and viewers. These are probably the biggest chapters in the book, as they cover the most important topics, such as performing in-memory lookup operations, standardizing data, removing various types of duplicates, pivoting and un-pivoting data rows, loading a warehouse using SCD transformation, and working with multiple aggregations.

Chapter 11 shows you Integration Services architecture and introduces you to programming and scripting concepts.

Chapter 12 is where the data warehouse and business intelligence features are covered. You are introduced to data warehousing concepts involving star schemas and snowflake structures. This chapter also introduces you to Microsoft's appliance-based data warehouses—the Fast Track Data Warehouse and the Parallel Data Warehouse. Finally, you are introduced to the features built into the SQL Server engine such as data compression, the MERGE statement, Change Data Capture, and Partitioned Table Parallelism that you can use to complement SSIS in the development of ETL processes.

Chapter 13 takes you through the deployment processes.

Chapter 14 helps you in migration of your Data Transformation Services 2000 and Integration Services 2005 packages to the Integration Services 2008 platform.

Chapter 15 is the last chapter, but it covers the very important subject of troubleshooting and performance enhancements.

Chapter 1

Introducing SQL Server Integration Services

In This Chapter

- Integration Services: Features and Uses
- What's New in Integration Services 2008
- Where Is DTS in SQL Server 2008?
- Integration Services in SQL Server 2008 Editions
- Integration Services Architecture
- Installing Integration Services
- Business Intelligence Development Studio
- SQL Server Management Studio
- Summary

Now that the SQL Server 2008 R2 is coming over the horizon packed with self-service business intelligence features, Integration Services not only remains the core platform for the data integration and data transformation solutions but has come out stronger with several product enhancements. With more and more businesses adopting Integration Services as their preferred data movement and data transformation application, it has proven its ability to work on disparate data systems; apply complex business rules; handle large quantities of data; and enable organizations to easily comply with data profiling, auditing, and logging requirements.

The current credit crunch has left businesses in a grave situation with reduced budgets and staff yet with the utmost need to find new customers and be able to close sales. Business managers use complex analytical reports to draw up long-term and short-term policies. The analytical reports are driven by the data collected and harvested by corporate transactional systems such as customer support systems (CRM), call centers and telemarketing operations, and pre- and post-sales systems. This is primarily due to the data explosion because of the increased use of the web. People now spend more time on the web to compare and decide about the products they want to buy. Efforts to study buyer behavior and to profile activities of visitors on the site have also increased data collection. Data about customers and prospects has become the lifeblood of organizations, and it is vital that meaningful information hidden in the data be explored for businesses to stay healthy and grow.

However, many challenges remain to be met before an organization can compile meaningful information. In a typical corporation, data resides at geographically different locations in disparate data storage systems—such as DB2, Oracle, or SQL Server—and in different formats. It is the job of the information analyst to collect data and apply business rules to transform raw data into meaningful information to help the business make well-informed decisions. For example, you may decide to consolidate your customer data, complete with orders-placed and products-owned information, into your new SAP system, for which you may have to collect data from SQL Server–based customer relationship management (CRM) systems, product details from your legacy mainframe system, order details from an IBM DB2 database, and dealer information from an Oracle database. You will have to collect data from all these data sources, remove duplication in data, and standardize and cleanse data before loading it into your new customer database system. These tasks of *e*xtracting data from disparate data sources, *t*ransforming the extracted data, and then *l*oading the transformed data are commonly done with tools called *ETL* tools.

Another challenge resulting from the increased use of the Internet is that "the required information" must be available at all times. Customers do not want to wait. With more and more businesses expanding into global markets, collecting data from multiple locations and loading it after transformation into the diverse data stores with

little or no downtime have increased work pressure on the information analyst, who needs better tools to perform the job.

The conventional ETL tools are designed around batch processes that run during off-peak hours. Usually, the data-uploading process in a data warehouse is a daily update process that runs for most of the night. This is because of the underlying design of traditional ETL tools, as they tend to stage the data during the upload process. With diverse data sources and more complex transformations and manipulations, such as text mining and fuzzy matching, the traditional ETL tools tend to stage the data even more. The more these tools stage data, the more disk operations are involved, and hence the longer the update process takes to finish. These delays in the entire process of integrating data are unacceptable to modern businesses. Emerging business needs require that the long-running, offline types of batch processes be redesigned into faster, on-demand types that fit into shorter timeframes. This requirement is beyond the traditional ETL tools regime and is exactly what Microsoft SQL Server 2008 Integration Services (SSIS) is designed to do.

Microsoft SQL Server Integration Services (also referred as SSIS in this book) is designed keeping in mind the emerging needs of businesses. Microsoft SQL Server 2008 Integration Services is an enterprise data transformation and data integration solution that can be used to extract, transform, and consolidate data from disparate sources and move it to single or multiple destinations. Microsoft SQL Server 2008 Integration Services provides a complete set of tools, services, and application programming interfaces (APIs) to build complex yet robust and high-performing solutions.

SSIS is built to handle all the workflow tasks and data transformations in a way that provides the best possible performance. SSIS has two different engines for managing workflow and data transformations, both optimized to perform the nature of work they must handle. The *data flow engine,* which is responsible for all data-related transformations, is built on a buffer-oriented architecture. With this architecture design, SSIS loads row sets of data in memory buffers and can perform in-memory operations on the loaded row sets for complex transformations, thus avoiding staging of data to disks. This ability enables SSIS to extend traditional ETL functionality to meet the stringent business requirements of information integration. The *run-time engine,* on the other hand, provides environmental support in executing and controlling the workflow of an SSIS package at run time. It enables SSIS to store packages into the file system or in the MSDB database in SQL Server with the ability to migrate the package between different stores. The run-time engine also provides support for easy deployment of your packages.

There are many features in Integration Services that will be discussed in detail in the relevant places throughout this book; however, to provide a basic understanding of how SSIS provides business benefits, the following is a brief discussion on the features and their uses.

Integration Services: Features and Uses

In order to understand how Integration Services can benefit you, let us sift through some of the features and uses that it can be put to. Integration Services provides rich set of tools, self-configurable components, and APIs that you can use to draw out meaningful information from the raw data, create complex data manipulation and business applications.

Integration Services Architecture

The Integration Services Architecture separates the operations-oriented workflow from the data transformation pipeline by providing two distinct engines. The Integration Services run-time engine provides run-time services such as establishing connections to various data sources, managing variables, handling transactions, debugging, logging, and event handling. The Integration Services data flow engine can use multiple data flow sources to extract data, none or many data flow transformations to transform the extracted data in the pipeline, and one or many data flow destinations to load the transformed data into disparate data stores. The data flow engine uses buffer-oriented architecture, which enables SSIS to transform and manipulate data within the memory. Because of this, the data flow engine is optimized to avoid staging data to disk and hence can achieve very high levels of data processing in a short time span. The run-time engine provides operational support and resources to data flow at run time, whereas the data flow engine enables you to create fast, easy-to-maintain, extensible, and reliable data transformation applications. Both engines, though separate, work together to provide high levels of performance with better control over package execution. You will study control flow in Chapters 3 to 5 and data flow components in Chapter 9 and Chapter 10.

Integration Services Designer and Management Tools

SQL Server 2008 provides Business Intelligence Development Studio (BIDS) as the development tool for developing and SQL Server Management Studio for managing Integration Services packages. BIDS includes SQL Server Integration Services Designer, a graphical tool built upon Microsoft Visual Studio 2008 that includes all the development and debugging features provided by the Visual Studio environment. This environment provides separate design surfaces for control flow, data flow, and event handlers, as well as a hierarchical view of package elements in the Package Explorer. The change in base technology of SSIS in this version from Visual Studio 2005 to Visual Studio 2008 for BIDS enables you to have both environments installed side by side on the same machine. BIDS 2008 provides several features that you will study later

in this chapter and subsequently use throughout this book. SQL Server Management Studio allows you to connect to Integration Services store to import, export, run, or stop the packages and be able to see list of running packages. You will also study SQL Server Management Studio later in this chapter.

Data Warehousing Loading

At the core, SSIS provides lots of functionality to load data into a data warehouse. The Data Flow Task is a special task that can extract data from disparate data sources using Data Source Adapters and can load into any data store that allows OLE DB and ADO .NET connections. Most modern systems use these technologies to import and export data. For example, SSIS provides a Bulk Insert Task in the Control Flow that can bulk-load data from a flat file into SQL Server tables and views. While the Data Flow includes destinations such as OLE DB Destination, ADO NET Destination, and SQL Server Destination, these destination adapters allow you to load data into SQL Server or any other data stores such as Oracle and DB2. While loading a data warehouse, you may also perform aggregations during the loading process. SSIS provides Aggregate Transformation to perform functions such as SUM and Average and use Row Count transformation to count the number of rows in the data flow. Here are several other Data Flow Transformations that allow you to perform various data manipulations in the pipeline:

- SSIS provides three Transformations—*Merge, Merge Join,* and *Union All* Transformations—to let you combine data from various sources to load into a data warehouse by running the package only once rather than running it multiple times for each source.
- *Aggregate Transformation* can perform multiple aggregates on multiple columns.
- *Sort Transformation* sorts data on the sort order key that can be specified on one or more columns.
- *Pivot Transformation* can transform the relational data into a less-normalized form, which is sometimes what is saved in a data warehouse.
- *Audit Transformation* lets you add columns with lineage and other environmental information for auditing purposes.
- A new addition to SSIS 2008 is the Data Profiling Task, which allows you to identify data quality issues by profiling data stored in SQL Server so that you can take corrective action at the appropriate stage.
- Using the Dimension Processing Destination and the Partition Processing Destination as part of your data loading package helps in automating the loading and processing of an OLAP database.

Most data warehouses need to maintain a slowly changing dimension. Integration Services provides a Slowly Changing Dimension (SCD) Transformation that can be used in the pipeline, enabling you to maintain a slowly changing dimension easily, which otherwise is not easy to maintain. The Slowly Changing Dimension Transformation includes the SCD Wizard, which configures the SCD Transformation and also creates the data flow branches to load the slowly changing dimension with new records, with simple type 1 updates and also updates where history has to be maintained, that is, type 2 updates. Another common scenario in data warehouse loading is the early arriving facts, that is, the measures for which dimension members do not exist at the time of loading. A Slowly Changing Dimension Transformation handles this need by creating a minimal inferred-member record and creates an Inferred Member Updates output to handle the dimension data that arrives in subsequent loading.

Standardizing and Enhancing Data Quality Features

Integration Services includes the following transformations that enable you to perform various operations to standardize data:

- *Character Map Transformation* allows you to perform string functions to string data type columns such as change the case of data.
- *Data Conversion Transformation* allows you to convert data to a different data type.
- *Lookup Transformation* enables you to look up an existing data set to match and standardize the incoming data.
- *Derived Column Transformation* allows you to create new column values or replace the values of existing columns based on expressions. SSIS allows extensive use of expressions and variables and hence enables you to derive required values in quite complex situations.

Integration Services can also clean and de-dupe (eliminate duplications in) data before loading them into the destination. This can be achieved either by using Lookup Transformation (for finding exact matches) or by using Fuzzy Lookup Transformation (for finding fuzzy matches). You can also use both of these transformations in a package by first looking for exact matches and then looking for fuzzy matches to find matches as detailed as you may want. Fuzzy Grouping Transformation groups similar records together and helps you to identify similar records if you want to treat the similar records with the same process, for example, to avoid loading similar records based on your fuzzy grouping criteria. The details of this scenario are covered in Chapter 10.

Converting Data into Meaningful Information

There is no reason to collect and process large volumes of data other than to draw out meaningful information from it. SSIS provides several components and transformations that you can use to draw out meaningful information from raw data. You may need to perform one or more of the following operations to achieve the required results:

- Apply repeating logic to a unit of work in the workflow using *For Loop* or *Foreach Loop* containers
- Convert data format or locale using *Data Conversion Transformation*
- Distribute data by splitting it on data values using a condition
- Use parameters and expressions to build decision logic
- Perform text mining to identify the interesting terms in text related to business in order to improve customer satisfaction, products, or services

Data Consolidation

The data in which you are interested may be stored at various locations such as relational database systems, legacy databases, mainframes, spreadsheets, or even flat files. SSIS helps you to consolidate this data by connecting to the disparate data sources, extracting and bringing the data of interest into the data flow pipeline and then merging this data together. This may sound very easy, but things can get a bit convoluted when you are dealing with different types of data stores that use different data storage technologies with different schema settings. SSIS has a comprehensive set of Data Flow Sources and Data Flow Destinations that can connect to these disparate data stores and extract or load data for you, while the Merge, Merge Join, or Union All Transformations can join multiple data sets together so that all of them can be processed using single pipeline process.

Package Security Features

A comprehensive set of security options available in SSIS enables you to secure your SSIS packages and the metadata specified in various components. The security features provided are:

- Access control
- Encrypting the packages
- Digitally signing the packages using a digital certificate

Depending on where the packages have been deployed, access control methods are provided by the underlying platform. For example, you can control access to packages saved into SQL Server using SQL Server roles for Integration Services, while Windows access control mechanisms are used if the packages are deployed to the file system. Integration Services packages can use various levels of encryption to protect sensitive information such as passwords and connection strings. You can also digitally sign your SSIS packages to establish the authenticity of the packages. Chapter 7 covers these security features in detail.

Service-Oriented Architecture

SSIS provides support for Service-Oriented Architecture (SOA) through a combination of HTTP connection manager, Web Service task, and XML source. These can be used together to pull XML data from URLs into the data flow.

SSIS Package as a Data Source

SSIS provides a DataReader destination that enables a SSIS package to be used as a data source. When you use a DataReader destination in your SSIS package, you effectively convert your SSIS package into an on-demand data source that can provide integrated, transformed, and cleansed data from multiple data sources to an external application such as SQL Server Reporting Services. You can also use this feature to connect to multiple web services, extract RSS feeds, and combine and identify interesting articles to be fed back to the application on demand. This is a very unique and powerful feature that places SSIS far ahead of other traditional ETL tools.

Programmability

SSIS provides a rich set of APIs in a native and managed form that enables you not only to extend the functionality provided by preconfigured components but also to develop new custom components using C++ or other languages supported by the .NET Framework (such as Visual C#, Visual Basic 2008). With the provision of this functionality, you can include your already-developed legacy applications or third-party components in SSIS processes, or you can program and extend SSIS packages by scripting or by writing your own custom components. These custom components can be developed for both Control Flow and Data Flow environments and can be included in an SSIS toolset quite easily so as to be reused in enterprise-wide development projects. Examples of custom components could be Control Flow tasks, Data Flow

Sources, Data Flow Destinations, Data Flow Transformations, Log providers, Connection Managers, and so on.

Scripting

SSIS also provides scripting components in both Control Flow and Data Flow environments to allow you to add ad hoc functionality quickly within your SSIS packages using Microsoft Visual Basic 2008 and Microsoft Visual C# 2008.

Easy Management of SSIS Packages

SSIS is designed with high development productivity, easy management, and fast debugging in mind. Some of the features that contribute to achieve these goals are listed here:

- Integration Services is installed as a Microsoft Windows service, which provides storage and management functions and displays running packages for SSIS packages.
- Integration Services provides rich logging features that allow you to choose the type of information you want to log at the package level or at the component level using one of the five built-in log providers, and if you're not happy with them, you have the flexibility to custom-code one that suits more to your requirements.
- If your package fails halfway through processing, you do not need to do all the work again. Integration Services has a restart capability that allows a failed package to be restarted from the point of failure rather than from the beginning, thus saving you time.
- Integration Services provides SSIS Service and SSIS Pipeline performance objects that include a set of performance counters for monitoring the running instances of packages and the performance of the data flow pipeline. Using these counters, you can fine-tune the performance of your packages.
- SSIS provides several utilities and wizards such as the dtexec utility, dtutil utility, Execute Package Utility, Data Profiler Viewer, Package Migration Wizard, and Query Builder that help you perform the work easily and quickly.
- SSIS provides the SQL Server Import and Export Wizard that lets you quickly copy data from a source to a destination. The packages saved with SQL Server Import and Export Wizard can later be opened in BIDS and extended. You will study the SQL Server Import and Export Wizard in Chapter 2.

Automating Administrative Tasks

SSIS can automate many administrative tasks such as backing up and restoring, copying SQL server databases and objects, loading data and processing SQL Server Analysis objects when you create the required logic in a package and schedule it using SQL Server agent job or any other scheduling agent.

Easy Deployment Features

You can enable package configurations to update properties of package components dynamically with the Package Configuration Wizard and deploy packages from development to testing and to production environments easily and quickly with the Deployment Utility. You will study deployment features and facilities in Chapter 11.

Legacy Support Features

You can install SQL Server 2008 Integration Services side by side with SQL Server 2005 Integration Services and SQL Server 2000 Data Transformation Services. Alternatively, you can choose to upgrade the legacy DTS 2000 or SSIS 2005 versions to the SQL Server 2008 version. Various installation options are discussed later in this chapter, when you will do an SSIS 2008 installation Hands-On. But here it is important to understand that SQL Server 2008 is a point upgrade of SQL Server 2005 Integration Services, though enough changes have been made that you cannot modify or administer packages developed in one version from the other version. However, run-time support has been maintained in SQL Server 2008; for example, you can run SSIS 2005 packages in SQL Server 2008 using BIDS, dtexec (2008 version), or SQL Server Agent. See Chapter 14 for more details on implications of choosing to upgrade or running the side-by-side option. DTS 2000 has been deprecated in SQL Server 2008 and is not included in the default installation option. The following section describes it in more detail. DTS packages can still be used with Integration Services, as legacy support still exists, but you will have to install DTS support components separately. SSIS 2008 also provides tools to migrate your DTS packages to Integration Services to enable you to take advantage of new features. You will study backward compatibility features and migration support provided in SQL Server 2008 in Chapter 14.

What's New in Integration Services 2008

While Integration Services 2005 was not only a complete rewrite of DTS 2000 but also a new product of its kind, SSIS 2008 contains several enhancements to increase performance and productivity. In this section, you will study the major enhancements

that have been included in SSIS 2008, while the others will be covered wherever we come across them. If you're new to Integration Services, you can skip this section, as this may not provide you relevant information. However, if you've worked with SSIS 2005, this section will acquaint you with the changes that have been made to Integration Services 2008.

Better Lookup

Most data integration or data loading projects need to perform lookups against already-loaded or standardized data stores. The lookup operation has been very popular with developers since Data Transformation Services first introduced this task. Integration Services 2008 has greatly improved the usability and performance of this component over its predecessor, SSIS 2005. The continuous growth in data volume and the increased complexity of BI requirements has resulted in more and more usage of lookup operations. As Integration Services 2005 was becoming a more appealing choice in data warehouses than ever, a better performing lookup was much needed because of the limited time-window available to such operations. Think of a practical scenario: if you have to load several flat files daily, it is most likely that you will be keeping your data flow task within a looping logic. And if you're using a Lookup Transformation in a data flow task, the lookup or reference data will be loaded every time the Lookup Transformation is used within the loop in Integration Services 2005. If your reference data doesn't change that often, then this recurring loading of reference data is a redundant operation and can cause unnecessary delays. Integration Services 2008 provides a much-improved Lookup Transformation that allows you to use a cache for the reference data set, and you don't need to perform a lookup against the reference data source repeatedly as you do in SSIS 2005. You can use an in-memory cache that is built before the Lookup Transformation runs and remains in memory until the package execution completes. This in-memory lookup cache can be created in the same data flow or a separate one and used over and over until the reference data set changes, at which time you can refresh the cache again. The ability to prepopulate the cache and to repeatedly use it makes the lookup operation perform much better in this version. And this is not all: you can also extend the use of in-memory cache beyond a package execution by persisting this cache to a cache file. The cache file is a proprietary raw-format file from which the cache data can be loaded into memory much faster than from a data source. Used in this way, a cache file enables you to share the cached reference data between multiple packages. Later, when you study Lookup Transformation in Chapter 10, you will also use a cache file and the other components used to create and use a cached lookup.

Improved ADO NET Components

DataReader Source and DataReader Destination components have been replaced with much improved ADO NET Source and ADO NET Destination components. DataReader adapters in SSIS 2005 allowed you to connect to ADO NET–compliant data stores; however, they were restrictive and could be configured only in an advanced editor. ADO NET adapters, on the other hand, have their own custom UI and look more like OLE DB Adapters, with the only difference being that they cannot use variables in the data access mode property. The enhanced functionality of ADO NET adapters enables SSIS 2008 to connect to ODBC destinations now.

Powerful Scripting

As mentioned earlier, BIDS is now based on VSTA (Visual Studio Tools for Applications), which is a Visual Studio 2008 IDE. This environment benefits both the Script Task and the script component by providing them a new programming IDE and an additional language, C#. In SSIS 2008 you can choose either Visual Basic 2008 or Visual C# 2008 as your preferred language. Replacement of Visual Studio for Applications (VSA) by VSTA has also made it easier to reference many more .NET assemblies and added real power to SSIS scripting.

Extended Import and Export Wizard

The Import and Export Wizard has been made more usable by extending the features it supports. You can now use ADO NET adapters within the Import and Export Wizard and take advantage of other enhancements; for instance, data type mapping information and data type conversions have been made available, along with better control over truncations and flexibility to create multiple data flows if you're dealing with several tables.

Ability to Profile Your Data

Sometimes you will receive data from external sources or from the internal lesser-known systems. You would want to check data quality to decide whether to load such data or not. May be you can build an automatic corrective action for such a data based on its quality. The ability to check quality or profile data is now included in Integration Services. The Data Profiling Task enables you to analyze columns for attributes such as column length distribution, percentage of null values, value distribution, and related statistics. You can actually identify relationship problems among columns by analyzing candidate keys, functional dependencies between columns, or value inclusion based on values in another column. SSIS 2008 provides a Data Profile Viewer application to see the results of Data Profiling Task.

Optimized Thread Allocation

The data flow engine has been optimized to create execution plans at run time. This enables data flow to allocate threads more efficiently and be able to perform better on multiprocessor machines; hence you get your packages processed quicker. You get this performance boost even without doing anything. This is an out-of-the-box improvement.

SSIS Package Upgrade Wizard

To help you upgrade your SSIS 2005 packages to the SSIS 2008 format, a SSIS Package Upgrade Wizard has been provided in this version. Though a SSIS 2005 package can be automatically upgraded to the SSIS 2008 format by opening in BIDS, this is a slow process if you have several packages in your projects. The SSIS Package Upgrade Wizard allows you to select packages from either File System or SQL Server MSDB database stores, select one or many packages at one time to upgrade, and keep a backup of the original packages in case you run into difficulties with upgraded packages.

Taking Advantage of Change Data Capture

The source systems that are used to populate a data warehouse are generally transactional systems hosting LOB applications that need the system not only to be available but also to perform at the best possible level. This virtually leaves only one option: for database developers to load a data warehouse during off-business hours. With more and more businesses using the Internet as a sales and marketing channel, either the off-business hours have reduced drastically or in many cases no off-business hours are left. This leaves very little or no time window for data warehouse processes to pick up the data from the source systems. Until recently, database developers have used triggers or timestamps to capture changed rows; however, the process makes systems complex and reduces the performance.

SQL Server 2008 includes a new feature called Change Data Capture that provides changes—that is, insert, update, and delete activities happening on the SQL Server tables—in a simple relational format in separate change tables and leaves the source systems working at their best. You will use this feature in Chapter 12 while studying the best practices for loading a data warehouse.

Benefiting from T-SQL Merge Statement

SQL Server 2008 includes a new T-SQL statement for performing insert, update, or delete operations on a table based on the differences found in another table. This enables you to perform multiple DML operations in a single statement, resulting in

performance improvement due to reduction in the number of times the data is touched in source and target tables. You can use Execute SQL Task to host the MERGE statement and leverage the performance benefit provided by this statement.

Enhanced Debugging

To debug pipeline crashes or deadlocks, you can now use command prompt options with the dtexec and dtutil command prompt utilities to create debug dump files. The options /Dump and /DumpOnError can be used with dtexec to create dump files either on certain events (debug codes) or on any error. The dtutil utility contains only the /Dump option and can create dump files on occurrence of any of the specified codes.

Inclusion of New Date and Time Data Types

Last but definitely not the least, Date and Time data types have been enhanced with introduction of the three new data types:

- **DT_DBTIME2** Includes fractional seconds support over DT_DBTIME
- **DT_DBTIMESTAMP2** Includes larger fractional seconds support over DT_DBTIMESTAMP2
- **DT_DBTIMESTAMPOFFSET** Supports time zone offsets

Where Is DTS in SQL Server 2008?

You might have worked with the DTS provided with SQL Server 2000. DTS is not an independent application in itself; rather, it is tightly bound with SQL Server 2000. DTS is a nice little tool that has provided users with great functionality and components. Some developers have even extended DTS packages by writing custom scripts to the enterprise level. Yet DTS has some inherent shortcomings; for example, it is bound to SQL Server, is not a true ETL tool, has a limited number of preconfigured tasks and components, offers a single design interface for both workflow and data flow that is limited in extensibility, and has no built-in repeating logic. Although you could fix all these shortcomings by writing a complex script, it wouldn't be easy to maintain and would be a big challenge to develop.

With the launch of SQL Server 2005 Integration Services Microsoft has replaced Data Transformation Services (addressed as DTS 2000 in this book) of SQL Server 2000. One thing you need to understand is that Integration Services is not a point upgrade of DTS rather it will be right to say that it is not an upgrade to DTS at all. The code for Integration Services has been written from scratch, thus, Integration

Services has been built from ground up. DTS was deprecated in SQL Server 2005 and now in SQL Server 2008 it has been removed from the default installation process; if you want to install DTS components, you have to choose it manually. Once DTS support components have been installed, you can modify the design or run DTS packages on SQL Server 2008. However, bear in mind that backward compatibility support has been provided to enable developers and organizations to migrate existing DTS packages to Integration Services and not to encourage development of new packages on DTS. You will read more about DTS support and the migration options in Chapter 14 of this book.

Before we move on to next section, I would like to stress a couple of facts again about DTS 2000 and SSIS. SQL Server 2008 Integration Services is not an upgrade to DTS 2000. Integration Services is installed as a Windows service and Integration Services service; it enables you to see the running SSIS packages and manage storage of SSIS packages. DTS 2000 was not a separate Windows service; rather, it was managed under the MSSQLSERVER service instance. Though it is highly recommended that you migrate your DTS 2000 packages to SQL Server 2008 Integration Services to take advantage of the better-performing, more flexible, and better controlled architecture, your existing DTS 2000 packages can still run as is under Integration Services.

Integration Services in SQL Server 2008 Editions

Not all the editions of SQL Server 2008 include Integration Services; in fact only Standard, Developer, Enterprise, and Premium Data Warehouse Editions have Integration Services. However, once you've installed Integration Services, you can use any of the SQL Server editions as a data source or a destination in your SSIS packages. In the following section you will study how Integration Services is spread across various versions of SQL Server 2008.

- **SQL Server 2008 Express Edition** The Express Edition of SQL Server 2008, including its two other siblings, called SQL Server Express with Tools and SQL Server Express with Advanced Services, is an entry-level free edition and does not include Integration Services. SQL Server Express Edition includes SQL Server Import and Export Wizard only. Though you cannot use Integration Services on this edition, you can run DTS packages on an Express Edition SQL Server when you install SQL Server 2000 client tools or DTS redistributable files on the computer. Installing this legacy software will install the DTS run-time engine on the SQL Server Express Edition. DTS 2000 packages can also be modified using SQL Server 2000 client tools. Also, note that the Express Edition doesn't support SQL Server Agent and, hence, your packages can't be scheduled.

- **SQL Server 2008 Web Edition** This is a low-cost SQL Server edition designed to host and support web site databases. As in the SQL Server Express Edition, the Integration Services components are limited to support the Import and Export Wizard only. The DTS 2000 run time can be installed and used as it can with the SQL Server Express Edition.
- **SQL Server 2008 Workgroup Edition** This edition of SQL Server 2008 is targeted to be used as a departmental server that is reliable, robust, and easy to manage. This edition includes the SQL Server Import and Export Wizard, which uses Integration Services to develop simple source-to-destination data movement packages without any transformation logic. Again, Integration Services isn't supported on this server, though basic components of SSIS do exist on this server to support the wizard creating data movement packages. As in earlier-mentioned editions, DTS 2000 support software can also be installed in this edition and used in a similar way. In fact, DTS components can be installed on any edition if required; however, it will be required more on the editions that don't have Integration Services support than the ones that do. The Workgroup Edition gives you a bit more than the Express Edition by enabling you to remotely modify DTS packages using the SQL Server Management Studio, as the Workgroup Edition supports SSMS.
- **SQL Server 2008 Standard Edition** The Standard Edition of SQL Server 2008 is designed for small- to medium-sized organizations that need a complete data management and analysis platform. This edition includes the full power of Integration Services, excluding some high-end components that are considered to be of importance to enterprise operations. The Integration Services service is installed as a Windows service, and BIDS, an Integration Services development environment, is also included. The separation of Standard Edition and Enterprise Edition is only on the basis of high-end components and does not impose any limitations to performance or functionality of components. What you get in Standard Edition works exactly as it would work in Enterprise Edition. The following components have not been included in this edition, however:
  - Data Mining Query Task
  - Data Mining Query Transformation
  - Fuzzy Grouping Transformation
  - Fuzzy Lookup Transformation
  - Term Extraction Transformation
  - Term Lookup Transformation
  - Data Mining Model Training Destination

  - Dimension Processing Destination
  - Partition Processing Destination
- **SQL Server 2008 Enterprise Edition** This most comprehensive edition is targeted to the largest organizations and the most complex requirements. In this edition, Integration Services appears with all its tools, utilities, Tasks, Sources, Transformations, and Destinations. (You will not only study all of these components but will work with most of them throughout this book.)
- **SQL Server 2008 Developer Edition** This has all the features of the Enterprise Edition.
- **SQL Server 2008 R2 Premium Editions** With the release of R2, Microsoft has introduced two new premium editions—the Datacenter and Parallel Data Warehouse Editions, which are targeted to large-scale datacenters and data warehouses with advanced BI application requirements. These editions are covered in detail in Chapter 12.

32-Bit Editions vs. 64-Bit Editions

Technology is changing quickly, and every release of a major software platform seems to provide multiple editions and versions that can perform specific tasks. SQL Server 2008 not only introduced various editions as discussed in the preceding section but also has 32-bit and 64-bit flavors. Though SQL Server 2000 was available in a 64-bit edition, it was not a fully loaded edition and ran only on Intel Itanium 64-bit CPUs (IA64). It lacked many key facilities such as SQL Server tools on the 64-bit platform—that is, Enterprise Manager, Query Analyzer, and DTS Designer are 32-bit applications. To manage the 64-bit editions of SQL Server 2000, you must run a separate 32-bit system. Moreover, 64-bit SQL Server 2000 was available in Enterprise Edition only and was a pure 64-bit edition with less facility to switch over.

On the other hand, the SQL Server 2008 64-bit edition is a full-featured edition with all the SQL Server tools and services available on the 64-bit platform, meaning you do not need to maintain a parallel system to manage it. SQL Server 2008 64-bit edition is available for Standard Edition and Enterprise Edition. It can run on both IA64 and x64 platforms and is enhanced to run on Intel and AMD-based 64-bit servers. You can run SQL Server 2008 and its components in 64-bit native mode, or you can run 32-bit SQL Server and 32-bit components in WOW64 mode. SQL Server 2008 provides a complete implementation of Integration Services in the 64-bit edition, though there are minor tweaks here and there. The performance benefits provided by 64-bit systems outweigh the costs and efforts involved, and it is also very simple to switch over to the 64-bit edition. If you're interested in knowing more about SQL Server 2008 Integration Services 64-bit editions, detailed information is provided in Chapter 13, along with discussion of performance and issues involved with it.

Integration Services Architecture

Now you understand the benefits Integration Services provides, with its vast array of features, and also know about various versions and feature sets associated with them. Before we move further and get our hands dirty by starting working with it, it's time to know its architecture. Once you understand its architecture, you will be able to appreciate how the various components perform their jobs to successfully execute an Integration Services package. Let's start with the architecture diagram provided in Microsoft SQL Server 2005 Books Online and shown in Figure 1-1.

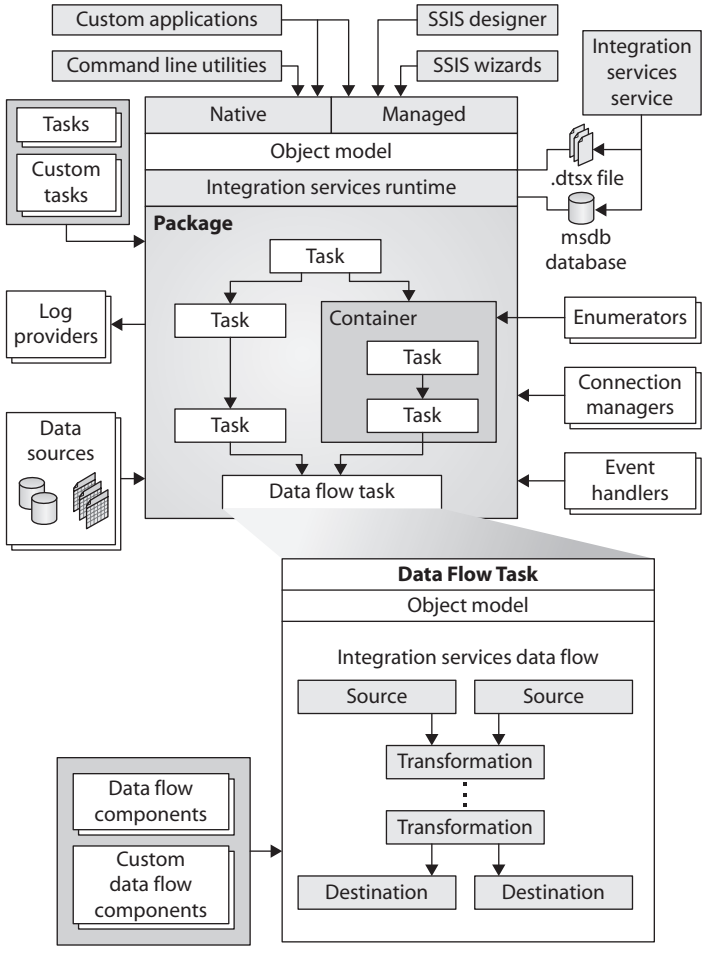


Figure 1-1 *Integration Services architecture*

Microsoft SQL Server 2008 Integration Services consists of the following four main components:

- Integration Services service
- Integration Services object model
- Integration Services run-time engine
- Integration Services data flow engine

Well, you've read little bit about these components earlier in this chapter as you were going through the features and uses of Integration Services. The following discussion on each of these components and their functions will clarify how Integration Services is architected.

Integration Services Service

Shown on the top-right corner of the architecture diagram (Figure 1-1), the Integration Services service is installed as a Windows service when you specifically choose Integration Services during installation. In the next section's Hands-On exercise, you will see where you make this choice and learn that choosing Integration Services specifically during installations installs other components as well. The Integration Services service allows you to execute Integration Services packages on local or remote computers, stop execution of running packages on local or remote computers, monitor running packages on local or remote computers, and connect to multiple Integration Services servers to manage multiple instances. In Figure 1-1, the Integration Services service points to a .dtsx file and MSDB database, implying that this service can manage SSIS packages stored to a file system or in an MSDB database within SQL Server 2008. The service manages SSIS packages by importing and exporting them from one type of storage location to another. You will learn a lot more about managing packages and their storage locations in Chapter 6.

You can connect to the Integration Services service using that SQL Server Management Studio, as you will do later in this chapter. Generally, with other software components, if the service is stopped, most of the components stop working. This is not true with Integration Services, because it is a component used to monitor running packages and manage their storage. You do not need to have this service running to design and run a package. You can save the newly designed package in BIDS on to the file system or in the SQL Server 2008 MSDB database and then execute it as well. However, you may find it a bit faster when running the Integration Services service, as it caches the metadata of the package and the connections. Also, if you need to monitor and list the packages using the SQL Server Management Studio, the Integration Services service must be running.

Integration Services Object Model

As mentioned earlier, Integration Services is a new product with an object model that supports both native and managed APIs. You can easily use this object model to write custom components such as tasks and transformations using C++ or any common language runtime (CLR)–compliant language. This object model provides easy accessibility for Integration Services tools, command-line utilities, and custom applications as shown on the top section of Figure 1-1. You can also develop custom components, build new packages, load and modify existing packages, and then execute them programmatically. This enables you to automate maintenance and execution of your packages completely. Programming Integration Services is a vast subject and deserves a separate book altogether. A complete discussion is beyond the scope of this book.

Integration Services Run Time

The Integration Services run time provides support for the package, containers, tasks, and event handlers during package execution. It also provides run-time services such as support for logging, breakpoints, connections to data sources and data stores, and transactions. You read earlier that Integration Services has two separate engines: a run-time engine for workflow and another engine for data flow. Basically, the Integration Services run time consists of whatever you configure in the Control Flow tab of BIDS plus the run-time services.

Integration Services Data Flow

As mentioned, the second engine of Integration Services provides services to the data flow within a package. This data flow is also known as the *pipeline* due to the nature of data flowing through various transformations one after another. The Data Flow Task is a unique task provided in the Control Flow tab of BIDS that encapsulates the data flow engine and the data flow components. Integration Services Data Flow consists of one or many Data Flow Sources; none, one, or many Data Flow Transformations; and one or more Data Flow Destinations. The Data Flow engine drives the data out of Data Flow Sources, brings it into pipeline, and lets the Data Flow Transformations perform the aggregations and conversions, merge data streams, conditionally split data into multiple streams, perform lookups, derive columns, and perform several other operations before loading the data into the destination data stores using Data Flow Destinations. You will work with Data Flow components in Chapters 9 and 10 in much detail.

Installing Integration Services

Now is the time to move forward and get our hands dirty by installing Integration Services. But take a few more minutes before we do that to learn about the installation options and the implications of these options. In real life, either you will be installing Integration Services on a clean Windows platform—that is, a system where no current or previous releases of SQL Server are installed—or you will be installing it on a computer that already has SQL Server 2005 Integration Services or Data Transformation Services of SQL Server 2000 installed. You may choose to install SQL Server 2008 Integration Services alongside SQL Server 2005 Integration Services or DTS 2000, or you may choose to upgrade the existing version of SSIS 2005 or DTS 2000. All these options and their implications have been discussed in the following sections.

Installing Integration Services on a Clean System

Most of the production systems are built using this method. Administrators prefer to install SQL Server on a fresh installation of a Windows server to avoid any debugging later on because of some old component on the server that doesn't work properly with SQL Server 2008 Integration Services. I recommend you use a sandbox for doing the Hands-On and install Integration Services clean so that you don't struggle initially with unwanted issues of compatibility or coexistence. You can install Integration Services either by using the SQL Server Installation Wizard or by running setup program from the command prompt.

You'll install Integration Services using the SQL Server Installation Wizard in the following Hands-On exercise. You will be installing SQL Server 2008 database engine and Integration Services together in this exercise; however, note that Integration Services does not require SQL Server in order to work. You can develop packages in Integration Services that connect to mainframes, Oracle or DB2 database servers, and output in flat files without installing SQL Server. A couple of high-end transformations such as the Fuzzy Lookup Transformation and Fuzzy Grouping Transformation need to create temporary tables for processing data in SQL Server and hence require connection to an SQL Server. However, even in this case, you do not need to have SQL Server running on the same local machine where your Integration Services package is designed or executed. Having said that, Integration Services is a fairly independent product and does not require SQL Server to operate; however, installing the SQL Server Database on the same server might prove beneficial, as most SSIS packages need to be run as SQL Server Agent jobs, which is a database engine feature.

Hands-On: Installing SQL Server 2008 Integration Services

This is your first Hands-On in which you will install SQL Server 2008 Integration Services using the SQL Server Installation Wizard on a clean system.

Method

It is important that you follow this process step-by-step, as this installation will be used throughout this book to create and run Integration Services projects. If you do not have SQL Server 2008 Enterprise Edition or Development Edition software, you can download the SQL Server 2008 Enterprise Evaluation Edition from Microsoft's download web site. This version is valid for 180 days and can be used for trial purposes. Details on non–Integration Services installation options are not covered here and are beyond the scope of this book. Refer to Microsoft SQL Server 2008 Books Online for more details on these installation options.

Exercise (Running the SQL Server Installation Wizard)

Load SQL Server 2008 DVD media in your computer's DVD drive and start the installation as follows:

1. After you load the DVD, the Autorun feature will open the Start screen, which displays various options. If Autorun doesn't do this, browse the DVD and run setup.exe from the root folder. Click the Installation hyperlink from the left sidebar and choose a new SQL Server stand-alone installation or else choose to add features to an existing installation to start the installation.
2. Setup first installs the .NET Framework 3.5 SP1 if it is not already installed. Some versions of Windows Server may require different versions of .NET Framework. Accept the License Agreement and click Install. The Installation Wizard will install .NET Framework, the SQL Server Native Client, and setup support files. Once installation completes, click Exit and the setup installs hot fixes for the operating system if it needs any. Click Finish to complete the preinstallation phase; it may require a restart to continue installation. After restart, again run setup.exe from the installation DVD and choose the New SQL Server installation link.
3. The installation program performs Setup Support Rules checks and lists pass, failure, and warning messages. Click OK to proceed further. On the Setup Support Files screen, click Install to install the required setup support files.
4. On the Feature Selection screen, choose Database Engine Services from the Instance Features section, choose Business Intelligence Development Studio, Integration Services, Client Tools Backwards Compatibility, Client Tools SDK,

SQL Server Books Online, and Management Tools—Complete from the Shared Features section as shown in Figure 1-2.

This is the most important step in the installation process, as you choose to install Integration Services here. However, even if you do not specifically select Integration Services to install, some components of Integration Services will still be installed because of they are needed to perform specific functions for other selected components. For example, the SQL Server Import and Export Wizard will be installed when you install SQL Server Database Services only. Also, most of the tasks and transformations will be available to you for developing your packages when you install Business Intelligence Development Studio without selecting Integration Services. Bear in mind that Integration Services is not required for developing and executing packages within BIDS; however, to run packages outside the development environment, you do need to choose Integration Services specifically at this step. Integration Services is installed as a Windows service and helps to manage storage of SSIS packages in SQL Server

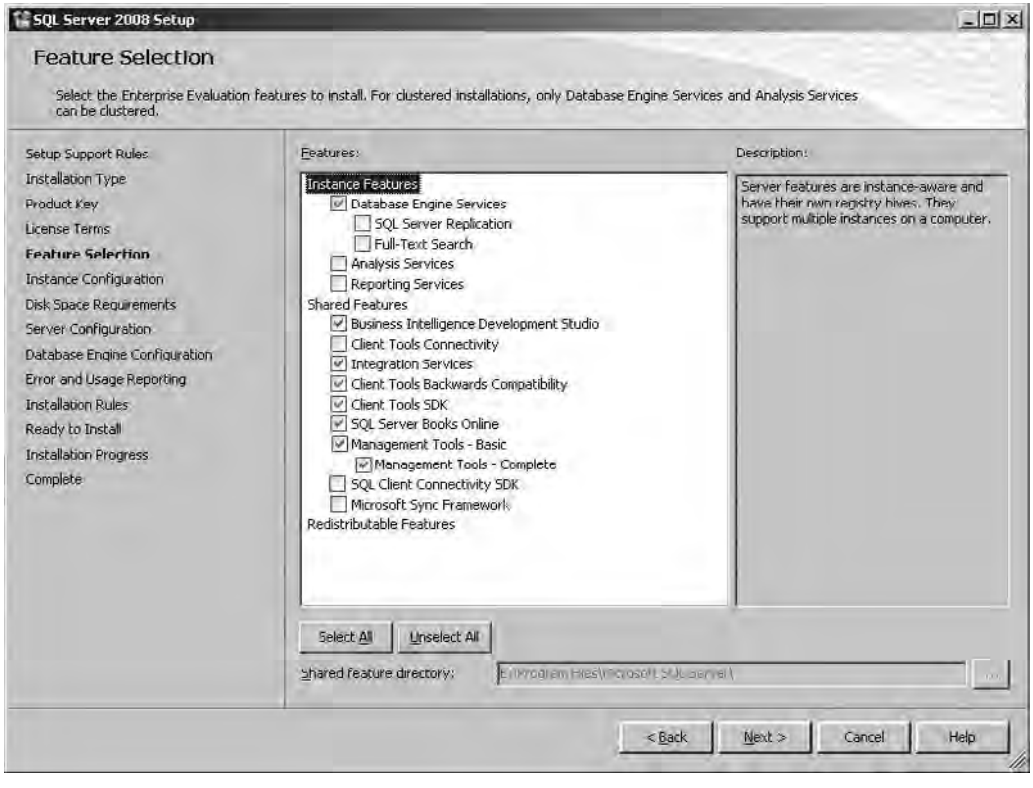


Figure 1-2 *Feature Selection for Integration Services*

or the file system using SQL Server Management Studio and enables you to monitor running packages in local as well as remote SSIS instances. Another benefit of installing Integration Services service is that it caches metadata of package components and speeds up loading of packages in Business Intelligence Development Studio. Selecting Integration Service also installs the ActiveX Script task and the DTS Package Migration Wizard.

Also, note that the Integration Services is not listed under Instance Features section. This means that you cannot install more than one instance of Integration Services. Though you don't have multiple instances of Integration Services on the server, it is instance aware. That is, it can connect to any instance of SQL Server and is not tied to a particular SQL Server instance. All you have to do is modify the Integration Services Configuration file to connect to a different SQL Server instance to store packages. More on this topic is covered in Chapter 6 of the book.

As mentioned earlier, Business Intelligence Development Studio is the designing tool for your packages, and selecting this installs the 32-bit design environment along with most of the tasks and transformations. Management Tools installs SQL Server Management Studio and is used to connect to Integration Services, manage storage of packages in the MSDB database and the file system, and monitor running packages.

Selecting Client Tools Backward Compatibility feature installs legacy support components and the Execute DTS 2000 Package task to enable you to run your DTS 2000 packages inside the Integration Services 2008 package. If you want the DTS 2000 run-time environment to be installed, you must install it separately. Chapter 14 covers more details on this subject. The Client Tools SDK will be required to install custom-developed managed assemblies for Integration Services, and finally, SQL Server Books Online is the documentation installation feature.

Select Shared Feature Directory and click Next.

5. Confirm that on the Instance Configuration page, Default Instance is selected and MESQLSERVER is specified in the Instance ID. Click Next. Again click Next to confirm the Disk Space Requirements.
6. Specify Account Name and password in the Server Configuration page. By default, Integration Service installs with NT AUTHORITY\NETWORK SERVICE account. It is recommended that you choose an appropriate domain account with minimum required permissions assigned. Proceed next to the Database Engine Configuration page to choose an authentication mode. Choose Mixed Mode and specify a password for the built-in system administrator account. Click Add Current User to add yourself to the administrators group and click Next three times to reach to Ready To Install page.
7. Review the options on the Ready To Install screen and press Install when ready.

8. You will see the installation of SQL Server 2008 components in the Installation Progress screen. This may take about 15 to 30 minutes to complete on a fast machine.
9. When the process completes, you will see the installation summary log in the Complete screen. Click Close to close the Installation Wizard. If prompted, restart your computer to complete the installation process.

Review

You installed the SQL Server 2008 database engine default instance and SQL Server Integration Services in the preceding exercise. In real life, you'll be installing Integration Services on 64-bit servers, as they are becoming more affordable and prevalent these days. Note that the 64-bit version of SQL Server 2008 software installs all the 64-bit components by default. However, BIDS is a 32-bit development environment for Integration Services and selecting this feature installs a 32-bit version of Integration Services tools, enabling you to run your packages in 32-bit mode on a 64-bit server. Also, BIDS is not supported on the 64-bit Itanium operating system and hence is not installed on Itanium servers. You can now check the programs that have been installed via the Control Panel. Also, open the Services console from the Administrative Tools and note the SQL Server 2008 services have been installed along with SQL Server Integration Services 10.0 service. Now you can play around with the software components installed by the Installation Wizard in the Programs group to make yourself acquainted with various components if you haven't had a chance to look at SQL Server 2008 up until now.

Installing Integration Services from the Command Prompt

Command prompt installation can help you roll out Installation to a team, install multiple nodes of a failover cluster, or use scripted installation files as a backup plan in case the worst happens. You can install the Integration Services by executing setup.exe along with parameters from the command prompt on a local or remote server. The parameters and their values can be specified either directly in the command or by use of an .ini file. The parameter-value pairs that are relevant for installing Integration Services and its components using a command prompt are as follows:

- **Action** This is a required parameter that specifies the installation type—install, upgrade, or repair. See Books Online for more parameter values.
- **Features** Indicates the SQL Server components to be installed, for instance, (SQL) for Database Engine, (IS) for Integration Services. Other options are (AS) for Analysis Services, (RS) for Reporting Services, and (Tools) for client tools.

- **ISSVCAccount** This is a required parameter option that specifies the service account for the Integration Services service.
- **ISSVCPassword** Specify a password for the ISSVCAccount using this option.
- **ISSVCStartupType** This is an optional parameter to specify the startup type for the service: automatic, manual, or disabled.

For example, if you want to install the SQL Server Database Engine, Integration Services, and client tools and online documentation, the syntax for the command will be something like this:

```
setup.exe /q /ACTION=install /FEATURES=IS /ISSVCACCOUNT="DomainName\
UserName" /ISSVCPASSWORD="StrongPassword"
```

To know about more available options, refer to Microsoft SQL Server 2008 Books Online.

Installing Side by Side

If you already have SQL Server 2005 Integration Services or SQL Server 2000 Data Transformation Services installed on a computer and don't want to remove or upgrade them, you can still install SQL Server 2008 Integration Services alongside them. They all can coexist on the same computer because all three have different execution and design environments. You may wonder, but in fact the SQL Server 2008 Integration Services has a different designer than its predecessor; for instance, BIDS in 2008 is built on a different architecture than BIDS 2005. Though they can coexist, there are some considerations for you to keep in mind when you're working with multiple versions of Integration Services.

SQL Server 2008 Integration Services has been optimized for performance over its previous version, SSIS 2005. While doing that, the development team at Microsoft has also made some underlying changes such as replacing the word "dts" with "ssis" from several places. As you can expect, this will mean that the code Integration Services works with in the 2005 version will most likely not work in the 2008 version. One such change affects storage of SSIS packages in SQL Server. The sysdtspackages90 table used in SQL Server 2005 to store packages in the MSDB database has been changed to the sysssispackages table in SQL Server 2008. It isn't hard to imagine that an Integration Services version won't be able to access packages across both versions due to different storage tables.

This also means that you cannot store packages in the MSDB database of one version that have been developed in another version of Integration Services. You must stick to the same version to open and modify packages using Business Intelligence Development

Studio. To clarify a bit more, we have a new version of Business Intelligence Development Studio in SQL Server 2008, which is based on Visual Studio 2008. BIDS also has a new scripting environment built around Visual Studio Tools for Applications (VSTA), replacing the earlier environment of Visual Studio for Applications (VSA) used in BIDS 2005. These underlying changes enable you to install BIDS 2008 side by side with BIDS 2005. However, this leaves BIDS 2008 unable to save packages in SSIS 2005 format. You can load and run packages in BIDS 2008 that have been developed in BIDS 2005; however, loading BIDS 2008 converts these packages into SSIS 2008 format and hence runs the package in 2008 environment. You can save this package in SSIS 2008 format but not in SSIS 2005. Hence if you want to modify your SSIS 2005 packages and want to keep them in 2005 format, you have to use BIDS 2005.

On the other hand, BIDS 2005 cannot load the higher version—i.e., SSIS 2008, packages at all. For similar reasons, you have a new version of the dtexec utility. While dtexec can run both SSIS 2008 and SSIS 2005 packages, it actually executes only SSIS 2008 format packages, as it converts SSIS 2005 packages temporarily to SSIS 2008 format before execution. This means that if your SSIS 2005 package can't be converted to SSIS 2008 format by dtexec, it can't be run. So, you must be a bit careful when working with the packages, as it may require you to keep multiple versions of the same package for SSIS 2005 and SSIS 2008.

On the other hand, things are rather simple, as Integration Services can coexist with DTS 2000 without any issues and there are no interoperability, issues as they are totally different in usability, operability, and many other respects. While Integration Services provides wizards and tools to migrate from DTS 2000 to its latest version, it still offers run-time support to run DTS 2000 packages as is. The Execute DTS 2000 Package task allows you to run a DTS 2000 package inside an Integration Services package. This task is not installed by default; you must choose the Client Tools Backward Compatibility feature during the Integration Services installation process.

Upgrading to SQL Server 2008 Integration Services

You can upgrade from SQL Server 2000 Data Transformation Services or SQL Server 2005 Integration Services. Both the Installation Wizard and the command prompt installation provides options to upgrade. With the Installation Wizard, select the Upgrade from SQL Server 2000 or SQL Server 2005 option, but for a command prompt installation, specify the /ACTION=upgrade option in the parameters.

When upgrading SQL Server 2005 Integration Services, you can upgrade either Database Engine and Integration Services together or just the Integration Services or just the Database Engine. The easiest option is to upgrade both Database Engine and Integration Services together when both are on the same computer. This option offers the fewest issues after upgrade, as the MSDB tables that store packages, metadata, and

log information are moved to Integration Services 2008 format and existing tables are removed after the upgrade. There are more things for you to do while upgrading that are discussed in detail in Chapter 14.

Upgrading Data Transformation Services to Integration Services 2008 is not that straightforward, though SQL Server 2008 provides tools to upgrade. This is due to major differences in the architecture of DTS 2000 and Integration Services. Chapter 14 discusses all the options that you have for upgrading your existing DTS 2000 packages to Integration Services.

Business Intelligence Development Studio

Business Intelligence Development Studio is designed for developing business intelligence solutions, including Integration Services packages, Reporting Services reports, and Analysis Services cubes and OLAP databases. BIDS is built on Visual Studio 2008, which allows you to design, build, test, execute, deploy, and extend Integration Services packages in an integrated development environment. Because of Visual Studio 2008 integration, BIDS provides the advantage of having integrated development features, a debugger, integrated help, and an integrated source control (such as Visual Studio Team Foundation Server or Visual SourceSafe) environment. You can use the same techniques to develop and store Analysis Services, Reporting Services, and Integration Services projects with BIDS. You can also develop a solution using BIDS that can have multiple Integration Services, Analysis Services, and Reporting Services projects.

BIDS is based on usual application design philosophy of solutions and projects. This provides lots of benefits, including the following:

- You don't need to have an SQL Server to develop an Integration Services package as was the case with DTS 2000, which required SQL Server 2000 Enterprise Manager to design and develop a package. Integration Services development and execution environments are fairly independent from the SQL Server Engine; however, the presence of SQL Server on the same server where the Integration Services service is running offers additional benefits such as ability to store packages in an MSDB database and to schedule running of SSIS packages using the SQL Server Agent.
- Your development gets an organized structure so that a number of projects can be run under the context of one solution and these projects further can be of different types such as Integration Services, Reporting Services, Analysis Services, or a C# class library project. Integration Services projects can contain data sources, data source views, SSIS packages, and other miscellaneous files. Availability of all of the support DDL and DML files at one place makes deployment a very easy task.

- Direct integration with Source Control servers such as the Visual Studio Team Foundation Server or Visual SourceSafe or one of many third-party source control servers facilitates immediate check-in whenever changes are made to projects. This feature enables a team to work on different parts of a project at the same time. BIDS doesn't work directly on packages stored in SQL Server, so during development, you should be saving your packages in the file system and checking in whenever making changes or on daily basis or on completion of a development cycle. For packages stored in SQL Server, BIDS imports them first by creating a copy on the file system and only then allows you to make changes to your code. After making changes, you will need to save those packages back into SQL Server. Hence, it is easier and better to work with the file system in development environment.
- Being built on the .NET Framework, BIDS makes it very easy to develop Custom Tasks or Components using .NET languages such as C# or Visual Basic 2008. As mentioned earlier, these custom development projects can also reside under the same solution.

The BIDS environment consists of various windows. Among the main windows are SSIS Designer, Solution Explorer, Toolbox, Variables, and Properties, in addition to other windows, such as Output, Error List, and Task List. All these windows can be docked anywhere and can be tabbed on to the main Designer window, set to autohide, or closed. These features provide a fantastic UI configuration feature that allows developers to customize their environment (to free up working space) and boost productivity.

Let's take a closer look at BIDS. In the following Hands-On exercise, you will create a blank Integration Services project and will learn various aspects of this tool.

Hands-On: Creating a Blank Integration Services Project

The objective of this exercise is to create your first blank Integration Services project and study various aspects of BIDS while working with it. You will use this project in Chapter 2 to add packages to it and take it further.

Exercise (Creating a Blank Integration Services Project)

In this part, you will learn about the Integration Services development environment within the Business Intelligence Development Studio while creating a blank project.

1. Choose Start | All Programs | Microsoft SQL Server 2008 and then click SQL Server Business Intelligence Development Studio to open this tool.
2. When the BIDS screen appears, choose File | New and then click Project. Alternatively, you can create a new project by clicking the Project URL next to the Create in Recent Projects section on the Start page. This will open the New

Project dialog box, in which the Business Intelligence Projects option is selected by default in Project Types. In the Templates pane, select Integration Services Project and then fill in the following details in the fields provided in the lower section of the dialog box (see Figure 1-3).

Name	My First SSIS Project
Location	C:\SSIS\Projects

Do not select the check box for Create Directory For Solution, as we do not want a parent folder for a solution to be created in this instance. Click OK when you have filled in these details to create an Integration Services Project.

3. The BIDS will create a blank Integration Services Project and will show you blank designer surface and the Package.dtsx SSIS package created in the Solution Explorer window as shown in Figure 1-4.

Let's take this opportunity to learn a bit more about the windows that make up an Integration Services project in the BI Development Studio.

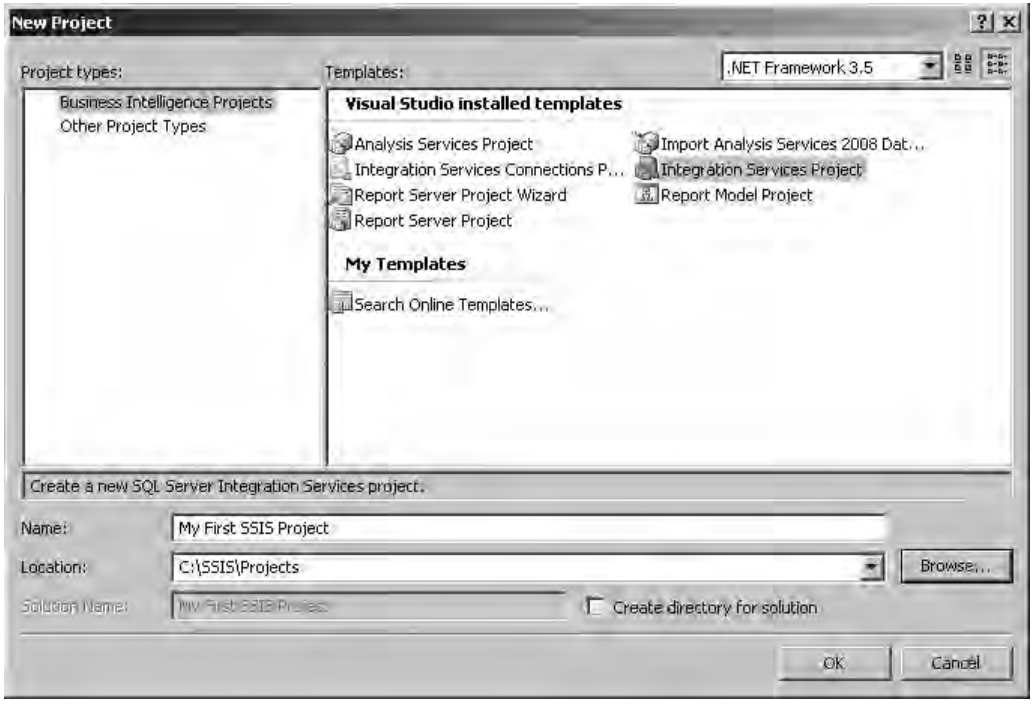


Figure 1-3 *Templates for creating new projects in BIDS*

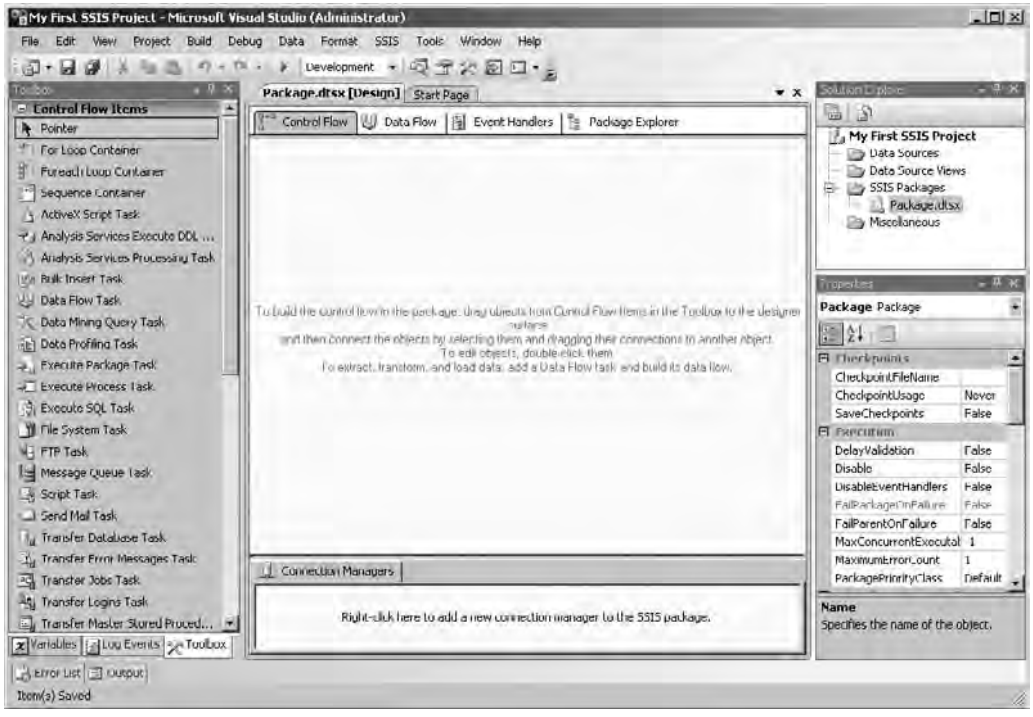


Figure 1-4 *Your first blank Integration Services project*

SSIS Designer

Notice the central area in the BIDS that has four tabs: Control Flow, Data Flow, and Event Handlers along with the design surfaces, and one Package Explorer tab to show components used in the package in a hierarchical structure; it is called the SSIS Designer. These are the design surfaces for developing Integration Services packages, while BIDS provides different designer surfaces for other BI project types such as Reporting Services and Analysis Services projects. The SSIS Designer provides a graphical view of objects that make data movement, workflow, and Integration Services development work possible with minimal or no programming at all.

At the top of the designer, the name of the package, Package.dtsx, is shown. It has three design surfaces—Control Flow, Data Flow, and Event Handlers—along with one Package Explorer tab to show components used in the package in a hierarchical structure. You can have multiple packages open at one time, each with its own design surfaces displayed in a tabular form. The layout of SSIS Designer displaying design surfaces in tab groups makes it particularly useful for handling large projects with many

packages open at the same time. There are other windows in the BIDS such as Solution Explorer, Toolbox, Properties, Error List, and Output. Not all the windows display at all times.

The workspace in BI Development Studio can be completely managed by the user as all the windows other than SSIS Designer can be docked anywhere in the working area, can float, can be tabbed in the SSIS Designer, can be set to autohide, or can be completely hidden or closed. If you click the down arrow button displayed in the Solution Explorer's menu bar, you will see these options. The other two buttons—a pushpin and a cross—are used to set windows to autohide or hide (close) respectively. By default, the Toolbox window is in autohide mode, hanging on the left side of the SSIS Designer, and the Solution Explorer is docked on the right side of the SSIS Designer. This is also shown in the Figure 1-4.

Solution Explorer

The Solution Explorer window provides an organized view of projects and files associated with them. The active solution lies at the top of the logical container and contains one or more different types of projects below it. You can add various other projects to a solution such as analysis services projects or reporting services projects to organize your different units of work all in one place.

Because the different projects store files in different ways, the Solution Explorer window does not reflect the physical storage of files. After you create an Integration Services project, you can attach it to a solution later on. You can attach a project or a file to a solution, but a project allows you to attach only files. You add files or projects by right-clicking the solution or the project in the Solution Explorer window and choosing Add from the context menu.

Properties Window

The Properties window is located at the lower right of the BIDS interface (see Figure 1-4). This is also a context-sensitive window and shows the properties of the item or object you have selected. Having the Properties window open during design time is a great time-saver. Like other windows, you can close the window or move it around on the desktop. You can open this window again from the View menu or by pressing F4.

Toolbox

You probably noticed a tabbed window on the left side of the SSIS Designer (see Figure 1-4) called the Toolbox. The Toolbox contains preconfigured tasks provided by Integration Services in the Control Flow tab, and the Data Flow tab has Data

Flow Sources, Data Flow Transformations, and Data Flow Destinations. The tasks are organized in two sections in the Toolbox window, which shows the tasks that are relevant to the project type you are working with. When you are in the Control Flow tab of the Designer window and open the Toolbox, you will see Control Flow Items; if you're in the Data Flow tab of the Designer, you will see Data Flow Sources, Data Flow Transformations, and Data Flow Destinations sections.

Control Flow Tab

The Control Flow tab consists of the tasks, arranged in the order in which they are performed—that is, precedence constraints and the looping structure defined by looping structure containers For Loop, Foreach Loop, and Sequence. You can draw objects on the graphical surface of the Designer and link them with other objects by dragging and dropping an arrow that extends from one object to another. This arrow signifies a precedence constraint and can be of type OnSuccess, which appears in green; OnCompletion, which appears in blue; and OnFailure, which appears in red. By defining the tasks and the precedence constraints between them, you design the control flow of your package and thus define the workflow in your package. You can logically group tasks to simplify complex packages and annotate them with text boxes to provide an explanation of the task. You will study more about precedence constraints and other control flow components in Chapters 3 to 5.

Data Flow Tab

The Data Flow Designer consists of the source adapters that extract data from heterogeneous data sources; the transformations that modify, aggregate, or extend data; and the destination adapters that load the transformed data into the final data stores. A package must include at least one Data Flow task in order to implement a data flow. You can create multiple Data Flow tasks in a package and create multiple data flows within a Data Flow task. The data is extracted from a source using source adapters and loaded to the destination using destination adapters. In between source adapters and destinations adapters you use transformations to modify, aggregate, and extend column data and to apply business logic to convert data. The flow of data from source to destination with transformations along the way is linked together with green or red lines called *data flow paths.* Adding data viewers to a path enables you to see the data as it moves from source to destination. This helps you debug and locate a troublemaking component that is converting data incorrectly. Extensive error handling can be included in the data flow task; for instance, error rows can be routed to a different destination whenever there is a row-level fault, to capture, analyze, and maybe correct and feed back to the main data flow.

Event Handlers Tab

You can extend package functionality by using *event handlers,* which are helpful in managing packages at run time. Event handlers are like subpackages waiting for the events to be raised so that they can come to life. They are powerful tools that can extend the package functionality greatly when properly implemented. Event handlers are created for the packages, Foreach Loop container, For Loop container, and Sequence container, as well as for the tasks in the same way as you create packages. Once created, event handlers can be explored in the Package Explorer tab by first expanding the Package and then expanding the Executables and finally expanding the Event Handlers node.

Package Explorer Tab

The Package Explorer represents the container hierarchy of the SSIS object model and lists all the package objects. This is the interface through which you can execute a package and monitor the running package. When you click the Package Explorer tab, your package appears at the top of the hierarchy. Click the Package to expand and expose the Variables, Executables, Precedence Constraints, Event Handlers, Connection Managers, and Log Providers objects. Event handlers are members of the Event Handlers collection, and all executables include this collection. When you create an event handler, SSIS adds the event handler to the Event Handlers collection. The Package Explorer tab in SSIS Designer lists the event handlers for an executable. Expand the Event Handlers node to view the event handlers that executable uses.

Progress Tab or Execution Result Tab

This tab doesn't show up during design time when you are still developing the package. When you run an Integration Services package in BIDS, a Progress tab will appear within the SSIS Designer. This Progress tab converts to an Execution Result tab once the package execution is completed and you've switched back to the design mode. Integration Services writes extensive information in this tab while executing a package. All sorts of information such as validation of tasks, start time, finish time, warnings, errors, failure messages, suggestions, and execution details, including the number of rows affected, are all written here. Once you become more familiar with Integration Services, you may find that this tab contains a bit more information than what you would like to see. For these reasons, you may find that Output window contains more concise and relevant information. However, if you are new to Integration Services, this tab provides some interesting information, such as the optimization hints in case more columns have been extracted than used in the data flow, and you may find these kinds of messages a learning aid to design better and more efficient packages.

Connection Managers Area

In this area, you add various connection managers, depending on the requirements of your package. For example, if your package needs to connect to a flat file, you will add a Flat File connection manager; if your package needs to connect to an SQL Server database, you will add an OLE DB connection manager.

BIDS provides many other windows for additional informational purposes. Some of these are described next, while others will be introduced wherever used.

Code Window

You can see the code of a package or an object in the SSIS Designer. To see this, go to the Solution Explorer window, right-click Package.dtsx, and choose View Code from the context menu. An additional tab on the SSIS Designer surface appears with a listing of code in XML form.

Task List Window

In the Task List window, you can add notes for descriptive purposes or as a follow-up for later development; you can also organize and manage the building of your application. To open the task list, choose View | Task List. These tasks can be filtered and sorted based on the predefined options provided.

Output Window

The Output window displays informational messages, including errors that occur during building, deployment, or run time. Third-party external tools can list their output to the Output window as well. Press the green triangle (play) button on the Standard toolbar to debug the package. The package completes execution immediately, as there is no task within the package, and you can see Output window open at the bottom of the screen among other windows. If you don't see it there, go to View menu and click Output from the option list to open it. Note that the Output window shows package start, finish, and success messages. The Output window has multiple panes to show information specific to the process that is sending informational messages. These panes can be accessed from the Show Output From drop-down box in the Output window. Click the down arrow and select Build to display messages from the build process.

Error List Window

This window provides detailed description of validation errors during design time. It also shows errors, warnings, and messages for the package you are developing. To open this window, choose View | Error List.

Locals Window

This window displays information about the local expressions and is available only when the package is in debug mode. When you use a breakpoint to stop package execution to debug, you can come to the Locals window and see values of variables at that particular moment. This is very useful and effective when you're debugging your package.

Watch Window

Like the Locals window, the Watch window is also available during debug mode and displays information only about the expressions that you've selected. For example, you can watch a variable to see how its value changes as the package execution progresses. You can open up to four watch windows.

1. Stop debugging of the package by pressing SHIFT-F5.
2. Using Windows Explorer, go to the C:\SSIS\Projects folder and note that various files have been created under the My First SSIS Project folder. Note the extension and type of these files, as each one of them represents a different function in the Integration Services project.
   - The *.sln file is the main file for a solution or project and contains solution configuration settings and the list of projects the solution contains.
   - The *.dtproj files are similar files, but contain information for project configurations and the items such as packages they contain.
   - The *.database file contains information required for internal use by BIDS.
   - The .dtsx files are the code files for your packages. These files are independent and self-contained and are not hard-bound with the solution in which they are created. This means that they can be freely copied between projects and folders.
   - The file *.suo, called Visual Studio Solution User Options, and the file *.dtproj.user, called Visual Studio Project User Options, are the two user settings files used by Integration Services at the solution and project levels.
   - The bin folder keeps the backups of the previous build versions of packages.

Congratulations! You have completed your first Integration Services project Hands-On.

Review

You've had your first encounter with the BIDS and created a blank Integration Services Project. Although you haven't developed much in the package, you have understood enough about the environment and the package itself that you're almost ready to get creative and develop a working package.

SQL Server Management Studio

SQL Server Management Studio is the main tool used to manage SQL Server databases and run T-SQL queries against the tables and the views. This tool also enables you to connect to the Integration Services service and perform management operations such as run packages, monitor running packages, manage package storage on the file system as well as the MSDB database, import and export packages from one storage area to another, assign Package Roles to control access, and upgrade packages. Let's start using this tool straightaway with a very simple Hands-On.

Hands-On: Connecting to Integration Services Service

In this exercise, you will connect to Integration Services and will also explore where DTS 2000 packages can be managed within SQL Server Management Studio.

Exercise (Using SQL Server Management Studio)

You have used BIDS earlier to create a blank project. While BIDS provides a development environment for developing SSIS packages, SQL Server Management Studio enables you to manage the deployed packages. In this part, you will connect to Integration Services using SQL Server Management Studio.

1. From Start | All Programs | Microsoft SQL Server 2008 and then click SQL Server Management Studio.
2. When the SQL Server Management Studio loads, you will see the Connect To Server dialog box, where you can choose a server type to which you want to connect and provide your authentication details. Click in the Server Type field and select Integration Services from the drop-down list. Type **localhost** in the Server Name field to connect to the local server and press Connect.
3. SQL Server Management Studio will connect to Integration Services and show you Running Packages and Stored Packages folders under Integration Services in the Object Explorer, as shown in Figure 1-5.
4. Expand these folders. You will not see any packages listed under Running Packages because no packages have yet been created. Click the Stored Packages folder and you'll see the File System and MSDB folders. This is where you will be managing the packages you've stored in the file system or the MSDB database in SQL Server. Managing these folders and the packages is covered in detail in Chapter 6.
5. Let's do a little more research to see where DTS 2000 packages go. In the Object Explorer window, click Connect and choose Database Engine from the drop-down list. Note that you can connect to Integration Services from here as well. Leave localhost as the Server Name and verify that Windows Authentication is selected in the Authentication field. Click Connect to connect to the local database server.

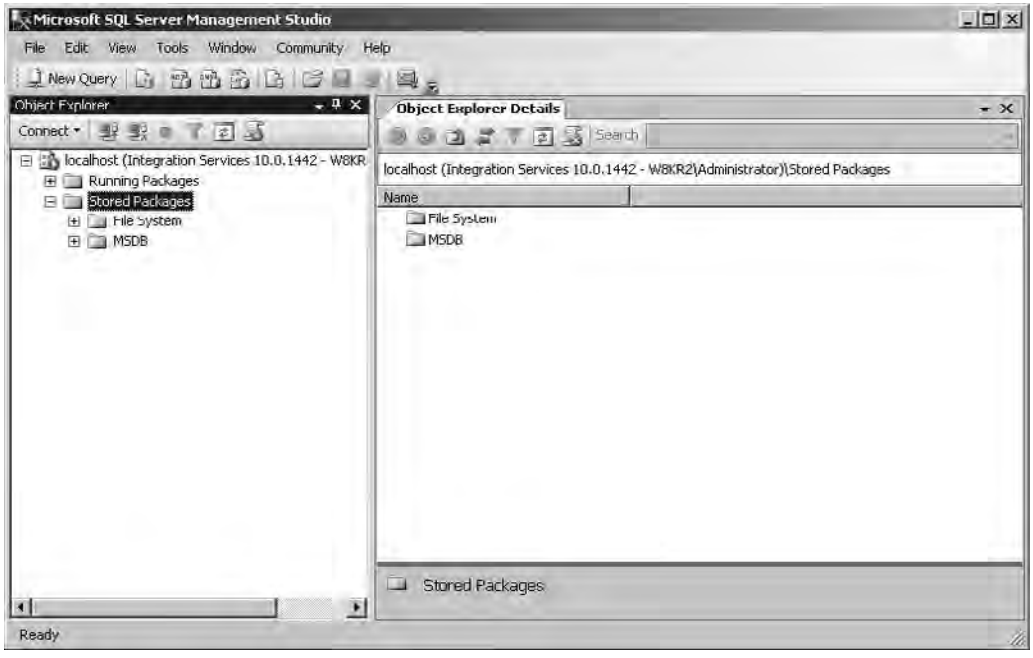


Figure 1-5 *Connecting to the Integration Services service*

6. Under the database server, expand the Management node and then the Legacy node. You will see a Data Transformation Services node. This is where you will be managing DTS 2000 packages that are imported into SQL Server 2008. How you run existing DTS 2000 packages or migrate them to the Integration Services format is covered in depth in Chapter 14.
7. Close SQL Server Management.

Review

Though you haven't yet built anything from the development point of view, you've seen quite a lot. If you've used DTS 2000, it may have already answered many of your questions, but if you are new to SQL Server, you can appreciate how easy it is to use SQL Server Management Studio to connect to Integration Services and manage SSIS packages storage locations. Also, you know where you can see DTS 2000 packages imported into SQL Server 2008. However, if you haven't installed the Integration

Services as we did in earlier exercise or are trying to connect to an already existing Integration Services service on a remote server and run into issues, you may like to refer to the section "Connecting to Integration Services Service" in Chapter 6 to understand the issues involved in connecting to the Integration Services service.

Summary

You have been introduced to Integration Services by following a couple of simple Hands-On exercises and reading a description of its architecture, features, and uses. You understand how various components of Integration Services work together to provide the manageability, flexibility, and scalability you need for your SSIS packages. You also appreciate the fact that all the components can be programmatically controlled and that custom tasks and transformations can be written using languages such as C++ or any CLR-compatible language. You know that Integration Services has two separate engines that provide workflow and data flow functions. You are now ready to launch into the realm of bigger challenges.

This page intentionally left blank

Chapter 2

Getting Started with Wizards

In This Chapter

- Starting SQL Server Import and Export Wizard
- Using Business Intelligence Development Studio
- Integration Services Connections Project Wizard
- Analyzing Data Quality with the Data Profiling Task
- Summary

Businesses of today are dealing with customers via a variety of channels—postal mail, telephone, e-mail, text message, web portal, agent, dealer, direct marketing, through business partners, and the list goes on. All these channels have their own systems to support business operations and result into data stored in a variety of data storage systems using diverse methodologies, and in different geographical locations. Along with storage requirement considerations, the sheer volume of data can be overwhelming.

Data must be integrated from all disparate sources rapidly within the organization to help it better understand its customers. It is the job of information analysts to integrate data stored in disparate sources. The toolset used to implement data consolidation are commonly known as extraction, transformation, and loading (ETL) tools. SQL Server Integration Services (SSIS) makes the job easy by providing various components that work with most of the data sources and the data stores, and transform data to meet even the most complex requirements.

However, before you jump-start the data integration process, it would be wise to understand the underlying issues with your data. Information is critical but only if accurate. To merge data from different sources, you should make sure that the data is accurate and consistent. Integration Services 2008 provides a Data Profiling Task to profile and analyze such data quality issues. This task generates statistical information in XML format that can be reviewed using Data Profile Viewer. After reviewing the statistical information and taking the corrective actions, you are ready to integrate your data streams.

The SQL Server Import and Export Wizard is the easiest utility to help you move data around. Its interactive and simple graphical user interface (GUI) allows even beginners to use the wizard for learning purposes and simple package development. Seasoned developers also use it for creating basic packages quickly, to be later extended using Business Intelligence Development Studio (BIDS) to reduce development time. The wizard is designed to perform simple tasks quickly and easily; hence it has limited transformation capabilities.

In this chapter you will study

- The SQL Server Import and Export Wizard
- The Integration Services Connections Project Wizard
- The Data Profiling Task and Data Profile Viewer

As the Data Profiling Task works with the data saved in SQL Server 2000 and later, you will be using SQL Server Import and Export Wizard to import data first in SQL Server and then will use Data Profiling Task to analyze the quality of data. This should be your standard practice even in real life while working with Data Profiling Task.

Starting SQL Server Import and Export Wizard

While you can use the SQL Server Import and Export Wizard to import and export data from any SQL Server database, the wizard can also move and transform data that may or may not involve SQL Server.

SQL Server Import and Export Wizard is handily available at locations where you would need it. Following are the locations from where you can start the wizard:

- DTSWizard.exe file
- Microsoft SQL Server 2008 program group
- SQL Server Management Studio
- Business Intelligence Development Studio (BIDS)

The behavior of SQL Server Import and Export Wizard varies slightly based on the location from where you start it. You will use different methods to start SSIS Import and Export Wizard while working in the following exercises. Before we progress further, install the software provided for the exercises in the book. Please refer to the Appendix for further details on this.

Hands-On: Importing a Flat File into SQL Server 2008

In your first exercise, you will import a simple text file into SQL Server 2008 and create a new database while importing this file.

1. Choose Start | Run. In the Run dialog box's Open field, type **DTSWizard**; then click OK to open the SQL Server Import and Export Wizard's Welcome page, as shown in Figure 2-1. (You can also run this file from the command prompt or by double-clicking the DTSWizard.exe file in the default location of C:\Program Files\Microsoft SQL Server\100\DTS\Binn.)
2. Click Next to open the Choose A Data Source window. By default, the wizard is set to point to a (local) server using SQL Server Native Client 10.0. Click the down arrow next to the Data Source field to see the other available data sources to which this wizard can connect. The drop-down list shows providers for several data sources that are installed on the local machine, including the following:
   - Flat Files
   - Microsoft Excel
   - Microsoft Access
   - .NET Framework Data Provider for Oracle

Figure 2-1 *The Import and Export Wizard's Welcome page*

- Microsoft OLE DB Provider for SQL Server
- SQL Server Native Client 10.0

3. The Import and Export Wizard can connect to and extract data from any of these data sources. Select Flat File Source from the Data Source drop-down list. The options on screen change to match the Flat File Source requirements.
4. Click the Browse button next to the File Name field, and in the Open dialog box navigate to the C:\SSIS\RawFiles folder. Select the CSV files (*.csv) as the file type to display .csv files, and then choose the RawDataTxt.csv file and click Open. You'll see that most of the options in the flat files fields will be automatically filled in for you as the wizard reads the file format.
5. Click the Column names in the first data row check box option, as shown in Figure 2-2, and then click Columns in the pane on the left.

   On this page, data is listed in the bottom half of the window. Note that the Column delimiter is Comma by default and that the headings of columns are displayed properly and are actually picked up from the first row of the text file.

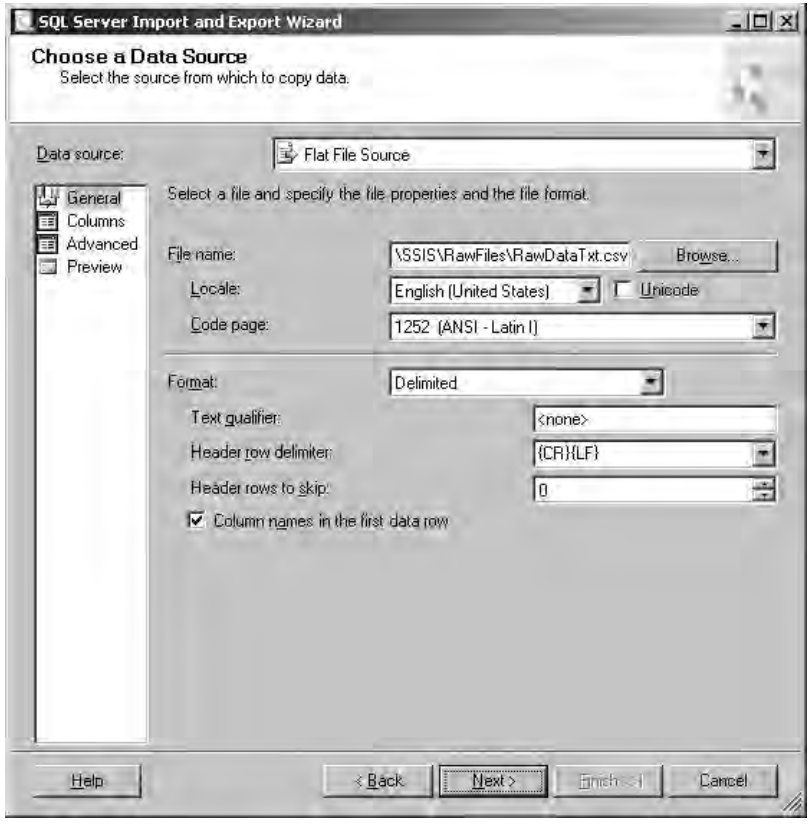


Figure 2-2 *Selecting a data source in the SQL Server Import and Export Wizard*

Go to the Advanced page by clicking it in the left pane. You'll see the list of columns in the middle pane and the column properties for the selected column in the right pane. You can change column properties such as Data Type or OutputColumnWidth here. To help you quickly identify the data types and length of the columns in the flat file a Suggest Types button has been provided. Click Suggest Types to see the options available in the Suggest Column Types dialog box shown in Figure 2-3. If you feel that the data types suggested by the flat file adapter are not correct, the most likely reason is that the flat file adapter didn't sample enough rows to guess it right. You can change this in Number Of Rows box by increasing the number of rows to sample up to a maximum of 9999. Note that if you choose a very large number here, the flat file adapter will take more time to suggest data types and you still must check that the appropriate column lengths and data types have been selected. Review the other options and click Cancel to come back.

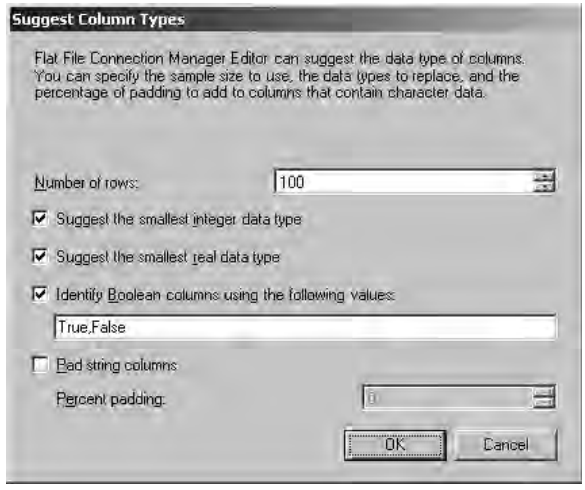


Figure 2-3 *Specifying a number of rows to sample appropriately*

Finally, if you click Preview, you will be able to set a value in the Data Rows to Skip field. Click Next to move on.

6. In the Choose a Destination window, leave the default SQL Server Native Client 10.0 selected in the Destination field. Microsoft SQL Server Native Client 10.0 provides an OLE DB interface to Microsoft SQL Server databases and supports several new features added to SQL Server 2008 such as Table-Valued Parameters, FILESTREAM, Large CLR UDTs, Sparse Columns, extended Service Principal Names (SPN) support, and Date and Time improvements.

   Just to remind you once again, you don't need to have SQL Server to run the SQL Server Import and Export Wizard or BIDS for developing Integration Services packages. You can actually move or transform your data from any source to any destination without involving SQL Server. So, for example, you can choose Destination as a text file and remove some of the columns from the destination file (so that you generate a modified text file from a text file), one operation that isn't easy to perform on text files otherwise.

   You can specify your server name in the Server Name field or select a name from the drop-down list by clicking the down arrow. Leave (local) specified for this exercise and leave the radio button for Use Windows Authentication selected.

   In the Database field, either you can specify a database name or you can opt to create a new database. The software provided for the book contains files for the Campaign database that you will use to perform various exercises. Please refer to the Appendix for further details on this. If you haven't attached the provided Campaign database yet to your SQL Server database engine, you can create it here by clicking the New button. Click New to create a new database.

7. If your login for SQL Server is correct, a new Create Database dialog box will open. Note that even if you can connect and open this dialog box, you still are required to have appropriate permissions on the SQL Server to create a new database; otherwise you'll a get create database permission denied error when you click OK to create a new database. Type **Campaign** in the Name field. The wizard will complete the rest of the details using the default database options similar to those shown in Figure 2-4. Leave the default options as is, and click OK to create the Campaign database. Note that if you've already attached the Campaign database provided with the software for this book, then you do not need to create the Campaign database here. Click Next to proceed.
8. In the Select Source Tables And Views window, verify that the check box next to the filename listed under Source is selected. The Destination will automatically be filled in with a table name. Click Edit Mappings. In the Column Mappings

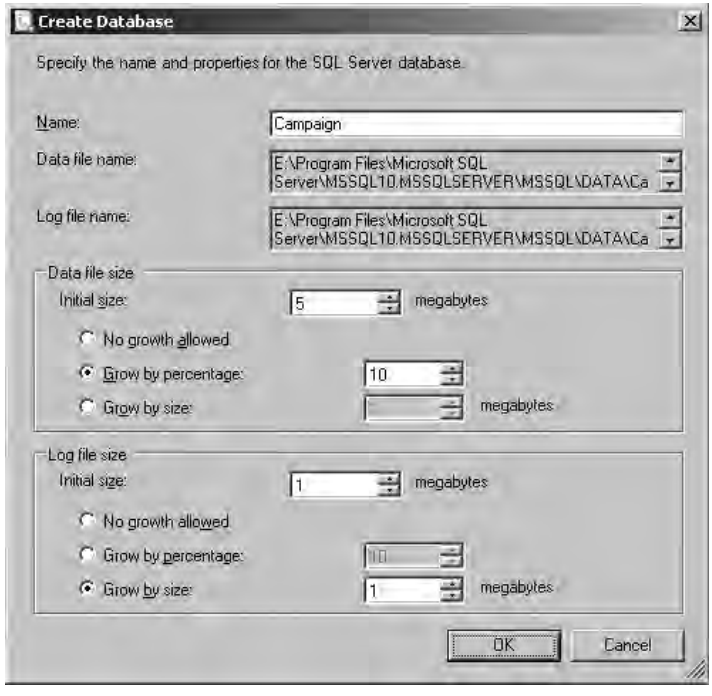


Figure 2-4 *Creating a new database in the SQL Server Import and Export Wizard*

dialog box as shown in Figure 2-5, you have various options to configure your destination table and the way you want to insert data into it. Following is the description of each of these options:

- **Create destination table** This option will be selected in your case. Choose this option whenever you want to create a destination table. When this option is highlighted and selected, you can click Edit SQL to generate an SQL statement that will be used to create this new table. You can customize this SQL statement; however, if you modify the default SQL statement, you will have to manually modify the column mappings.
- **Delete rows in destination table** This option will not be available to you, as the destination table is yet to be created. However, if you select an existing table to import data into, you can then choose to delete rows from the existing table.
- **Append rows to the destination table** If you're importing into an existing table, you can choose to append the rows to the existing table. However, in your case, this option will be grayed out.

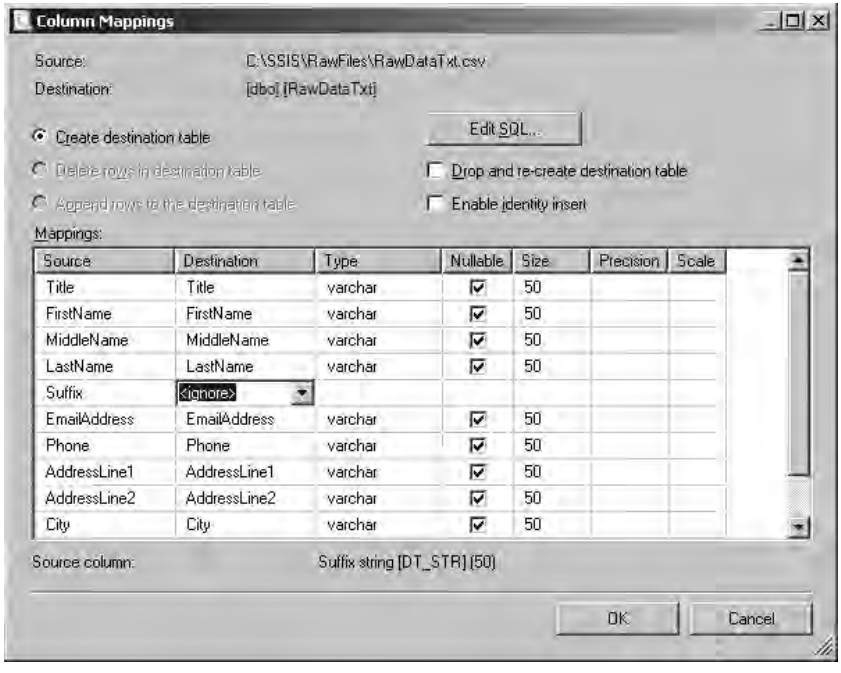


Figure 2-5 *Making changes in the mappings between Source and Destination*

- **Drop and re-create destination table** This check box option allows you to delete an existing table and then re-create and load data into it.
- **Enable identity insert** You will be using this option if you want to keep the identity in the input data and want to insert it into the destination table.

9. Let's change a mapping in the Mappings section to see how it works. Click in the Suffix destination column and then the drop-down box to select <Ignore>, as shown in Figure 2-5.

   Now click Edit SQL, and note that the SQL statement for creating the destination table has picked up the change you just made. You can make changes to the SQL statement or use already existing scripts to create a table. However, once you have modified the SQL statement, any subsequent changes made from the Mappings section will not be automatically picked up by the SQL, but must be manually edited. First click Cancel and then click OK to return to the Select Source Tables And Views window. Click Next to move on to the Save And Run Package dialog box.

10. By default, the Run Immediately option will be selected. Check the Save SSIS Package option to save this package, and then select the File System radio button. Click the drop-down arrow in the Package Protection Level dialog box to see the available options. Integration Services protects your package according to the protection level you select for your package. You will study these options in detail in Chapter 7; however, brief descriptions of these options are provided here for your quick review:

    - **Do not save sensitive data** Using this option, you choose not to save sensitive information such as connection strings, passwords, and so on within the package.
    - **Encrypt sensitive data with user key** This is the default option that can save sensitive information within the package after encrypting it using a key that is based on the user who creates and saves the package. With this option selected, only the user who creates the package will be able to run the package without providing the sensitive information. If other users try to load the package, the sensitive information is replaced with blank data and the package will fail to run unless they provide the sensitive information manually.
    - **Encrypt sensitive data with password** With this option, you choose to save the sensitive information in the package after encrypting it with a password. Once the package is encrypted with this option, any user will be able to open and execute the package by providing the password. If the user is unable to provide the password, he or she will have to provide the sensitive information manually to run the package.

- **Encrypt all data with user key** Selecting this option means that all the information and metadata in the package will be encrypted using a key that is based on the user who is creating the package. Once the package has been encrypted using this option, only the user who created the package can open and run it.
- **Encrypt all data with password** With this option, you can encrypt all the information and the metadata in the package by providing a password. Once the package has been encrypted with this option, it can be opened and run only after providing the password.

If you choose to save the SSIS package into SQL Server, you will see one additional option in the Package protection level:

- **Rely on server storage and roles for access control** When you save your packages to the MSDB database of an SQL Server, Integration Services provides additional security features that allow you to secure packages based on the three fixed database-level roles: db_ssisadmin, db_ssisltduser, and db_ssisoperator. You will study more about these roles and the ways to use them to control access to your packages in Chapter 7.

11. Select the Encrypt sensitive data with password option and type **12wsXZaq** in the Password and Retype Password fields and click OK.
12. Type **Importing RawDataTxt** in the Name field and **Importing comma delimited RawDataTxt file** in the Description field. Type **C:\SSIS\Packages\Importing RawDataTxt** in the File Name field, as shown in Figure 2-6.
13. Click Next to open the Complete The Wizard screen. Here you have the option of reviewing the choices you have made. Once you are happy with them, click Finish.

    The wizard now runs the package created, imports all the records, and shows a comprehensive report for all the intermediary steps it processed, as shown in Figure 2-7. This window not only shows any messages or errors appearing at run time but also suggests some of the performance enhancements that can be applied to your SSIS packages.

    You can see the detailed report by clicking Report and selecting View Report. You can also save the report to a file, copy it to the Clipboard, or send it as an e-mail using the Report button.
14. Click Report and choose View Report. Scroll down to the Validating section to read that SQL Server Import and Export Wizard suggests removing the Suffix column from the source selections to improve the performance of execution, as this is not used in the Data Flow task. The data is handled in the memory buffers

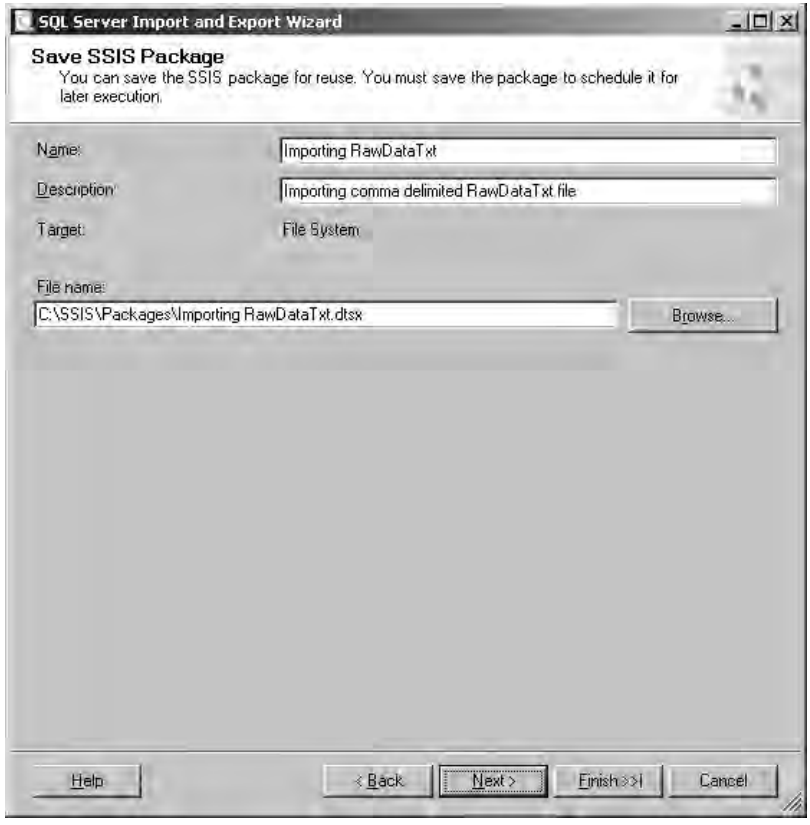


Figure 2-6 *Saving SSIS package to the file system*

as it flows through the data flow or the pipeline. More columns mean that fewer rows can be accommodated in a memory buffer. So, removing unused columns will save memory space and allows more rows to be accommodated in a buffer, making the data flow task more efficient. It is always worth taking a quick look at this report, which may offer valuable comments about the package performance. Close all the windows.

15. Start Windows Explorer and navigate to the C:\SSIS\Package folder. You will see a file called Importing RawDataTxt.dtsx saved in this folder. This is the Integration Services package saved in the XML format with an extension of *.dtsx*. Right-click this file, choose Open With, and then choose Notepad from the list of programs. You will see that the package settings listed in this file are in XML format. Close Notepad without making any changes.

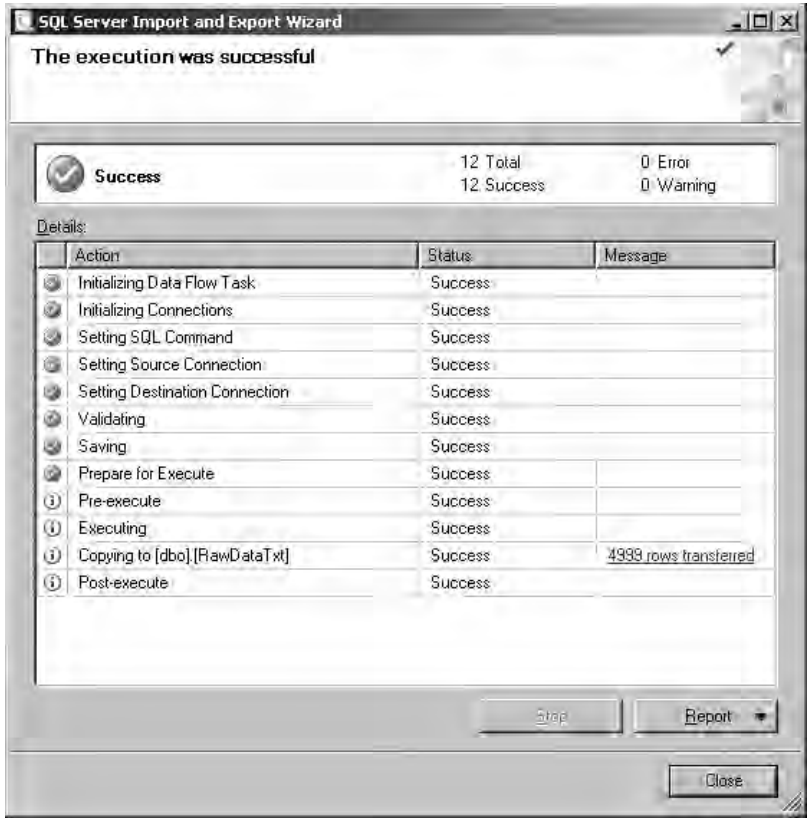


Figure 2-7 *Checking the execution results*

16. Choose Start | All Programs | Microsoft SQL Server 2008, and click SQL Server Management Studio to start the program.
17. When the SQL Server Management Studio starts, you will be asked to provide authentication and connection details in the Connect To Server dialog box. In the Server Type field, select Database Engine. You can connect to various services such as Analysis Services, Integration Services, Reporting Services, and SQL Server Compact Edition by selecting an option from the drop-down menu. Specify the server name or simply type **localhost** to connect to the local server in the Server name field. In the Authentication field, leave Windows Authentication selected and click Connect to connect to the SQL Server 2008 database engine.
18. When the SQL Server Management Studio loads, in the Object Explorer on the left, expand the Databases node. Note that the Campaign Database is also listed in the list of databases. Expand the Campaign Database node and then Tables

node by clicking the plus sign next to them. You will see the dbo.RawDataTxt table listed there.

19. Click New Query on the menu bar of Management Studio to open a query pane. Type the following in the query pane, highlight it, and press F5 to run it.

```
SELECT * FROM Campaign.dbo.RawDataTxt
```

You will see all the records imported into this table in the lower half of the pane.

Review

Now that you've used the SQL Server Import and Export Wizard to import data from a .CSV text file, try your hand at exporting the data from Campaign database to an Excel file by starting the SQL Server Import and Export Wizard from within the SQL Server Management Studio. To give you a hint, right-click the Campaign database and choose Export to start. Also, don't forget to save the package, as you will use it in the next Hands-On.

Using Business Intelligence Development Studio

One of the features of SSIS Import and Export Wizard is that the packages can be saved for later use and can be extended with BIDS. However, note that the packages cannot be saved while working with SQL Server Import and Export Wizard in the SQL Server Express Edition.

You have read about BIDS in Chapter 1 and have also created a blank Integration Services project. In the following Hands-On exercise, you will open this blank project and then add the packages you just created in the last Hands-On. Don't worry if you don't understand exactly how Integration Services works; the purpose of the following exercise is to give you a preview, a feel, of how an Integration Services package looks. All the features shown here are covered in detail in Chapter 3.

Hands-On: Exploring an SQL Server Import and Export Wizard Package Using BIDS

In this Hands-On exercise, you will open the various components of the Integration Services package and explore their properties and configurations. Though you will be learning about various attributes and properties of the components, the focus will be on learning how these components work together rather than understanding each component in detail.

Method

This Hands-On exercise is broken down into the following parts:

- You will add an existing package to an Integration Services blank project.
- You will then explore the components in the Control Flow and Data Flow Designer surfaces.

Exercise (Adding SSIS Import and Export Wizard Package in BIDS)

In this part, you will add the package Importing RawDataTxt.dtsx that you've created earlier using the SQL Server Import and Export Wizard, in My First SSIS Project that you created in Chapter 1.

1. Start the SQL Server Business Intelligent Development Studio.
2. Open My First SSIS Project from the Start Page | Recent Projects section. Alternatively, you can also open this package from the File | Open | Project/ Solution option and navigating to the C:\SSIS\ Projects\My First SSIS Project folder to select My First SSIS Project.sln. The blank solution you created in Chapter 1 will open up.
3. In Solution Explorer, right-click the SSIS Packages node and select Add Existing Package from the context menu. This will open the Add Copy Of Existing Package dialog box. Select the File System in the Package Location field. In the Package Path field, type **C:\SSIS\Packages\Importing RawDataTxt.dtsx**. Alternatively, you can use the ellipsis button (…) provided opposite to the Package Path field to locate and select this package. Click OK to add this package to the SSIS Packages node.
4. Though the package has been added to the SSIS Package node, it has not yet been loaded into the designer environment. Double-click the Importing RawDataTxt .dtsx package to load it into the designer environment. You will see the package loaded in BIDS, as shown in Figure 2-8.
5. Note that the Connection Managers tab, on the bottom half of the SSIS designer, shows two connection managers—DestinationConnectionOLEDB for choosing the Campaign database as a destination and SourceConnectionFlatFile for the source RawDataTxt.csv text file. Right-click SourceConnectionFlatFile and choose Edit from the context menu to edit this connection. You will see the Flat File Connection Manager Editor dialog box, which is similar to the window you filled in for the flat file while running the SSIS Import and Export Wizard. Explore the various tabs and click Cancel when you're done.
6. Right-click DestinationConnectionOLEDB and click Edit. In the Connection Manager dialog box, note that the layout is different, but the settings are exactly the same as those you chose in the Choose A Destination window while running the SQL Server Import and Export Wizard. Click Cancel.

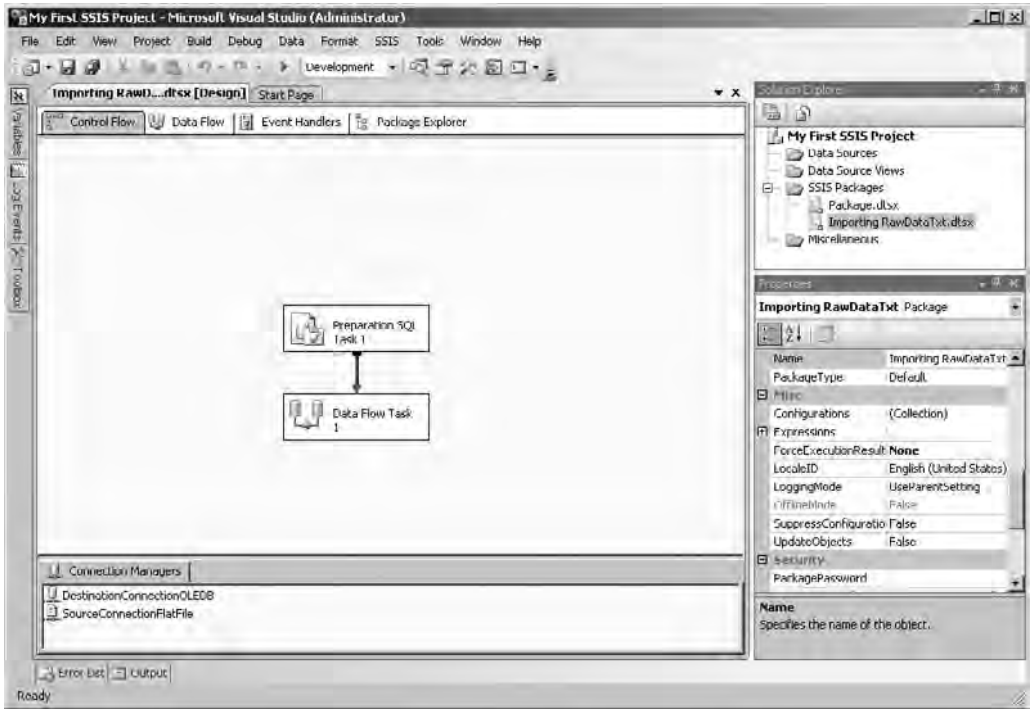


Figure 2-8 *Loading the Importing RawDataTxt.dtsx package in BIDS*

Exercise (Exploring Control Flow Components)

As you know, Integration Services has separate engines—control flow for managing workflow and data flow to manage the pipeline activities. The package Importing RawDataTxt.dtsx creates a table called RawDataTxt in the Campaign database while running in the control flow engine's scope. It uses Execute SQL task to perform this function. After it has created the table, it passes control over to the data flow engine to perform the data-related activities.

7. In the Control Flow tab, double-click Preparation SQL Task 1 to open the Execute SQL Task Editor dialog box, shown in Figure 2-9. You'll see various configuration options that have been set for this task.

   Note that under the SQL Statement section, the Connection field is pointing to the Campaign database using DestinationConnectionOLEDB Connection Manager. SQLSourceType is currently set to Direct Input, which gives you the option of typing an SQL statement. If you hover your mouse over the SQLStatement field, the SQL query will pop up. Clicking in this field enables an ellipsis button, which

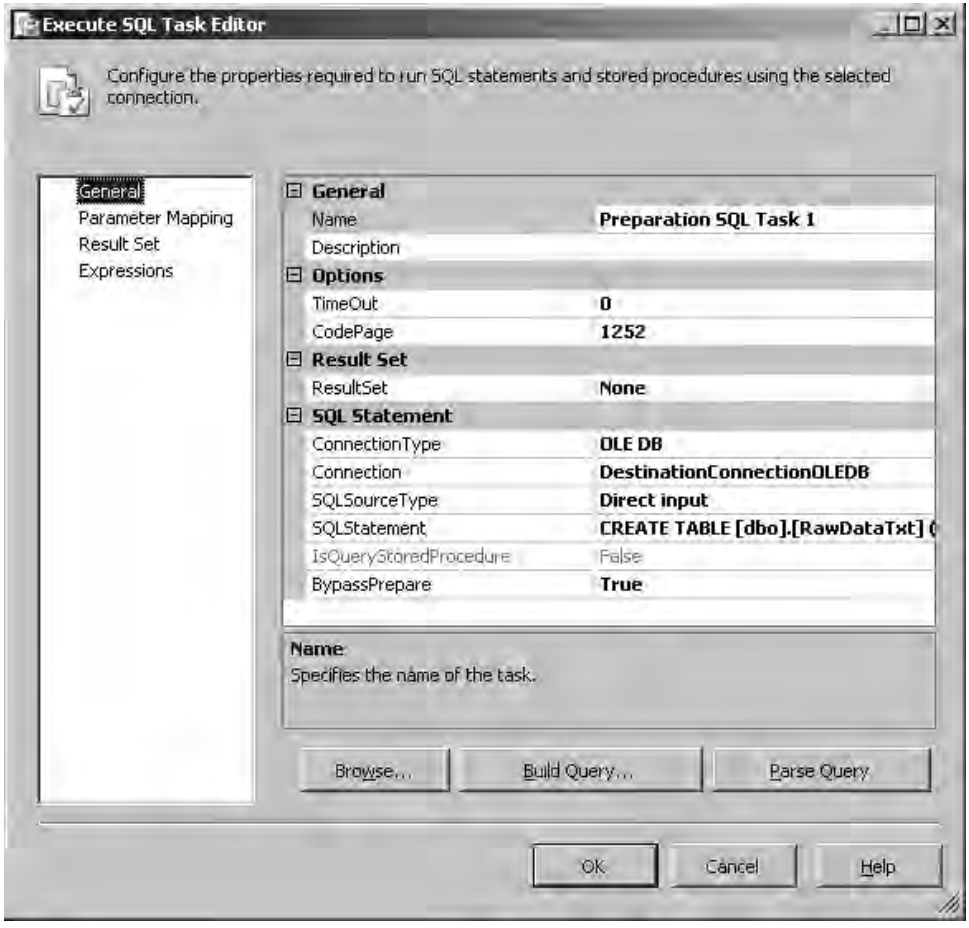


Figure 2-9 *Preparation SQL Task Settings*

you can click to open the Enter SQL Query window. In the Enter SQL Query window, you can type in a long query or paste in an existing one. Click Cancel to close the Enter SQL Query window. You can also configure this task to read an SQL statement from a file, which is available only when you've selected File Connection in the SQLSourceType field. You can also click Build Query to build simple queries and Parse Query to check the syntax of your queries. Click Cancel to undo any changes you have made and return to the SSIS Designer. The most important thing to understand at this time is that using this task, Integration Services package creates a table RawDataTxt in the Campaign database.

8. Double-click Data Flow Task 1 to see the Data Flow tab of the BIDS. The components that constitute data flow in a package reside here. The Data Flow tab will look similar to Figure 2-10.

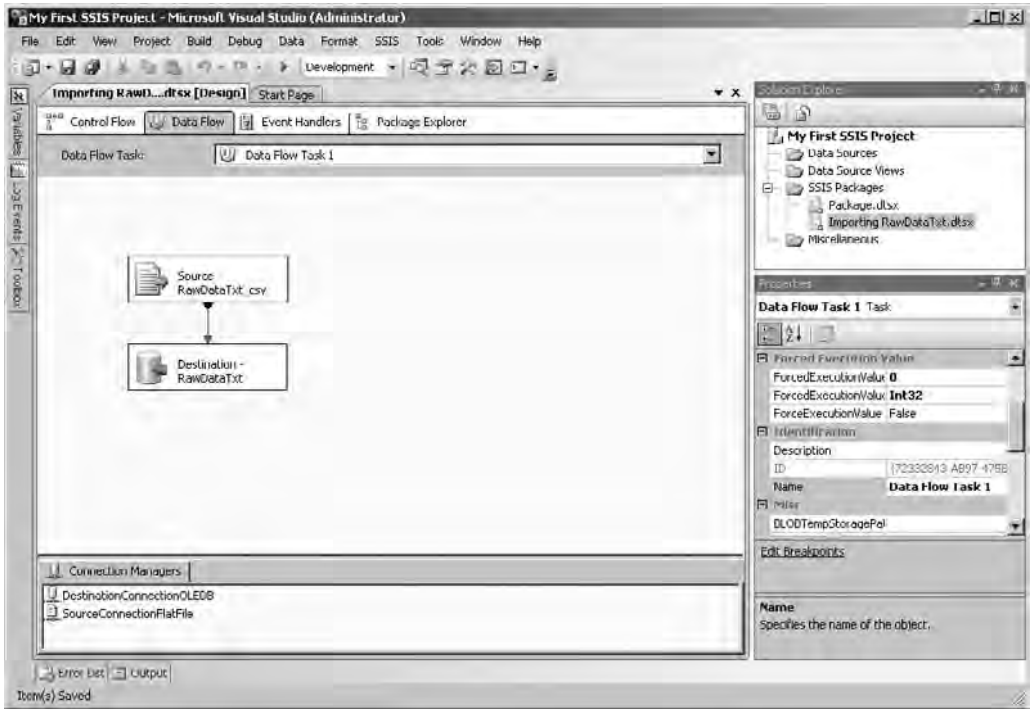


Figure 2-10 *Data Flow consists of a Source and a Destination*

The data flow engine is responsible for data extraction, manipulation and transformation, and then loading data into the destination. In this package, the source and destination are:

- **Source** RawDataTxt_csv, a Data Flow Source
- **Destination** RawDataTxt, a Data Flow Destination

This is a simple data loading package that doesn't have any transformation component. The Data Flow Source, Source—RawDataTxt_csv, extracts data from the flat file RawDataTxt.csv. It then passes on the data to the Destination—RawDataTxt that loads the extracted data into an SQL Server table, which was created earlier by an Execute SQL task in the Control Flow.

9. The first component, Source—RawDataTxt_csv in the Data Flow, is called a *Flat File Source,* which reads the data rows from the RawDataTxt.csv file row by row and forwards them to the downstream destination component.

   If you hover your mouse on the Toolbox, which is shown as a tabbed window on the left side of the screen in Figure 2-10, it will slide out to show you all the components that can be used in the data flow. Note that Flat File Source appears

in the Data Flow Sources section and the OLE DB Destination appears in the Data Flow Destinations section. There are several transformations components listed in the Data Flow Transformations section, though none is used here. This simple package shows a typical example of a data flow that contains a Data Flow Source and a Data Flow Destination.

Finally, the Data Flow Destination, Destination—RawDataTxt, is an OLE DB destination that loads the converted records to the RawDataTxt table in the Campaign database.

10. The data flow components of an Integration Services package expose their properties in a custom user interface that is built for most of the components, or in the Advanced Editor that is common to all the components. Some of the components do not have a custom user interface, so they use only the Advanced Editor to expose their properties and attributes. Sometimes you may have to use the Advanced Editor even though the component has a custom UI, as some components do not expose all the properties in the custom UI. You can open the custom user interface by choosing the Edit command from the component's context menu and the Advanced Editor using the Show Advanced Editor command. Right-click the Source—RawDataTxt_csv object and choose Show Advanced Editor from the context menu.
11. You will see four tabs in the Advanced Editor. The Connection Managers tab specifies SourceConnectionFlatFile connection manager that this component uses to connect to the RawDataTxt.csv flat file. The Connection Manager field here displays all the connection managers defined in the package.
12. Move on to the Component Properties tab. Here you will see the Common Properties that specify properties such as Name and Description, and Custom Properties sections.
13. Click the Column Mappings tab. In the upper half of this tab, you can see the columns mapped by the mapping lines and the lower half lists these mapped external columns with the output columns. External columns reference the data columns read from the source text file and the output columns are the columns this adapter passes on to the downstream data flow component. These output columns will become input columns for the next component in the data flow.
14. You can change these mappings if you want an External Column to be redirected to a different Output Column. Click the mapping line joining the AddressLine2 columns of Available External Columns and Available Output Columns and press the DELETE key on your keyboard. Similarly, delete the mapping line joining City columns. Now click and hold the mouse on the AddressLine2 column in the Available External Columns list and drag and drop it on the City column in the Available Output Columns list. You've created a mapping line to map AddressLine2 column to City column, which means the data in the AddressLine2 column will be sent to City column. This can also be done in the

lower half of the window. Click the column that shows <Ignore>, just below City, in the Output Column. The column is converted into a drop-down list box. Click the down arrow to see the list of available columns and choose AddressLine2 from the list. As you do that, a mapping line corresponding to the affected columns will be added in the upper section. Your mappings should look as shown in Figure 2-11.

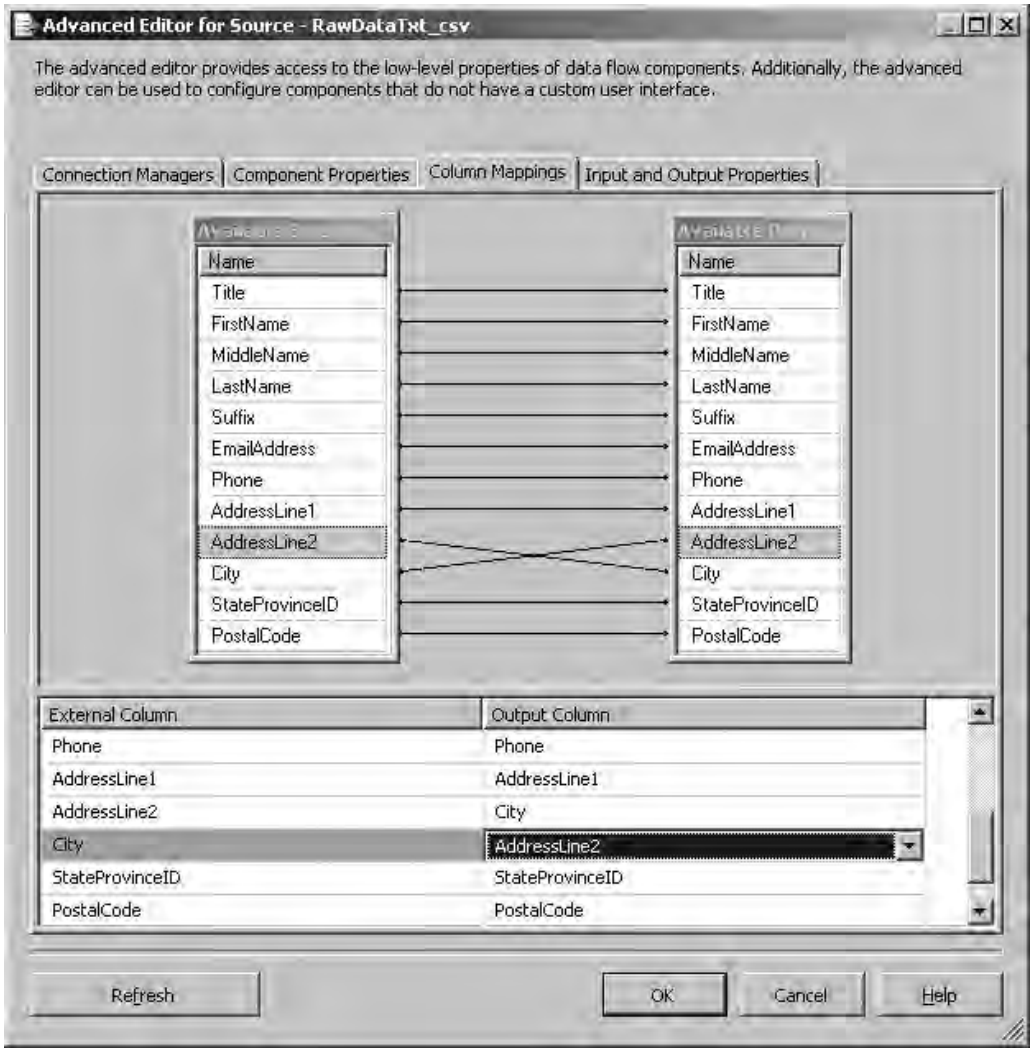


Figure 2-11 *Working with column mappings*

Now right-click anywhere on the blank surface in the upper half and choose Select All Mappings to select all the mapping lines. Again right-click and choose Delete Selected Mapping. This will remove all the mappings, and the Output Column in the lower half of the window shows <Ignore> in all the columns. Again, right-click anywhere in the upper section and choose Map Items By Matching Names. This will map all the corresponding columns together.

15. Open the Input And Output Properties tab, and you can see Flat File Source Output and Flat File Source Error Output under Inputs and Outputs. Expand the Flat File Source Output node to see External Columns and Output Columns. As mentioned earlier, External Columns are the reference columns of the source text file and Output Columns are the columns that Flat File Source Adapter passes on to the downstream component in the data flow path. Expand External Columns and click any column to see column properties such as CodePage, DataType, and Length in the right pane.

    Now expand Output Columns and click any of the columns; you will see the Output Column properties such as CodePage, DataType, Length, FastParse, SortKeyPosition, and so on. Note that the Data Type of External Columns and Output Columns is [DT_STR] by default. The FastParse option can be set to either True or False. To load data between heterogeneous data sources, the source adapters parse the data of each column to convert it to SSIS data type, and when the data is to be loaded into a data store, the destination adapter parses the data and converts it to the type destination requires.

    The two parsing techniques, Fast parse (when FastParse option is True) and Standard parse (when FastParse option is False), are available in the Flat File source and Flat File destination adapters and the Data Conversion and Derived Column transformations. This is because only these data flow components convert data from a string to a binary data type, or vice versa. The FastParse option allows use of simpler (commonly used date and time formats), quicker, but locale-insensitive, fast parsing routines. You can set FastParse to True on the columns that are not locale-sensitive to speed up the parsing process. By default, FastParse is set to False, indicating Standard parse is used, which supports all the data type conversions. For more information on parsing techniques, refer to Microsoft SQL Server 2008 Books Online.

    Click Cancel to return to the SSIS Designer.

16. An OLE DB Destination loads input records into an OLE DB–compliant data store. To explore its custom user interface, double-click the Destination—RawDataTxt component. You will see the Connection Manager page shown in Figure 2-12.

17. As the name suggests, an OLE DB Destination uses an OLE DB connection manager to connect to the destination table. In this package, DestinationConnectionOLEDB is used, which is specified in the OLE DB Connection Manager field.

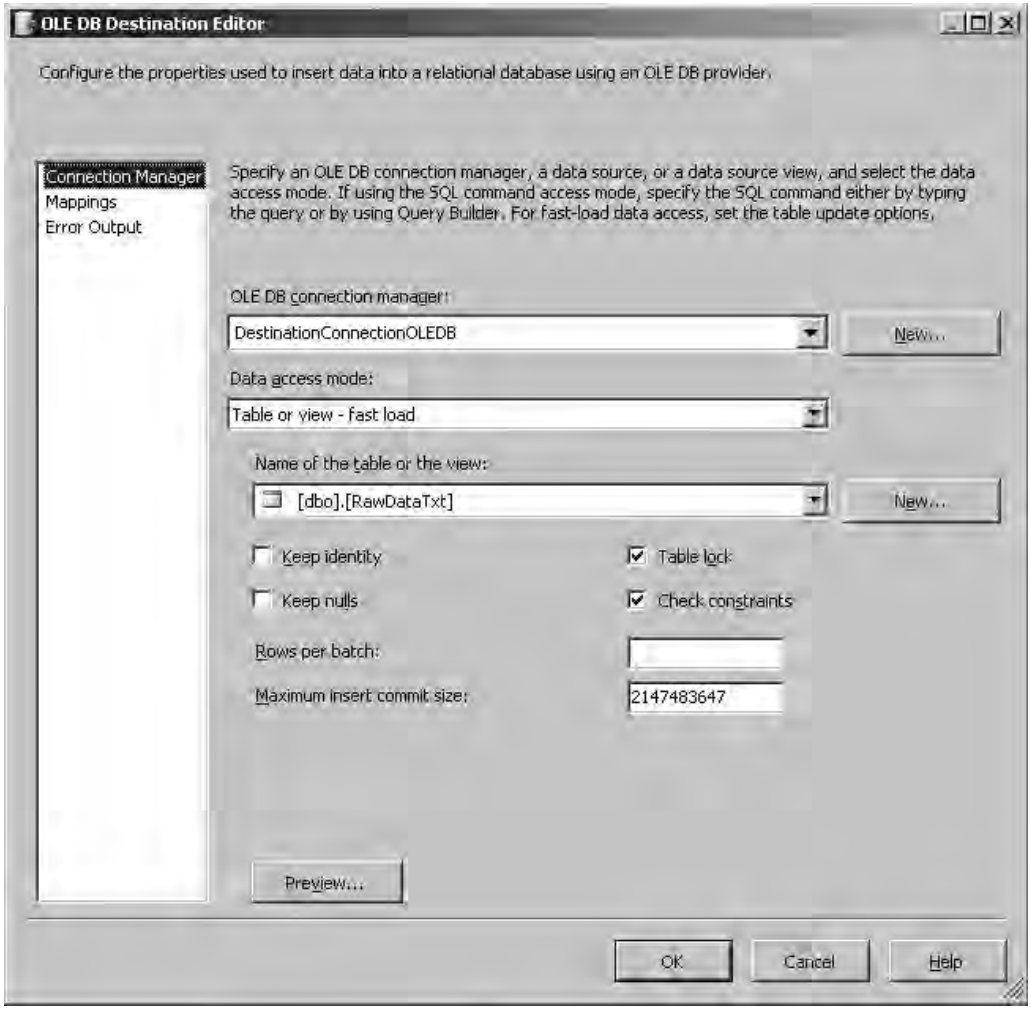


Figure 2-12 *Connection Manager page of the OLE DB Destination Editor*

18. Click in the Data Access Mode field, and you'll see the available five options for data access in the drop-down list:
    - **Table or view** When you select this option, the data is loaded into a table or view in the database specified by the OLE DB connection manager, and you specify the name of the table or the view in the Name of the table or the view field.
    - **Table or view – fast load** Using this data access mode, you can load data into a table or view as in the previous option, but using the fast load options such as acquiring table lock and specifying maximum insert commit size.

- **Table name or view name variable** Using this option, you still load data into a table or a view, but instead of specifying the table or view name directly, you specify a variable that contains the table or view name.
- **Table name or view name variable – fast load** This data access mode works like Table or view – fast load access mode, except in this access mode you supply the variable that contains the table or the view name.
- **SQL command** You can load the result set of an SQL statement using this option.

19. Go to the Mappings page and note that the Available Input Columns are mapped to Available Destination Columns. Note that the Suffix column in the input columns is not mapped to any destination column. You may have to scroll up and down to see the mappings properly. Click Cancel to close the editor.

Review

This Hands-On exercise presented a simple package that contains various types of components for you to see how Integration Services packages are organized. You've seen how Control Flow manages workflow in a package and makes the required objects available when they are required by Data Flow components. We haven't tried to execute this package because the package is not designed from a multiuse perspective. The Execute SQL task creates a table in a Campaign database for the first time the package is run, but what do you expect will happen if you again try to run the same package? The package will not succeed, as the table that Execute SQL task tries to create in the subsequent runs already exists, and the Execute SQL task attempt will fail, resulting in failure of the package. If you want to run the package more than once, you could either drop the table already created before trying to create it again or use TRUNCATE TABLE command with the existing table instead of creating a new table.

It will be worthwhile to review the package you have created during the last Hands-On when you exported data to an Excel file. I would encourage you to add it to this project and explore its various components to get a feel for them. Don't worry if they don't make much sense to you now, as each of the preconfigured components that SSIS provides will be covered in detail in the chapters to come.

Last, if you still want to play a little bit more with BIDS and SQL Server Import and Export Wizard, you can perform another Hands-On exercise using C:\SSIS\RawFiles\RawDataAccess.mdb file to build an Integration Services package directly in BIDS. To give you a hint, start the SQL Server Import and Export Wizard from Project menu command and note that this time the wizard doesn't give you an option to save the package as it has done in the previous exercises.

Integration Services Connections Project Wizard

Just as you can use the SQL Server Import and Export Wizard to create a basic package in BIDS that you can enhance later, Integration Services provides you another wizard to quickly create a package with all the required connection managers. It allows you to choose data providers and configure them to create connection managers. Once configured, you can them select to use a connection manager as a source or a destination or both. Finally, it creates a project with configured connection managers and a data flow task containing defined sources and destinations. This can be very helpful as creating connection managers is usually the first task when you're creating a package. You can invoke this wizard from File | New | Project and then choosing the Integration Services Connections Project Wizard from the Visual Studio project templates.

Analyzing Data Quality with the Data Profiling Task

During various stages of a data warehouse project, you'll need to make sure that the values in certain columns stay within the defined perimeters. To verify and implement this requirement, you may be required to run ad hoc queries such as distinct values, lengths of various values in the column, or percentage of null values against the data warehouse. And if you find deviations in data, you'll need to fix that either (optimally) in the ETL or using ad hoc queries. You might choose to apply constraints in the data warehouse to prevent deviations from happening; however, constraints bring their own problems, such as failures of ETL, increased loading time, and complex delete operations. It will be easier if you can quality control these issues at the loading stage and hence the data warehouse always receives the correct data. This will enable the data warehouse to perform better avoiding unnecessary ad hoc queries and changes. To explain it further, consider if business reports are using a two-digit country code column extensively, you'll need to make sure that this column always has the correct country code and doesn't include any stray values. You may check the nulls in the column, the length of country code values, or distinct values in the column as part of your resolution. If you implement these checks and their relative corrections while loading the data warehouse, you will have solved most of your data specifications–related problems upfront. Integration Services now includes a Data Profiling Task to facilitate the process of finding anomalies in data.

The Data Profiling Task connects to an SQL Server database table or view and creates various aggregate statistics to help you discover the problems in data. The Data Profiling Task enables you to compute statistics either on a single column or on multiple

columns or both. The column analysis brings out the true metadata of a column, as it is based on the data itself and helps you to understand column data in detail. The multiple column statistics give you an insight on how the values in one column depend upon the values in another. These configurations are called Profile Requests; five of them are available for individual column statistics, and three are available to analyze multiple columns or relationships between columns.

Single-Column Profiles

Single-column profiles enable you to analyze single column independently for Null values, column statistics, pattern profile, length distribution, and value distribution within the column.

- **Column Length Distribution Profile** You will perform this computation on a column containing text strings to identify any outliers. For example, if the column you are profiling contains fixed-length codes, any variation in length will indicate a problem in the data. This profile type computes all the distinct lengths of string values in the selected column and the percentage of rows in the table that each length represents.
- **Column Null Ratio Profile** You will perform this computation to find out missing data in a column with any data type. For example, an unexpectedly high ratio of null values in a column indicates the absence of data. This profile computes the percentage of null values in the selected column.
- **Column Pattern Profile** This profile request generates a set of regular expressions and the percentage of related string values. You will be using this profile to determine invalid strings in data. This profile can also suggest regular expressions that can be used in the future to validate new values.
- **Column Statistics Profile** This profile request works with numeric and datetime columns and can compute statistics for minimum and maximum values. Additionally, you can also generate statistics for average and standard deviation values for numeric columns. This profile can help you to identify values that lie outside the range you expect in a column or have a higher standard deviation than expected.
- **Column Value Distribution Profile** This profile will be of the most interest to you in case you want to know the distinct values and their percentage of rows in the column. This can help you understand your data a bit more, or if you already know the number of values, you can figure out the problems in data. This profile request works with most data types, such as numeric, string, and datetime formats.

Multiple-Column Profiles

Using multiple-column profile, you can profile a column based on the values existing in other columns such as candidate key profile, functional dependency profile, and the value inclusion profile.

- **Candidate Key Profile** This profile request can identify the uniqueness of a column or set of columns and hence can help you to determine whether the column or set of columns is appropriate to serve as a key for the selected table. You can also use this profile request to find duplicates in the potential key column.
- **Functional Dependency Profile** This profile request finds out the extent to which the values in one column are dependent on the values in another column or set of columns. Using this profile, you can validate the data in a column based on the other column.
- **Value Inclusion Profile** This profile request checks whether the values in a column also exist in another column. Using this profile, you can identify the dependency and can determine whether a column or set of columns is appropriate to serve as a foreign key between the selected tables.

You can choose one or more of these profile requests to create data profiles. Based on the profile requests, the Data Profiling Task first runs metadata queries against INFORMATION_SCHEMA.COLUMNS to find out the column names and their attributes, such as data type, character length, numeric precision and scale, null-ability, and collation name. Then it runs several queries to compute values such as SUM, COUNT, DISTINCT, and LEN. While computing all this, it keeps the calculations and the information in the temporary tables in the TEMPDB database and drops them later once it's done with all the computations.

As you can imagine from this, you need read/write and create table permissions on the TEMPDB database to be able to run the Data Profiling Task as it performs various activities. In the end, all that information is written in an XML format in a variable or an output file. You can review the data statistics using the Data Profile Viewer, a stand-alone utility provided with SQL Server 2008 for viewing and analyzing data profiles. Though you can review and analyze the profiles manually by inspecting the output file and decide whether to import the data from the profiled table, you can actually automate this decision making in the workflow of your package by checking the data statistics in the xml variable.

Hands-On: Using Data Profiling Task

In this Hands-On you will use Data Profiling Task to profile the data imported into RawDataTxt table and will use Data Profile Viewer utility to review the statistics generated.

1. Start BIDS and open My First SSIS Project. Double-click Package.dtsx to open this blank package if it is not open already.
2. From the Toolbox, drag and drop the Data Profiling Task on to the Control Flow surface.
3. Double-click the icon to open the Data Profiling Task Editor dialog box.
4. In the General Page, click in the DestinationType field and then expand the list by clicking the drop-down arrow. Note that you have two options to choose from, File or a variable, where you would like the output of this task to be written. Leave the FileConnection selected. Click in the Destination field and select <New File Connection…> from the drop-down list to open File Connection Manager Editor.
5. Choose Create File in the Usage type field and type **C:\SSIS\RawFiles\DataProfileFile.xml** in the File field.
6. Click Quick Profile to open the Single Table Quick Profile Form. Click New shown opposite to ADO.NET Connection to open the Connection Manager dialog box. Note that it limits you to using the SqlClient Data Provider, indicating that the Data Profile task can profile only SQL Server 2000 and above databases. Type your server name or type **localhost** in the Server Name field. Select the Campaign database in the Select box or enter a database name field. Click Test Connection to test the configuration. Click OK twice to come back to Single Table Quick Profile Form.
7. Select [dbo].[RawDataTxt] in the Table Or View field as shown in Figure 2-13. Click OK to create profile requests.
8. Go to the Profile Requests page and review the various profile requests and their options. Click OK to complete the Data Profiling Task configurations.
9. From the Debug menu, select Start Debugging or press F5 to run the package, or else press the respective button on the toolbar. Once the package completes execution, stop the package by pressing SHIFT-F5.
10. Navigate to C:\SSIS\RawFiles and verify that the DataProfileFile.xml file has been created. Click Start | All Programs | Microsoft SQL Server 2008 | Integration Services | Data Profile Viewer to start the viewer utility.
11. In the Data Profile Viewer dialog box, click Open and navigate to C:\SSIS\RawFiles and open DataProfileFile.xml. Review the different profiles to understand the way Data Profiling Task creates the profiles. For example, if you review the Candidate Key Profiles, you will see that EmailAddress column has been selected as the Key column with a Key Strength of 100%. Similarly, Figure 2-14 shows the Column Length Distribution Profiles for the PostalCode column.

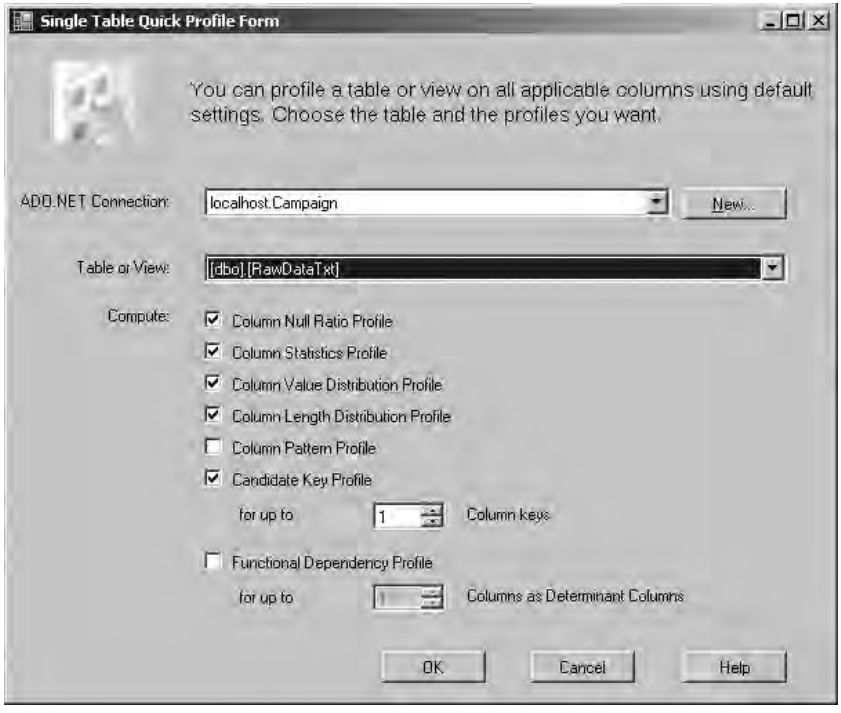


Figure 2-13 *Using the Quick Profile Form*

Review

In the preceding Hands-On you've worked with the Data Profiling Task and used the Quick Profiles option to create profile requests quite easily. However, you can configure the requests manually the way you want by clicking in the new row in Profile Requests page. Though the review of the profiles was manual, yet it provided you greater understanding of the profile structure created by the Data Profiling Task. If you really want to monitor data quality going forward, you will need to build business rules around data quality to create a scorecard matrix. To begin with, you can create a baseline of scorecards. When the data profiling activity is repeated over time, you can compare the scorecards generated each time against the baseline and deduce whether the data quality has improved or deteriorated over time.

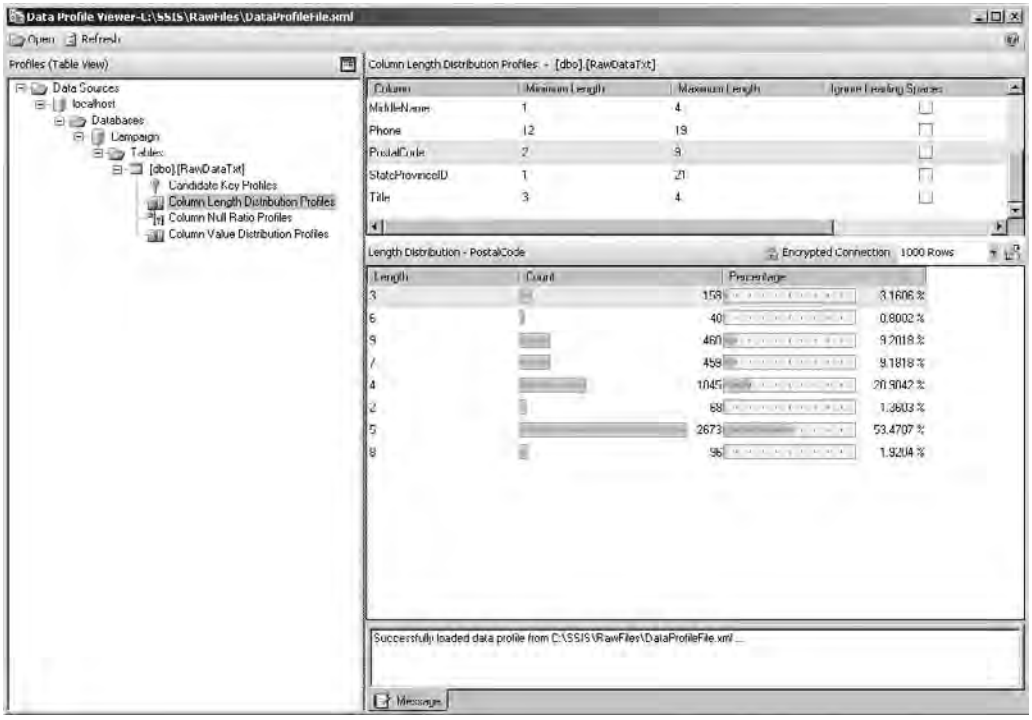


Figure 2-14 *Column Length Distribution Profiles*

Summary

You created an Integration Services blank project in Chapter 1. In this chapter, you created packages using the SQL Server Import and Export Wizard and then added those packages into your blank project. You also created a package directly in the BIDS again using the SQL Server Import and Export Wizard. But above all, you explored those packages by opening component properties and configurations, and now hopefully you better understand the constitution of an Integration Services package. Last, you worked with the Data Profiling Task to identify quality issues with your data. In the next chapter, you will learn about the basic components, the nuts and bolts of Integration Services packages, before jumping in to make complex packages in Chapter 4 using various preconfigured components provided in BIDS.

Chapter 3

Nuts and Bolts of the SSIS Workflow

In This Chapter

- Integration Services Objects
- Solutions and Projects
- File Formats
- Connection Managers
- Data Sources and Data Source Views
- SSIS Variables
- Precedence Constraints
- Integration Services Expressions
- Summary

So far, you have moved data using the SQL Server Import and Export Wizard and viewed the packages created by opening them in the Business Intelligence Development Studio (BIDS). In this chapter, you will extend your learning by understanding the nuts and bolts of Integration Services such as use of variables, connection managers, precedence constraints, and SSIS Expressions. If you have used Data Transformation Services (DTS 2000), you may grasp these issues quickly; however, there is a lot of new stuff about them in Integration Services. Usability and management of variables have been greatly enhanced, connectivity needs for packages are now satisfied by connection managers, enhanced precedence constraints have been included to provide you total control on the package workflow, and, above all, the SSIS Expression language offers a powerful programming interface to let you generate values at run time.

Integration Services Objects

Integration Services performs its operations with the help of various objects and components such as connection managers, sources, tasks and transformations, containers, event handlers, and destinations. All these components are threaded together to achieve the desired functionality—that is, they work hand in hand, yet they can be configured separately.

A major enhancement Microsoft has provided to DTS 2000 to make it Integration Services is the separation of workflow from the data flow. SSIS provides two different designer surfaces, which are effectively different integrated development environments (IDEs) for developing packages. You can design and configure workflow in the Control Flow Designer surface and the data movement and transformations in the Data Flow Designer surface. Different components have been provided in each of the designer environment, and the Toolbox window is unique with each environment.

The following objects are involved in an Integration Services package:

- **Integration Services package** The top-level object in the SSIS component hierarchy. All the work performed by SSIS tasks occurs within the context of a package.
- **Control flow** Helps to build the workflow in an ordered sequence using containers, tasks, and precedence constraints. Containers provide structure to the package and looping facility, tasks provide functionality, and precedence constraints build an ordered workflow by connecting containers, tasks, and other executables in an orderly fashion.
- **Data flow** Helps to build the data movement and transformations in a package using data adapters and transformations in ordered sequential paths.

- **Connection managers** Handle all the connectivity needs.
- **Integration Services variables** Help to reuse or pass values between objects and provide a facility to derive values dynamically at run time.
- **Integration Services event handlers** Help extend package functionality using events occurring at run time.
- **Integration Services log providers** Help in capturing information when log-enabled events occur at run time.

To enhance the learning experience while you are working with the SSIS components, first you will be introduced to the easier and more often-used objects, and later will be presented with the more complex configurations.

Solutions and Projects

Integration Services offers different environments for developing and managing your SSIS packages. The SSIS packages are designed and developed in, most likely, the development environment using BIDS, while the SQL Server Management Studio can be used to deploy, manage, and run packages, though there are other options to deploy and manage the packages as you will study in Chapter 13. Both environments have special features and toolsets to help you perform the jobs efficiently.

While BIDS has the whole toolset to develop and deploy SSIS packages, SQL Server Management Studio cannot be used to edit or design Integration Services solutions or projects. However, in both environments, you use *solutions* and *projects* to organize and manage your files and code in a logical, hierarchical manner. A *solution* is a container that allows you to bring together scattered projects so that you can organize and manage them as one unit. In general, you will use a solution to focus on one area of the business—such as one solution for accounts and a separate solution for marketing. However, complex business problems may require multiple solutions to achieve specific objectives. Figure 3-1 shows a solution that not only affects multiple projects but also includes projects of multiple types. This figure shows an analysis services project having a Sales cube, an integration services projects having two SSIS packages, and a reporting services project with a Monthly Sales report, all in one solution.

Within a solution, one or more projects, along with related files for databases, connections, scripts, and miscellaneous files, can be saved together. Not only can multiple projects be stored under one solution, but multiple *types* of projects can be stored under one solution. For example, while working in BIDS, you can store a data transformation project as well as a data-mining project under the same solution. Grouping multiple projects in one solution has several benefits such as reduced development time, code

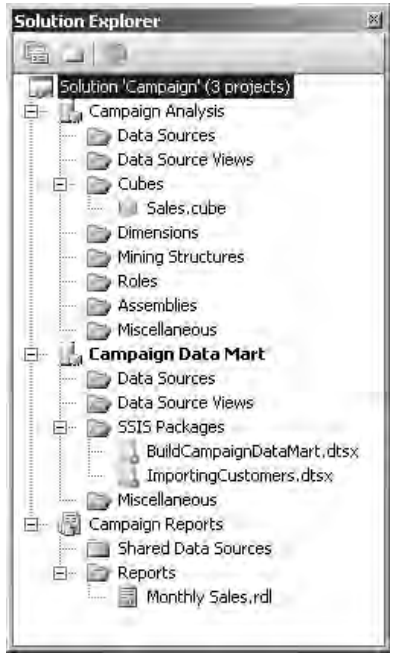


Figure 3-1 *Solution Explorer showing a solution with different types of projects*

reusability, interdependencies management, settings management for all the projects at a single location, and the facility to save all the projects to Visual SourceSafe or Team Foundation Server in the same hierarchical manner as you have in development environment. Both SQL Server Management Studio and BIDS provide templates for working with different types of projects. These templates provide appropriate environments—such as designer surfaces, scripts, connections, and so on—for each project with which you are working.

When you create a new project, Visual Studio tools automatically generate a solution for you while giving you an option to create a separate folder for the solution. If you don't choose to create a directory for the solution, then the solution file is created along with other project files in the same folder; however, if you choose to create a directory for the solution, then a folder is created with project folder created under this as a subfolder. So, you get a hierarchical structure created for you to which you can then add various projects—data sources, data source views, SSIS packages, scripts, miscellaneous files—as and when required. Solution Explorer lists the projects and the files contained in them in a tree view that helps you to manage the projects and the files (as shown

in Figure 3-1). The logical hierarchy reflected in the tree view of a solution does not necessarily relate to the physical storage of files and folders on the hard disk drive, however. Solution Explorer provides the facility to integrate with Visual SourceSafe or Team Foundation Server for version control, which is a great feature when you want to track changes or roll back code.

File Formats

Whenever an ETL tool has to integrate with legacy systems, mainframes, or any other proprietary database systems, the easiest way to transfer data between the systems is to use flat files. Integration Services can deal with flat files that are fixed width, delimited, and ragged right format types. For the benefit of users who are new to the ETL world, these formats are explained next.

Fixed Width

If you have been working with mainframes or legacy systems, you may be familiar with this format. Fixed-width files use different widths for columns, but the chosen width per column stays fixed for all the rows, regardless of the contents of those columns. If you open such a file, you will likely see lots of blank spaces between the two columns. As most of the data in a column with variable data tends to be smaller than the width provided, you'll see a lot of wasted space. As a result, these types of files are more likely to be larger in size than the other formats.

Delimited

The most common format used by most of the systems to exchange data with foreign systems, delimited files separate the columns using a delimiter such as a comma or tab and typically use a character combination (for example, a combination of carriage return plus linefeed characters—{CR}{LF}) to delimit rows/records. Generally, importing data using this format is quite easy, unless the delimiter used also appears in the data. For example, if users are allowed to enter data in a field, some users may use a comma while entering notes in the specified field, but this comma will be treated as column delimiter and will distort the whole row format. This free-format data entry conflicts with the delimiter and imports data in the wrong columns. Because of potential conflicts, you need to pay particular attention to the quality of data you are dealing with while choosing a delimiter. Delimited files are usually smaller in size compared to fixed-width files, as the free space is removed by the use of a delimiter.

Ragged Right

If you have a fixed-width file and one of the columns (the rightmost one) is a nonuniform column, and you want to save some space, you can add a delimiter (such as {CR}{LF}) at the end of the row and make it a ragged-right file. Ragged-right files are similar to fixed-width files except they use a delimiter to mark the *end* of a row/record—that is, in ragged-right files, the last column is of variable size. This makes the file easier to work with when displayed in Notepad or imported into an application. Also, some vendors use this type of format when they want the flexibility to change the number of columns in the file. In such situations, they keep all the regular columns (the columns that always exist) in the first part of the file and the columns that may or may not exist combined as a single string of data in the end of the row. Depending upon the columns that have been included the length of the last column will vary. The applications generally use substring logic to separate out the columns from the last variable-length combined column.

Connection Managers

As data grow in random places, it's the job of the information analyst to bring it all together to draw out pertinent information. The biggest problem of bringing together such data sets and merging them to a single storage location is how to handle different data sources, such as legacy mainframe systems, Oracle databases, flat files, Excel spreadsheets, Microsoft Access files, and so on. Connection managers provided in Integration Services come to the rescue.

In Chapter 2, you saw how the connection managers were used inside the package to import data. The components defined inside an Integration Services package require that physical connections be made to data stores during run time. The source adapter reads data from the data source and then passes it on to the data flow for transformations, while the destination adapter loads the transformed data to the destination store. Not only do the extraction and loading components require connections, but these connections are also required by some other components. For example, during the lookup, transformation values are read from a reference table to perform transformations based on the values in the lookup table. Then there are logging and auditing requirements that also need connections to storage systems such as databases or text files.

A connection manager is a logical representation of a connection. You use a connection manager to describe the connection properties at design time, and these are interpreted to make a physical connection at run time by Integration Services. For example, at design time, you can set a connection string property within a connection manager, which is then read by the Integration Services run-time engine to make a physical connection. A connection manager is stored in the package metadata and cannot be shared with other packages.

Connection managers enhance connection flexibility. Multiple connection managers of the same type can be created to meet the needs of Integration Services packages and enhance performance. For example, a package can use, say, five OLE DB connection managers, all built on the same data connection.

You can add connection managers to your package using one of the following methods in BIDS:

- Choose New Connection from the SSIS menu.
- Choose the New Connection command from the context menu that opens when you right-click the blank surface in the Connection Managers area.
- Add a connection manager from within the editor or advanced editor dialog boxes of some of the tasks, transformations, source adapters, and destination adapters that require connection to a data store.

The connection managers you add to the project at design time appear in the Connection Managers area in the BIDS designer surfaces, but they do not appear in the Connection Managers collection in Package Explorer until you run the package successfully for the first time. At run time, Integration Services resolves the settings of all the added connections, sets the connection manager properties to each of them, and then adds them to the Connection Managers collection in Package Explorer.

You will be using many of the connection managers in Hands-On exercises while you create solutions for business problems later on. For now, open BIDS, create a new blank project, and check out the properties of all the connection managers as you read through the following descriptions. Figure 3-2, which appears in the later section "Microsoft Connector 1.0 for SAP BI," shows the list of all the connection managers provided in SQL Server 2008 Integration Services.

ADO Connection Manager

The ADO Connection Manager enables a package to connect to an ADO recordset. This connection manager has been provided mainly for legacy support. You will most likely use it when you're working with a legacy application that is using ActiveX Data Objects (ADO) to connect to the data sources. You might have to use this connection manager when developing a custom component where such legacy application is used.

ADO.NET Connection Manager

The current model of software applications is very different from the earlier connected, tightly coupled client/server scenario, where a connection was held open for the lifetime. Now, you've varied types of data stores and these data stores are being hit with several

hundred connections every minute. ADO.NET overcomes these shortcomings and provides disconnected data access, integration with XML, optimized interaction with databases, and the ability to combine data from numerous data sources. These features make ADO.NET connection managers quite reliable and flexible with lots of options; however, they might be a little bit slower than the customized or dedicated connection managers for a particular source. You can also have consistent access to data sources using ADO.NET providers. The ADO.NET Connection Manager provides access to data sources, such as SQL Server or sources exposed through OLE DB or XML, using a .NET provider. You can choose from the .NET Framework Data Provider for SQL Server (SqlClient), the .NET Framework Data Provider for Oracle Server (OracleClient), the .NET Framework Data Provider for ODBC (Open Database Connectivity), and the .NET Framework Data Provider for OLE DB. The configuration options of the ADO.NET Connection Manager change, depending on the choice of .NET provider.

Cache Connection Manager

The Cache Connection Manager is primarily used for creating cache for the Lookup Transformation. When you have to repeatedly run a Lookup Transformation in a package or have to share the reference (lookup) data set among multiple packages, then you might prefer to persist this cache to a file to improve the performance. You would then use a cache transformation, which in turn uses the Cache Connection Manager to write the cached information to a cache file (.caw). Later in Chapter 10, "Data Flow Transformations," when you will be working with the Lookup Transformation, you will use this connection manager to cache data to a file.

Excel Connection Manager

This connection manager provides access to the Microsoft Excel workbook file. It is used when you add Excel Source or Excel Destination in your package. With the launch of Excel 2007, the data provider for Excel is changed to OLE DB provider for the Microsoft Office 12.0 Access Database Engine from the earlier used Microsoft Jet OLE DB Provider. If you check the ConnectionString property of the Excel Connection Manager after adding it using the Microsoft Excel 97-2003 version, you will see the Provider listed as Microsoft.Jet.OLEDB.4.0, whereas this property will show you the provider as Microsoft.ACE.OLEDB.12.0 when you add the Excel Connection Manager using Microsoft Excel 2007 version. It is important to understand the connection string, as you may need to write the connection string yourself in some packages, for example, if you're getting the file path at run time and you want to dynamically create the connection string. Here is the connection string shown for both versions of the Excel driver:

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\SSIS\RawFiles\
RawDataTxt.xls;Extended Properties="Excel 8.0;HDR=YES";

Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\SSIS\RawFiles\
RawDataTxt.xlsx;Extended Properties="Excel 12.0;HDR=YES";
```

Note the differences between the providers for both the versions as has been explained earlier. There are some additional properties that you need to specify in the extended properties section. The first is that you use Excel 8.0 for Excel versions 97, 2000, 2002, and 2003 in the extended properties, while you use Excel 12.0 for Excel 2007 version. Second, you use the HDR property to specify if the first row has column names. The default value is yes; that is, if you do not specify this property, the first row will be deemed to contain columns. Also, sometimes the Excel driver fails to pick up some values in the columns where you have string and numeric values mixed up. The Excel driver samples, by default the first eight rows, to determine the data type of the column and returns the null values if other data types exist in the column. You can override this behavior by importing all the values as strings using the import mode setting IMEX=1 in the extended properties of the connection string.

If you will be deploying this connection manager to a 64-bit server, which is most likely the case these days, you will need to run the package in 32-bit mode, as both the aforesaid providers are available in 32-bit version only. You will need to run the package using the 32-bit version of dtexec.exe from the 32-bit area, which is by default in the C:\Program Files(x86)\Microsoft SQL Server\100\DTS\Binn folder.

File Connection Manager

This connection manager enables you to reference a file or folder that already exists or is created at run time. While executing a package, Integration Services tasks and data flow components need input for values of property attributes to perform their functions. These input values can be directly configured by you within the component's properties, or they can be read from external sources such as files or variables. When you configure to get this input information from a file, you use the File Connection Manager. For example, the Execute SQL task executes an SQL statement, which can be directly input by you in the Execute SQL task, or this SQL statement can be read from a file.

You can use an existing file or folder, or you can create a file or a folder by using the File Connection Manager. However, you can reference only one file or folder. If you want to reference multiple files or folders, you must use a Multiple Files Connection Manager, described a bit later.

To configure this connection manager, choose from the four available options in the Usage Type field of the File Connection Manager Editor. Your choice in this field sets

the FileUsageType property of the connection manager to indicate how you want to use the File Connection Manager—that is, you want to create or use an existing file or a folder.

Flat File Connection Manager

This connection manager provides access to data in a flat file. It is used to extract data from a flat-file source or load data to a destination and can use delimited, fixed-width, or ragged-right format. This connection manager accesses only one file. If you want to reference multiple flat files, you must use a Multiple Flat Files Connection Manager.

FTP Connection Manager

Use this connection manager whenever you want to upload or download files using File Transfer Protocol (FTP). It enables you to connect to an FTP server using anonymous authentication or basic authentication. The default port used for FTP connection is 21. FTP Connection Manager can send and receive files using active or passive mode. The transfer mode is defined as active mode when the server initiates the FTP connection and passive mode when the client initiates the FTP connection.

HTTP Connection Manager

Whenever you want to upload or download files using HTTP (port 80), use this connection manager. It enables you to connect to a web server using HTTP. The Web Service task provided in Integration Services uses this connection manager. Like the FTP Connection Manager, the HTTP Connection Manager allows connections using anonymous authentication or basic authentication.

MSMQ Connection Manager

When you're working with mainframe systems or on systems with messaging architecture, you will need to use Message Queuing within your packages for which you will have to use an MSMQ Connection Manager. For example, if you want to use the Message Queue task in Integration Services, you need to add an MSMQ Connection Manager. An MSMQ Connection Manager enables a package to connect to a message queue.

Analysis Services Connection Manager

If you are creating an analysis services project or database as part of your solution, you may want to update or process Analysis Services objects as part of your SSIS jobs. One simple example could be that your SSIS packages update the data mart nightly, after

which you may want to process the cube and dimensions to include the latest data in the SSAS database. For such reasons as these, you may include the Analysis Services Connection Manager into your SSIS packages. This connection manager provides access to Analysis Services objects such as cube and dimensions by allowing you to connect to an Analysis Services database or an Analysis Services project in the same solution, though you can connect to an Analysis Services project only at design time. You will use the Analysis Services Connection Manager with the Analysis Services Processing task, Analysis Services Execute DDL task, or Data Mining Model Training destination objects in your package.

Multiple Files Connection Manager

When you have to connect to multiple files within your Script task or Script component scripts, you will use the Multiple Files Connection Manager. When you add this connection manager, you can add multiple files or folders to be referenced. Those multiple files and folders show up as a piped delimited list in the ConnectionString property of this connection manager. To specify multiple files or folders, you can also use wildcards. Suppose, for example, that you want to use all the text files in the C:\SSIS folder. You could add the Multiple Files Connection Manager by choosing only one file in the C:\SSIS folder, going to the Properties window of the connection manager, and setting the value of the ConnectionString property to C:\SSIS*.txt.

Similar to the File Connection Manager, the Multiple Files Connection Manager has a FileUsageType property to indicate the usage type—that is, how you want to create or use an existing file or a folder.

Multiple Flat Files Connection Manager

As you can reference only one flat file using the Flat File Connection Manager, you use the Multiple Flat Files Connection Manager when you need to reference more than one flat file. You can access data in flat files having delimited, fixed-width, or ragged-right format. In the GUI of this connection manager, you can select multiple files by using the Browse button and highlighting multiple files. These files are then listed as a piped delimited list in the connection manager. You can also use wildcards to specify multiple files. Suppose, for example, that you want to use all the flat files in the C:\SSIS folder. To do this, you would add C:\SSIS*.txt in the File Names field to choose multiple files. However, note that all these files must have the same format.

So, when you have multiple flat files to import from a folder, you have two options. One is to loop over the files using Foreach Loop Container, read the filenames and pass those filenames one by one to the Flat File Connection Manager so that the files can be imported iteratively. The second option is to use a Multiple Flat Files

Connection Manager where you don't need to use a looping construct; rather, this connection manager reads all the files, collates the data, and passes the data directly to the downstream components in a single iteration as if the data were coming from a single source such as a database table instead of multiple flat files.

Both these options have their usability in particular scenarios; for example, if you have to import several files from the same folder and you're not worried much about auditing and lineage—i.e., where the data is coming from, you can use the Multiple Flat Files Connection Manager method. This method bulk-imports the data quite quickly comparative to the looping construct of dealing with each file. The cost of speed is paid in terms of resource utilization. As all the files are read within the same batch, the CPU utilization and memory requirements are quite high in this case, although for a short duration, depending upon the file sizes. On the other hand, the iterative method deals with a file at a time, requiring less CPU and memory resources, but for a longer duration. Based on the file size, lineage, and auditing requirements, the resource availability on your server and the time window available to import data, you can choose one of these two methods to address the requirements.

ODBC Connection Manager

This connection manager enables an Integration Services package to connect to a wide range of relational database management systems (RDBMS) using the Open Database Connectivity (ODBC) protocol.

OLE DB Connection Manager

This connection manager enables an Integration Services package to connect to a data source using an OLE DB provider. OLE DB is an updated ODBC standard and is designed to be faster, more efficient, and more stable than ODBC; it is an open specification for accessing several kinds of data. Many of the Integration Services tasks and data flow components use the OLE DB Connection Manager. For example, the OLE DB source adapter and OLE DB destination adapter use OLE DB Connection Manager to extract and load data, and one of the connections that the Execute SQL task uses is the OLE DB Connection Manager to connect to an SQL Server database to run queries.

SMO Connection Manager

SQL Management Objects (SMO) is a collection of objects that can be programmed to manage SQL Server. SMO is an upgrade to SQL-DMO, a set of APIs you use to create and manage SQL Server database objects. SMO performs better, is more scalable, and is easy to use compared to SQL-DMO. SMO Connection Manager enables an

Integration Services package to connect to an SMO server and hence enable you to manage SQL Server objects using SMO scripts. For example, Integration Services transfer tasks use an SMO connection to transfer objects from one server to another.

SMTP Connection Manager

An SMTP Connection Manager enables an Integration Services package to connect to a Simple Mail Transfer Protocol (SMTP) server. For example, when you want to send an e-mail notification from a package, you can use Send Mail Task and configure it to use SMTP Connection Manager to connect to an SMTP server.

SQL Server Compact Edition Connection Manager

When you need to connect to an SQL Server Compact database, you will use an SQL Server Compact Connection Manager. SQL Server Compact Destination adapter uses this connection to load data into a table in an SQL Server Compact Edition database. If you're running the package that uses this connection manager on a 64-bit server, you will need to run it in 32-bit mode, as the SQL Server Compact Edition provider is available in a 32-bit version.

WMI Connection Manager

Windows Management Instrumentation (WMI) enables you to access management information in enterprise systems such as networks, computers, managed devices, and other managed components using the Web-Based Enterprise Management (WBEM) standard. Using a WMI Connection Manager, your Integration Services package can manage and automate administrative tasks in an enterprise environment.

Microsoft Connector 1.0 for SAP BI

You can import and export data between Integration Services and SAP BI by using Microsoft Connector 1.0 for SAP BI. Using this connector in Integration Services, you can integrate a non-SAP data source with SAP BI or can use SAP BI as a data source in your data integration application. The Microsoft Connector for SAP BI is a set of managed components that transfers data from and to an SAP NetWeaver BI version 7 system in both Full and Delta modes via standard interfaces. This connector is not installed in the default installation; rather it is an add-in to Integration Services and you have to download the installation files separately from the Microsoft SQL Server 2008 Feature Pack download web page. The SAP BI Connector can be installed on an Enterprise or a Developer Edition of SQL Server 2008 Integration Services; however,

you can transfer data between SAP BI 7.0 and any of the versions from SQL Server 2000 and later. The SAP BI connector provides three main components:

- SAP BI Source
- SAP BI Destination
- SAP BI Connection Manager

As you can guess, SAP BI Source can be used to extract data from an SAP BI system, SAP BI Destination can be used to load data into an SAP BI system and the SAP BI Connection Manager helps to manage the RFC connection between the Integration Services package and SAP BI. When you install the SAP BI connector, the SAP BI Connection Manager is displayed in the list of connection managers; however, you will need to add the SAP BI Source and SAP BI Destination manually. You can do this by right-clicking the Data Flow Sources in the Toolbox, selecting the Choose Items option, and selecting SAP BI Source from the list in the SSIS Data Flow Items tab. Similarly, you can add the SAP BI Destination by right-clicking the Data Flow Destinations in the Toolbox. Figure 3-2 shows the SAPBI Connection Manager in the Add SSIS Connection Manager dialog box, the SAP BI Source in Data Flow Sources section, and the SAP BI Destination in the Data Flow Destinations section of the Toolbox.

Microsoft Connector for Oracle by Attunity

Microsoft Oracle and Teradata connectors are developed by Attunity and have been implemented in the same fashion as the SAP BI connector. That is, when you install these connectors, you get a connection manager, a Source component, and a Destination component, though you will have to manually add source and destination components in to the Data Flow Designer Toolbox. Refer to Figure 3-2 to see how these components have been implemented. The Oracle connector has been developed to achieve optimal performance when transferring data from or to an Oracle database using Integration Services. The connector is implemented as a set of managed components and is available for Enterprise and Developer Editions of SQL Server 2008 Integration Services only. The Attunity Oracle Connector supports Oracle 9.2.0.4 and higher-version databases and requires Oracle client software version 10.*x* or 11.*x* be installed on the same computer where SSIS will be using this connector. With this connector, you can:

- **Fast Load** Bulk Load Destination using OCI (Oracle Call Interface) Direct Path.
- **Arrayed Load** Bulk Load Destination in batches and the entire batch is inserted under the same transaction.
- **Bulk Extract Source** Using OCI Array Binding.

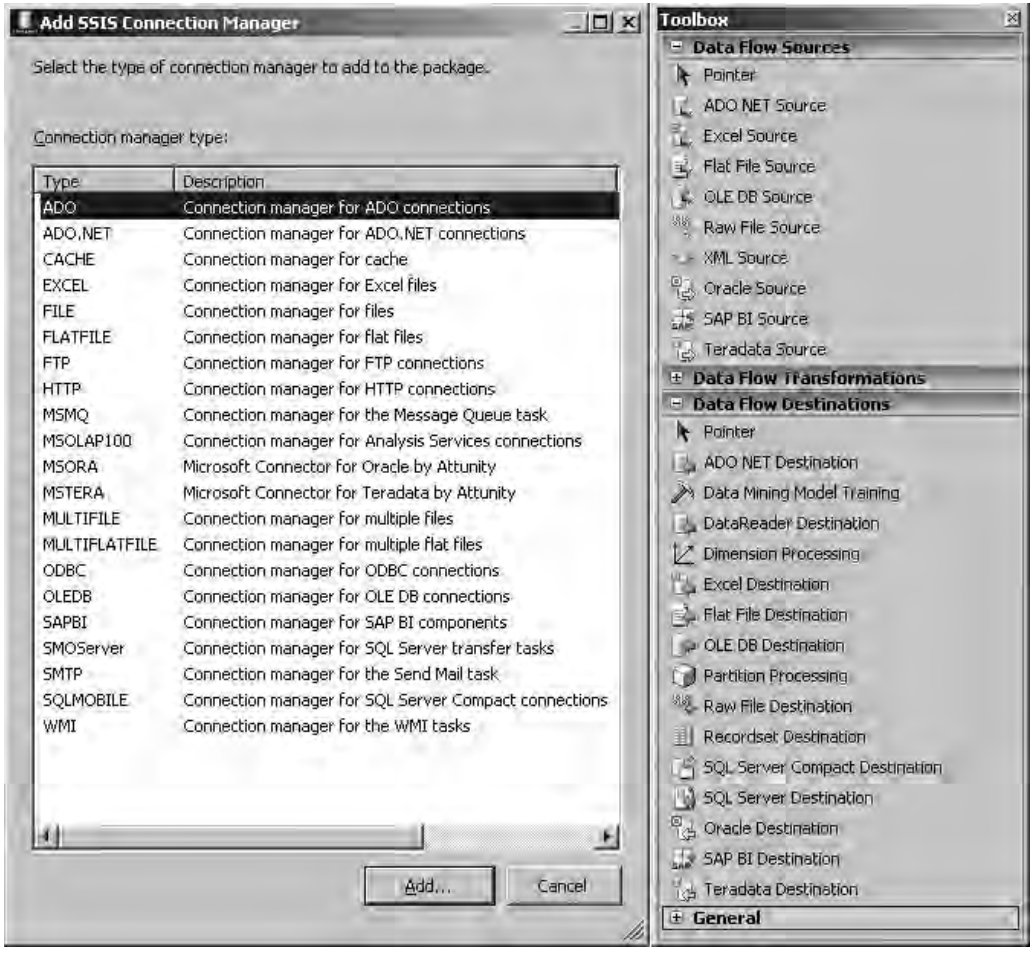


Figure 3-2 *SSIS connection managers and data flow sources and destinations*

Microsoft Connector for Teradata by Attunity

The Microsoft Connector for Teradata is a set of managed components developed to achieve optimal performance for transferring data from or to a Teradata database using Integration Services. The connector is available for the Enterprise and Developer Editions of SQL Server 2008 Integration Services only. The SSIS components for Teradata—i.e., Teradata Source, Teradata Destination, and Teradata Connection

Manager (see Figure 3-2) use the Teradata Parallel Connector (TPC) for connectivity. The Microsoft Connector for Teradata supports

- Teradata Database version 2R6.0
- Teradata Database version 2R6.1
- Teradata Database version 2R6.2
- Teradata Database version 12.0

To use this connector, you will have to install Teradata Parallel Transporter (TPT) version 12.0 and the Teradata ODBC driver (version 12 recommended) on the same computer where SSIS will be using this connector. You can use this connector for

- Bulk Load Destination using TPT FastLoad
- Incremental Load Destination using TPT Tpump
- Bulk Extract Source using TPT

Data Sources and Data Source Views

We have talked about connection managers that can be added in the packages. However, you might have noticed two folders, Data Sources and Data Source Views, in your project in Solution Explorer. These folders can also contain data source connections. However, these are only design-time objects and aren't available at run time. The connection managers embedded in the packages are used at run time.

Data Sources

You can create design-time data source objects in Integration Services, Analysis Services, and Reporting Services projects in BIDS. A *data source* is a connection to a data store—for example, a database. You can create a data source by right-clicking the Data Sources node and selecting the New Data Source option. This will start the Data Source Wizard that will help you create a data source. So, the data source object gets created outside the package and you reference it later in the package. Once a data source is created, it can be referenced by multiple packages. You can reference a data source in a package by right-clicking in the Connection Managers area and selecting the New Connection from Data Source option from the context menu.

When you reference a data source inside a package, it is added as a connection manager connection and is used at run time. This approach of having data source created outside a package and then referencing it or embedding it in the package as

a connection manager has several benefits. You can provide a consistent approach in your packages to make managing connections easier. You can update all the connection managers used in various packages that reference a data source by simply making a change at one place only—in the data source itself, as the data source provides synchronization between itself and the connection managers. Last, you can delete a data source any time without affecting the connection managers in the packages. This is possible because there is no dependency between the two. Connection managers don't need data sources to be able to work, as they are complete in themselves. The only link between a data source and the connection managers that reference it is that the connection managers get synchronized at times or when the changes occur. The data sources and the data source views are only design-time objects that help in management of the connection managers across several packages. During run time, the package doesn't need a data source to be present, as it uses connection managers that gets embedded in it anyway. Data sources are not used when building packages programmatically.

Data Source View

A data source view, built on a data source, is a named, saved subset that defines the underlying schema of a relational data source. A data source view can include metadata that can define sources, destinations, and lookup tables for SSIS tasks, transformations, and data adapters. While a data source is a connection to a data store, the data source views are used to reference more specific objects such as tables or views or their subsets. As you can apply filters on a data source view, you can in fact create multiple data source view objects from a data source. For example, a data source can reference a database, while different data source views can be created to reference its different tables or views. To use a data source view in a package, you must first add the data source to the package.

Using data source views can be beneficial. While you can use a data source view in multiple packages, refreshing a data source view reflects the changes in its underlying data sources. Data source views can also cache metadata of the data sources on which they are built and can extend a data source view by adding calculated columns, new relationships, and so on. You can consider this as an additional abstraction layer provided to you for polishing the data model or aligning the metadata as per your package requirements. This can be a very powerful facility in case you're dealing with third-party databases or working with systems where it is not easy for you to make a change.

The data source view can be referenced by data flow components such as OLE DB source and lookup transformations. To reference a data source view, you instantiate the data source and then refer the data source view in the component. Figure 3-3 shows an OLE DB source referencing a CampaignZone1 data source view, where

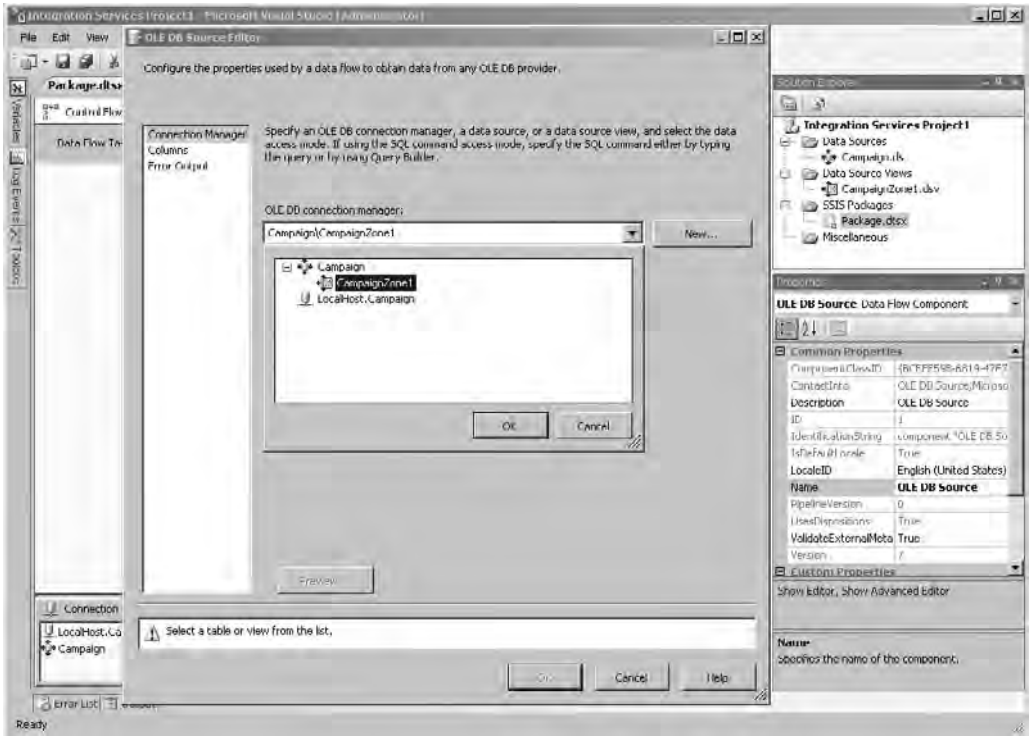


Figure 3-3 *Referencing a data source view inside an OLE DB source*

Campaign is a data source. Once you add a data source view to a package, it is resolved to an SQL statement and stored in a property of the component using it. You create a data source view by using the Data Source View Wizard and then modify it in the Data Source View Designer. Data source views are not used when building packages programmatically.

SSIS Variables

Variables are used to store values. They enable SSIS objects to communicate among each other in the package as well as between parent and child packages at run time. You can use variables in a variety of ways—for example, you can load results of an Execute SQL task to a variable, change the way a package works by dynamically updating its parameters at run time using variables, control looping within a package by using a loaded variable, raise an error when a variable is altered, use them in scripts, or evaluate them as an expression.

DTS 2000 provides global variables, for which users set the values in a single area in the package and then use those values over and over. This allows users to extend the dynamic abilities of packages. As the global variables are defined at the package level, sometimes managing all the variables at a single place becomes quite challenging for complex packages. SSIS has improved on this shortcoming by assigning a *scope* to the variables. Scopes are discussed in greater detail a bit later in the chapter in the section "User-Defined Variables."

Integration Services provides two types of variables—system variables and user-defined variables—that you can configure and use in your packages. System variables are made available in the package and provide environmental information or the state of the system at run time. You don't have to create the system variables, as they are provided for you, and hence you can use them in your packages straightaway. However, you must *create* a user-defined variable before you can use it in your package. To see the variables available in a package in BIDS, either go to the Variables window or go to the Package Explorer tab and expand the Variables folder.

System Variables

The preconfigured variables provided in Integration Services are called *system variables.* While you create user-defined variables to meet the needs of your packages, you cannot create additional system variables. They are read-only; however, you can configure them to raise an event when they change their value. System variables store informative values about the packages and their objects, which can be used in expressions to customize packages, containers, tasks, and event handlers. Different containers have different system variables available to them. For example, PackageID is available in the package scope, whereas TaskID is available in the Data Flow Task scope. Some of the more frequently used system variables for different containers are defined in Table 3-1.

Using these system variables, you can actually extract interesting information from the packages on the fly. For example, at run time using system variables, you can log who started which package at what time. This is exactly what you are going to do in the following Hands-On exercise.

Hands-On: Using System Variables to Create Custom Logs

This exercise demonstrates how you can create a custom log for an Integration Services package.

System Variable	Assigned to	Description
CancelEvent	Package	When set to a nonzero value, the task stops running
CreationDate	Package	Date when the package was created
CreatorName	Package	Name of the person who built this package
MachineName	Package	Name of the computer on which the package runs
PackageID	Package	Unique identifier of the package
PackageName	Package	Name of the package
StartTime	Package	Time that the package started to run
UserName	Package	Account of the user who started the package
VersionBuild	Package	Package version
VersionComment	Package	Comments about the package version
TaskID	Tasks	Unique identifier of a task instance
TaskName	Tasks	Name of the task instance
TaskTransactionOption	Tasks	Transaction option the task uses
ErrorCode	Event handlers	Error identifier
ErrorDescription	Event handlers	Description of the error
PercentComplete	Event handlers	Percentage of completed work
SourceDescription	Event handlers	Description of the executable in the event handler that raised the event
SourceID	Event handlers	Unique identifier of the executable in the event handler that raised the event
SourceName	Event handlers	Name of the executable in the event handler that raised the event
VariableDescription	Event handlers	Variable description
VariableID	Event handlers	Unique identifier of the variable

Table 3-1 *Partial List of System Variables Available in Various Containers*

Method

You will create a simple package consisting of one task to demonstrate how you can use system variables to create a custom log, which is desirable if standard logging doesn't meet your purposes.

You will create a table in the Campaign database to host logging information and will then use the Execute SQL task to read values of system variables for logging to the earlier created table. You will also add a connection manager to connect to the Campaign database.

Exercise (Adding a Connection Manager)

We will start with creation of CustomLog table and creating an SSIS package in this part.

1. Start SQL Server Management Studio and connect to the database engine of your local server. In the Object Explorer, expand the Databases folder and click Campaign Database. You have created the Campaign database in the last chapter, but if you haven't, you can attach the Campaign database provided with the software for this book. See the Appendix for more details. Click the New Query button located on the Standard toolbar to open a new query pane. Type the following query in this pane to create a table CustomLog in the Campaign database.

   ```
   USE Campaign
   CREATE TABLE CustomLog(
   username varchar(50),
   machinename varchar(50),
   packagename varchar(50),
   packageID varchar(50),
   taskname varchar(50),
   starttime datetime)
   ```

   Click Execute on the Toolbar to run the query and create a CustomLog table in the Campaign database. Leave SQL Server Management Studio running as you will return later to see the results.
2. Start BI Development Studio. Choose File | New | Project, or press CTRL-SHIFT-N to open the New Project window.
3. Choose Integration Services Project from the Templates pane. In the Name field, enter **Use of System Variables**. Choose the location C:\SSIS\ Projects, as shown in Figure 3-4. Verify that the Create directory for Solution check box is not selected. Click OK to create a new project.
4. In Solution Explorer, right-click package.dtsx under the SSIS packages folder; then choose Rename. Type **CustomLog.dtsx** and press ENTER. You'll see a message asking whether you want to rename the package object as well. Click Yes.
5. Locate the Toolbox window, by default, positioned on top-left side of the screen. Hover your mouse over the Toolbox tab to open it. From the Toolbox window, drag and drop the Execute SQL task onto the Control Flow Panel. The Execute SQL task will have a crossed red circle on it. Hovering the mouse on this crossed red circle will show the error message, "No connection manager is specified."
6. Now create a connection manager for this task. Right-click anywhere in the Connection Managers' tab and choose New OLE DB Connection from the context menu. This will open the Configure OLE DB Connection Manager dialog box. Click New to open a Connection Manager dialog box. You will find that the Native OLE DB\SQL Server Native Client 10.0 has already been added in the Provider drop-down box. Click the down arrow to look at the other available OLE DB providers. Click Cancel to return. Type **localhost** in the

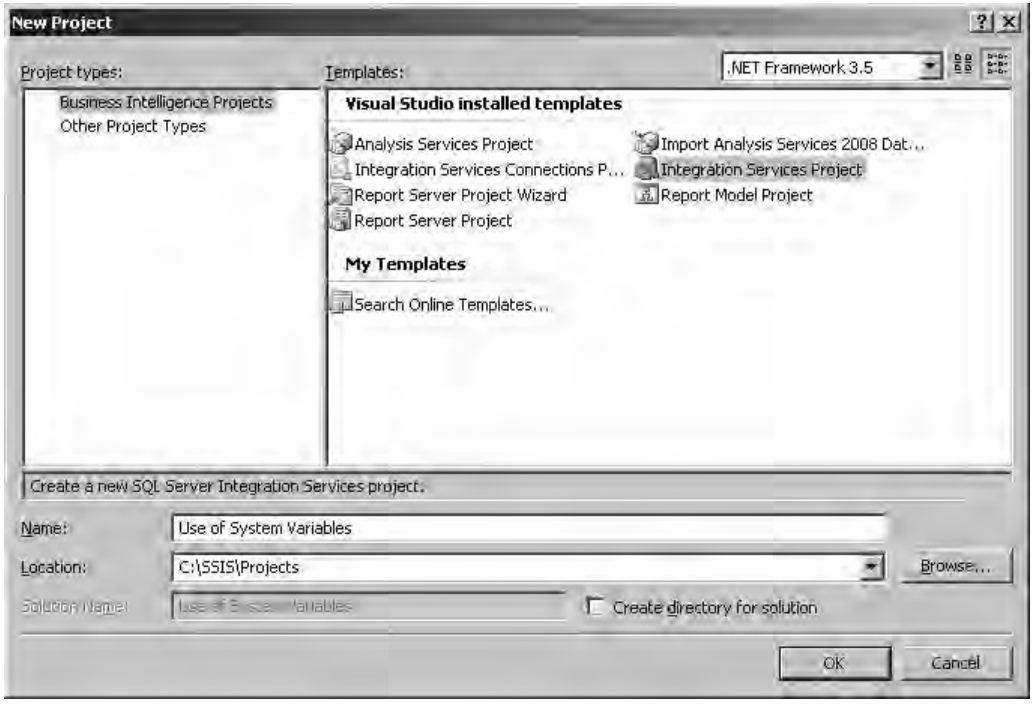


Figure 3-4 *Creating a new SSIS project for Use of System Variables*

Server Name field. Leave Use Windows Authentication in the Log On To The Server area. In the Connect To A Database area, make sure the "Select or enter a database name" radio button is selected. Click the down arrow and select the Campaign database from the drop-down list, as shown in Figure 3-5.

7. Click Test Connection to test your connection. You will see a message saying "Test connection succeeded." Click OK three times to return to the Control Flow Panel.

Exercise (Configuring Execute SQL Task)

Now is the time to configure the Execute SQL Task. As this is a very versatile task and is probably the most commonly used task, you will use this task in several Hands-On exercises in this book to learn about different uses and configurations that can be applied to this task.

8. Right-click the Execute SQL Task and choose Edit from the context menu to open the Execute SQL Task Editor window. In the SQL Statement group, click in the Connection box and then the down arrow that appears in the far-right corner to choose from the available connections. Notice that you can create a new connection from here as well. Choose localhost.Campaign from the list.

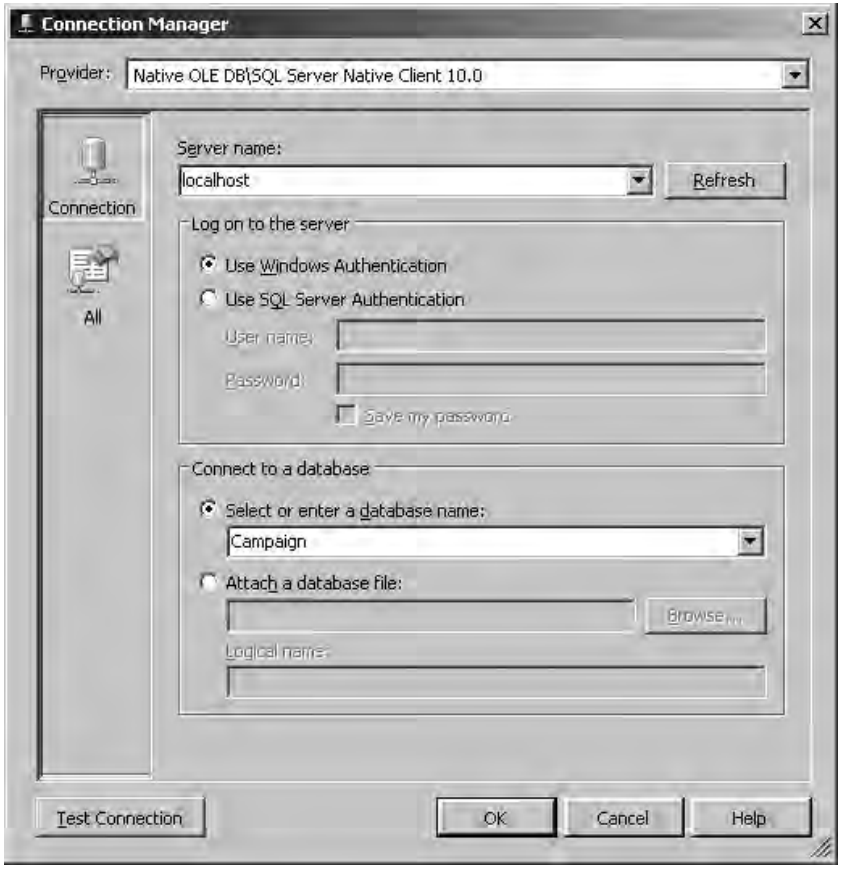


Figure 3-5 *Adding an OLE DB Connection Manager for Campaign database*

9. Click in the SQLStatement field, and then on the ellipsis button to open the Enter SQL Query window. Type in the following query and then click OK.

   INSERT INTO CustomLog VALUES (?,?,?,?,?,?)

   This is a parameterized query in which the question marks represent the parameters. You need to be careful about the order of your parameters while defining them, as they will be considered in the order they are added.

10. Go to Parameter Mapping page to define parameters for your query. Click Add. You will see a system variable being added in the dialog box. You need to change it to the one you need. Click the new variable that has just been added. As you click, the box will turn into a drop-down box. Click the down arrow and choose System::UserName from the list. Similarly, choose VARCHAR in the Data Type column and assign a value of 0 (zero) in the Parameter Name field, as shown in Figure 3-6.

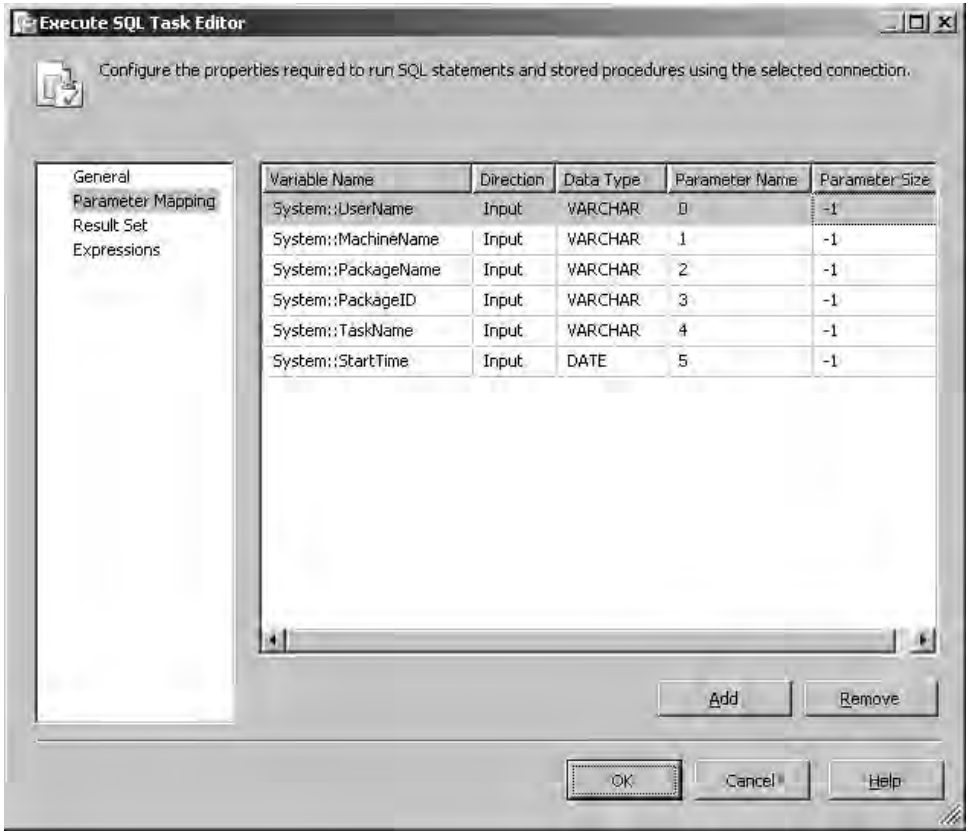


Figure 3-6 *Configuring Parameter Mapping page for Execute SQL task*

Now add five more variables by clicking Add. Change the Variable Name, Data Type, and Parameter Name values as per the following list. Click OK to close the window once all the parameters have been defined.

Variable Name	Direction	Data Type	Parameter Name
System::UserName	Input	VARCHAR	0
System::MachineName	Input	VARCHAR	1
System::PackageName	Input	VARCHAR	2
System::PackageID	Input	VARCHAR	3
System::TaskName	Input	VARCHAR	4
System::StartTime	Input	DATE	5

Exercise (Executing the Package to Populate CustomLog Table)

Finally, let us execute the package and populate CustomLog table.

11. Choose Debug | Start Debugging to run the package in the debugging mode. The Execute SQL task will turn yellow and then green. Click the stop debugging button on the debugging toolbar and press CTRL-SHIFT-S to save all the files in the project.
12. Switch back to SQL Server Management Studio. In the query window (open a query window if this is not already open), execute the following query: **SELECT * FROM CustomLog**.

You will see a record entered in the CustomLog table with all the details configured for logging.

Review

In this exercise, you read values of system variables at the point in time when the package was run and loaded those values to a table in SQL Server. SSIS provides extensive logging facilities using log providers (discussed in Chapter 8) that enable you to log in various formats—for example, log to text files or to an SQL Server table. In instances when the standard logging doesn't fit your requirements, you can use custom logging, as you've done in this Hands-On exercise. You've used Execute SQL task to run a parameterized query. If you are wondering about Execute SQL task, it will be discussed many times throughout this book and you will understand all the aspects of its usage. This is one of the main tasks provided in the Control Flow and will be covered in detail in Chapter 5 when we discuss all the preconfigured tasks provided in the Control Flow.

User-Defined Variables

You can configure user-defined variables for packages, Foreach Loop Containers, For Loop Containers, Sequence Containers, tasks, and event handlers. To configure a user-defined variable, you will provide a name, description, namespace, scope, and value for the variable along with choosing other properties, such as whether the variable raises an event when its value changes or whether the variable is read-only or read/write.

Before you begin creating variables, go through the following concepts to understand their creation and usage.

Namespace

SSIS keeps its variables under *namespaces*; by default, custom variables are in the User namespace, and system variables are in the System namespace. You can define the

custom variables in the User namespace or can create an additional namespace for them, but you cannot modify or add the System namespace or its variables.

Scope

As mentioned earlier, Integration Services allows variables to be created in the scope of a package or within the scope of a container, task, or event handler in the package. This helps in managing variables, as they are created at the location where they are used. When you create a variable, the scope is selected by the context—i.e., based on your selection of the container or task while you're creating a variable. You cannot change the scope of a variable after it has been created. The only way to change the scope of a variable is to delete and recreate it in the scope you want. To communicate among objects within a package and to flow the variable values from a parent package to a child package, variables follow these rules:

- *Variables defined at the container scope are available to all the tasks or the child containers within the container.* For example, all the tasks or containers within the Foreach Loop container can use the variables defined within the scope of the Foreach Loop container. Variables defined at the package scope can be used like global variables of DTS 2000, as the package is also a container and is at the top of the container hierarchy. However, for complex packages, it is recommended that you create the variables within the scope where they are needed to help manageability.
- *If a variable has the same name as the variable defined at its parent or grandparent, the "local" variable will supersede the one descending from the ancestor container.* Note that the variables are case-sensitive; therefore, this rule will not affect variables with the same name but with different cased letters.
- *Variables defined in the scope of the called package are never available to the calling package.* SSIS provides a facility to run a package as a child package inside a parent package using the Execute Package task. When a package is run as a child package using the Execute Package task, the variables defined within the Execute Package task scope are available to the called package. That is, the Execute Package task can pass the variables to the child packages. The process of flow of variables from parent package to child package works at run time—that is, the child package variables get values from the parent package when run in the parent package process. If the child package is run on its own, the variable won't get updated due to the parent not being available.

Data Type and Value

While configuring a variable, you first need to choose the data type the variable is going to be. The value and the data type must be compatible with each other—for example, for an integer data type, you cannot use a string as the value. You can assign a literal or an expression to the value.

Evaluating a variable using an expression is a powerful feature that allows you to use dynamic values for variables. Each time the package tasks want to use the variable, they have to evaluate the expression to calculate the variable value. A common use of such a scenario is to load data from the update files received daily, having the date as part of the filename. In this case, you would need to write an expression using the GETDATE() function to evaluate the variable value. Then the expression is evaluated at run time and the variable value is set to the expression result.

You can also change variable values using package configurations, which provide the ability for updating values of properties at package loading time as well as package run time. Configurations are useful when deploying packages, as you can update values of variables and connection strings of connection managers. You will study more about configurations in Chapter 13.

Hands-On: Creating a Directory with User-Defined Variables

Many times, while creating a workflow for SSIS packages, you will need to create folders into which you'll copy files. In this exercise, you will create a folder using a user-defined variable; this will demonstrate how SSIS can use the values of variables at run time.

Method

The File System task provided in the Control Flow tab of BIDS does file and folder operations and can create a folder as one of its operations. You will be using this task to create a C:\SSIS\FSTdirectory folder. This exercise is built in two parts: In the first part, you will use static (that is, hard-coded) values in the File System task to create the C:\SSIS\FSTdirectory folder. In the second part, you will use a variable to perform the same task. While the use of variables is extensively covered in Hands-On exercises used for package developments throughout this book, here you will do a simple exercise to add a variable and see how it functions. The File System task is relatively simple to use and configure. This task is covered in detail in Chapter 5, but here the focus is on the use of variables.

Exercise (Using Hard-Coded Values)

In the first part of this Hands-On exercise, you will create a folder using File System task and directly specifying the folder path and name in the task itself.

1. Start BIDS and choose File | New | Project to create a new project, or press CTRL-SHIFT-N. The New Project window opens.
2. In the New Project window, choose Integration Services Project from the Templates area. In the Name field type **Creating Directory** and in the Location field type **C:\SSIS\Projects**. Click OK to create this new project.
3. After the new project is created, go to the Solution Explorer window and right-click package.dtsx under the SSIS Packages folder. Choose Rename from the context menu. Rename the package as **Creating Directory.dtsx** and click Yes in the confirmation box.
4. While in the Control Flow designer, open the Toolbox window and drag and drop the File System Task onto the Control Flow designer. You will see a crossed red circle on the task. Hover your mouse on it and it will display a message indicating a validation error. These validation errors are a big help in debugging and creating a syntactically correct package.
5. Double-click the icon of the File System task to open File System Task Editor. With this task, you can create, move, delete, rename, and set attributes on files and directories by selecting one of these operations in the Operation field. Click in the Operation field and select Create Directory from the drop-down list. Selecting this option changes other available fields. This is one of the great features of Integration Services components—to change the available fields dynamically based on the value selected in another field. This way, you can configure only the relevant information in the component rather than getting confused with loads of options.
6. Click in the SourceConnection field to provide the path for the folder and select <New connection…> from the drop-down list. This will open the File Connection Manager Editor. In the previous Hands-On for system variables, you created a connection manager by right-clicking in the Connection Managers area; here you are going to create a connection manager from within a task. Remember that the File Connection Manager allows you to either refer to an existing file or folder or create a new file or a folder. Don't get confused by the word *directory,* which is used in File System Task, and *folder,* which is used in File Connection Manager. They have the same meaning.
7. In the Usage type field, choose Create Folder from the drop-down list. In the Folder field, type **C:\SSIS\FSTdirectory**. Click OK to return to the File System Task Editor. Note that *FSTdirectory* has been added in the Connection Managers area and also appears in the SourceConnection field (see Figure 3-7). Click OK to close the File System Task Editor.

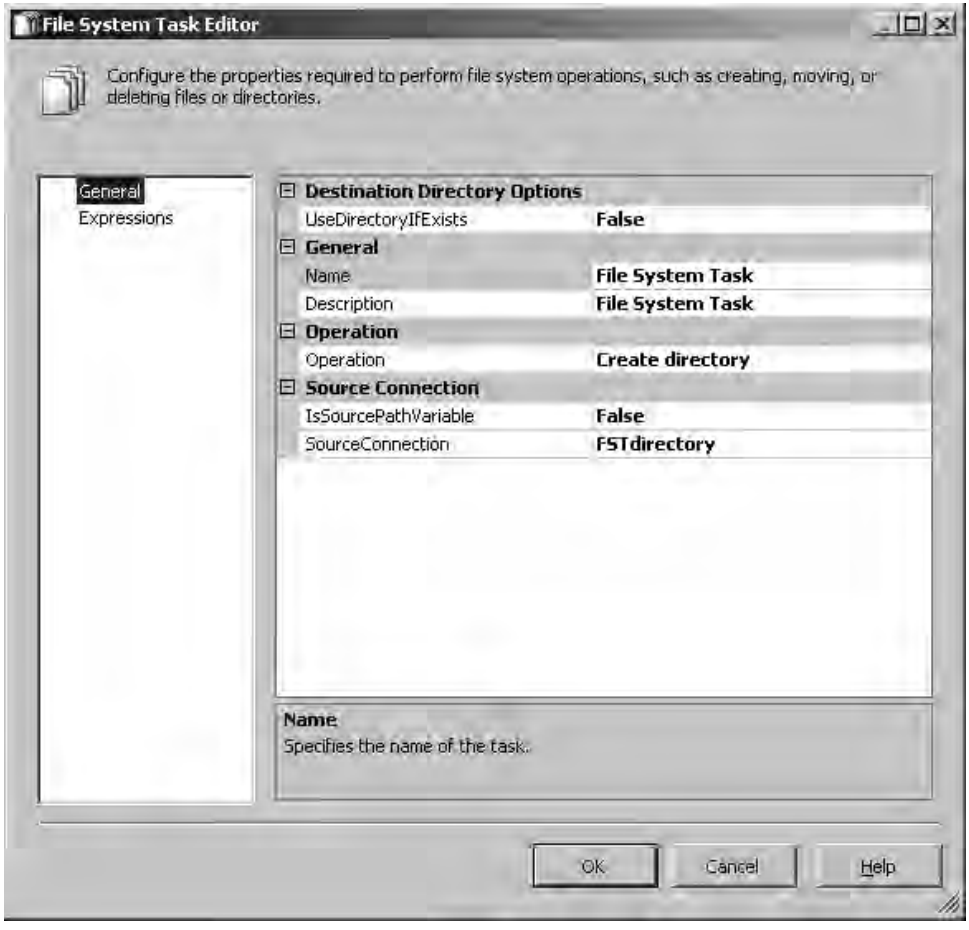


Figure 3-7 *Configuring the File System task with hard-coded values*

8. Choose Debug | Start Debugging or press F5 to execute the package you've developed. The File System task will quickly change to green and will show a "Package Execution Completed" message. The output window also pops up below the Connection Managers area and will show a success message. Press SHIFT-F5 to stop debugging the package and return to design mode.
9. Run Windows Explorer to see that C:\SSIS\FSTdirectory has been created.

Exercise (Using a User-Defined Variable to Pass the Folder Path to File System Task)

In this second part of the Hands-On, you will create a user-defined variable and assign a value to it. Then you will assign this variable to SourceConnection field in the File

System task and will see that the variable value is used by the task to create the required folder.

10. Delete the newly created folder FSTdirectory form the C:\SSIS folder and the connection manager from BIDS by simply selecting it and pressing DELETE. Click Yes on the pop-up dialog box to confirm deletion of the connection manager.
11. Right-click anywhere on the blank surface of the Control Flow Panel and choose Variables from the context menu. This will open the Variables window. Add a variable by clicking the Add Variable button, which is the leftmost button on the menu bar in the Variables window. In the Name column, type **directory** and change the Data Type to **String**. In the Value field, type **C:\ SSIS\ FSTdirectory**, as shown in the Figure 3-8.
12. Double-click the File System task to open the Task Editor. In the IsSourcePathVariable field's drop-down list, select True. Note that this converts the SourceConnection field to a SourceVariable field.
13. Click in the SourceVariable field and select the User::directory variable from the list of variables. Click OK to close the File System Task Editor as shown in Figure 3-9.
14. Press F5 to execute the package. The File System task turns green and the Output window shows the Success message.
15. Press SHIFT-F5 to stop debugging the package, or choose Debug | Stop Debugging. Switch to Windows Explorer to see that the folder has been created.
16. In BIDS, choose File | Save All and leave the project open, as you will return to this project later in the chapter.

Review

You have created a user-defined variable and used its value to populate the SourceVariable field in the File System Task Editor. Because you specified the IsSourcePathVariable as True, so when you run this package, the File System task knows that it has to read the value of the SourceVariable field from the User::directory variable. Though you have defined a static value to the variable, this is a good example of how a variable can affect the task. In real-life scenarios, the variables are populated either by package configurations or by the upstream tasks in the package at run time and the downstream tasks use those variables to get dynamically updated values.

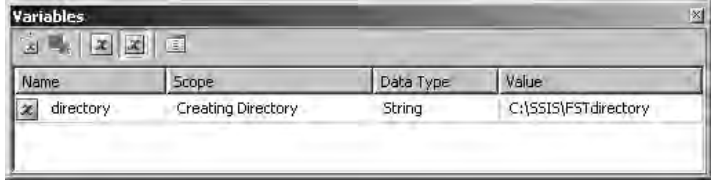


Figure 3-8 *Adding a variable to the package*

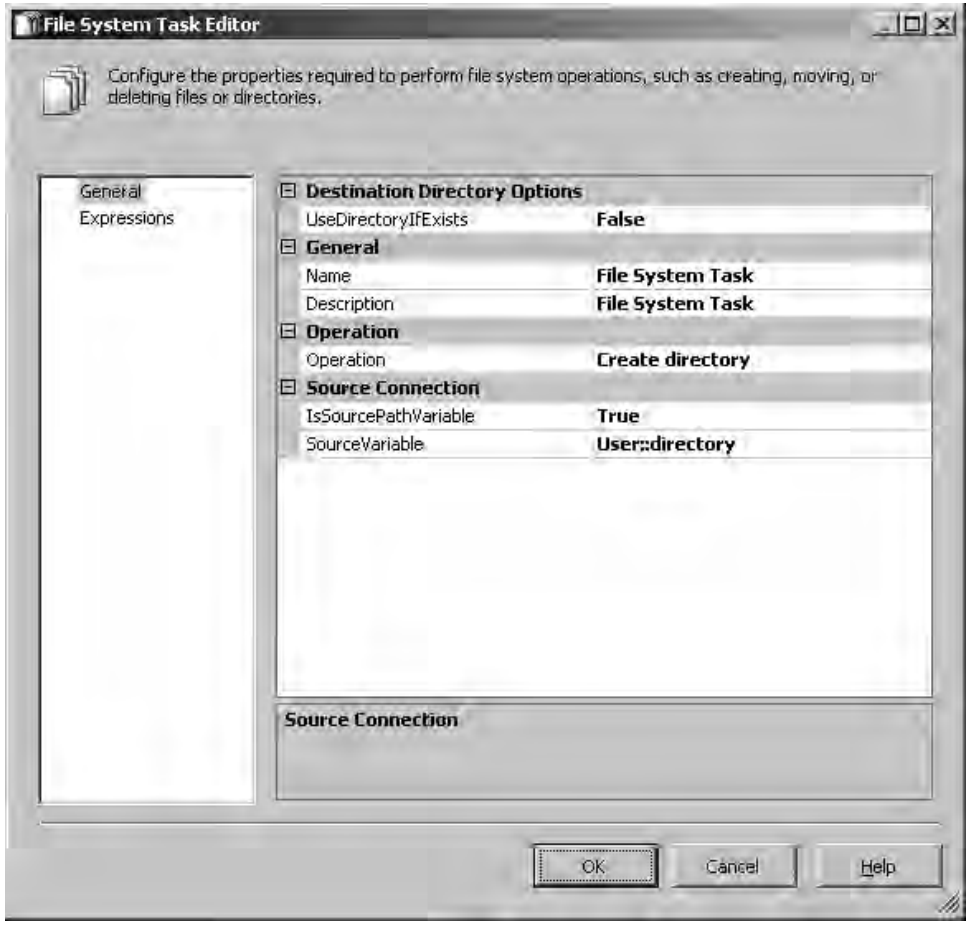


Figure 3-9 *Configuring the File System Task Editor to get a value from a user-defined variable*

Precedence Constraints

While creating the workflow and linking tasks, you can decide the conditions that can affect the running of successive tasks. To do this, you use *precedence constraints* when linking two or more executables. Within the SSIS world, an *executable* is defined as a container, a task, or an event handler. The executable for which you define the precedence constraint to be applied after it has run is called the *precedence executable,* and the executable for which you define a precedence constraint to be applied before it could run is called the *constrained executable.* The precedence constraints can be affected by the execution result of the precedence executable. For example, you may wish to stop

the constrained executable from running if the precedence executable fails to complete. The precedence constraints created in a package can also be seen in the Package Explorer tab in BIDS.

When you connect two tasks on the Control Flow Panel by dragging the green arrow from a task on to the other task, you actually use a precedence constraint. You can open the Precedence Constraint Editor by double-clicking the green and can specify various options for configuring the constraints. The Editor window is divided into two parts: Constraint Options and Multiple Constraints.

Constraint Options

The precedence constraint can be based on a combination of the execution results and the evaluation of the expressions. In the Constraint Options section, you can select the execution result and provide an expression to be evaluated.

- **Evaluation Operation** This defines the EvalOp property of the constraint. You have four choices here that you can see when you click the down arrow button. The selection of the option here determines the way the precedence constraint combines execution results and evaluation expressions to determine whether to run the constrained executable.
  - **Constraint** Use only the execution result from the precedence executable to determine whether the constrained executable runs.
  - **Expression** Use only the Expression property to determine whether the constrained executable runs. When selected, the Expression field becomes available to enable you to define an expression, which must be a valid SSIS expression.
  - **Expression And Constraint** Combines the requirements of both the Constraint and the Expression.
  - **Expression Or Constraint** Uses either of the two options to decide the execution of the constrained executable.
- **Value** This field refers to the value property of the precedence constraint and specifies the value of the execution result of the precedence executable to determine whether to run the constrained executable. You have three options available here:
  - **Completion** The constrained executable will run on the completion of the precedence executable with no regard to the outcome—that is, success or failure. For example, you may want to run a consistency check on your database toward the end of your nightly update operation without regard to whether the data uploading task successfully loaded data or failed to load any data.

- **Success** The constrained executable will run only when the precedence executable has completed successfully or has a result of "No Errors Occurred" from the preceding task. The No Errors Occurred result is returned by a disabled task in the control flow to let the work flow continue.
- **Failure** The constrained executable will run only when the precedence executable has failed. For example, you may want to alter the data flow when the data loading task fails.

- **Expression** In this field, you can specify an expression that evaluates to Boolean true to run the constrained executable. This Expression field becomes available only when you choose the Expression and Constraint or Expression Or Constraint option in the Evaluation Operation field. The expression can include functions, operators, and system and custom variables. You can click Test to evaluate the expression or test whether the expression you have specified is syntactically correct. A simple expression is shown in Figure 3-10.

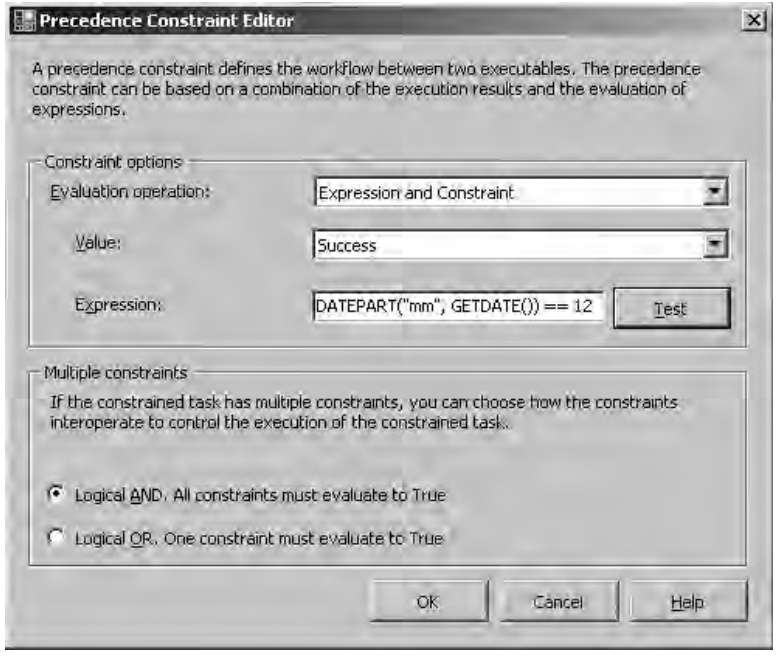


Figure 3-10 *The Precedence Constraint Editor*

Multiple Constraints

A constrained executable can have multiple constraints—that is, more than one precedence executable may apply precedence constraints on a constrained executable. In situations for which you have multiple constraints on a constrained executable, you can specify the criteria for multiple constraints to control the execution of the constrained executable in the Multiple Constraints section. You can choose one of the two mutually exclusive options—Logical AND or Logical OR. Selecting Logical AND sets the LogicalAnd property of the precedence constraints to True and implies that *all* the constraints must evaluate to True before running the constrained executable. Selecting Logical OR sets the LogicalAnd property to False and implies that *one* of the constraints must evaluate to True to run the constrained executable.

The precedence constraint behavior is covered in a Hands-On exercise in Chapter 5, where you will be able to see how the constrained executables are affected by your choice of a particular constraint.

Integration Services Expressions

Integration Services allows you to set task values dynamically using variables that are updated at run time by other tasks. This facility is quite powerful and great in many respects; however, it does not solve all the real-life problems associated with population of configuration values. Developers must often evaluate a value for a task based on a value generated by another task. This requires that they write a custom piece of code to evaluate the required value, as the value generated by the first task can't be directly used in the second task. For example, if you want to create a folder in which the update files you receive daily are kept, and you want to add a date part in the name of the folder, you must evaluate the folder name using the two values: one for the static part and one for the dynamic date part. This is where Integration Services shows its real power. SSIS allows you to evaluate a value using expressions. SSIS tasks expose some of their properties that can be dynamically updated using expressions. Expressions can be used for the following purposes:

- Variables and property expressions can use expressions to update values at run time.
- Integration Services has a For Loop Container, which you will study in the next chapter that uses expressions for the looping criteria.
- As you've seen earlier in the chapter, precedence constraints use expressions to evaluate a condition.
- Data flow transformations such as the Conditional Split Transformation and the Derived Column Transformation have an Expression Builder interface.

Integration Services expressions consist of one or more variables, columns, functions, and operators. You can use all these together to build an expression to evaluate the required value. The language of the Integration Services expressions uses syntax similar to that used in C and C#. The Expression syntax contains identifiers, literals, operators, and functions. The collection of syntax, data type conversion rules, truncation rules, and string padding is called the Expression grammar.

SSIS provides GUI tools to help you build expressions. The Expression Builder is present in most of the Control Flow tasks in the Expressions page to help you apply expressions to properties of the tasks. The expressions you apply to properties are called Property Expressions. Figure 3-11 shows the example of property expression for the File System task that you used in the last Hands-On exercise. If you open the package again and open Expression Builder via the Expressions page and build the expression shown (as explained in the next paragraph), you can then click Evaluate Expression to evaluate this expression at design time.

In the Expression Builder, you can type information into the Expression box and drag and drop any of the items from the top two panes to the Expression box to build

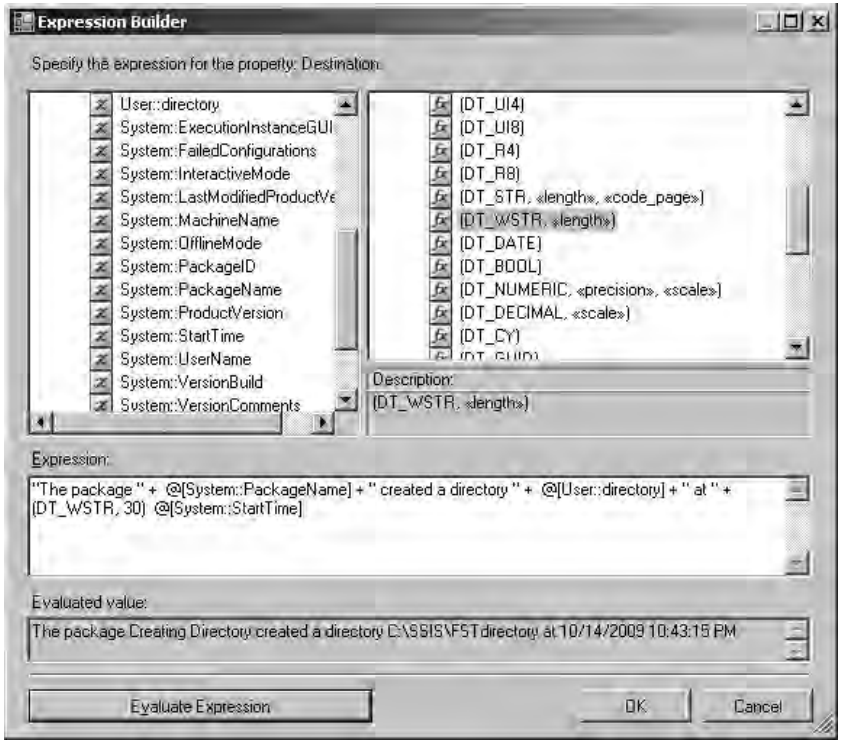


Figure 3-11 *Expression Builder user interface*

your expression. In the top-left pane of this dialog box, you can drag and drop any of the system- or user-defined variables. When you drop variables in the Expression box, the required syntax elements such as the @ prefix are automatically added to the variable names. You can also select columns from this pane if they are applicable to the task and are available there. And from the pane on the right, you can choose from various functions and operators. Figure 3-11 also shows a sample expression, which was built using literals, concatenate operator (+) from the operators, system- and user-defined variables, and a type cast to convert the date type to string.

You can evaluate your expression at design time by clicking Evaluate Expression and the result will be shown in the Evaluated Value field. This will not include any values that are not available at design time. Understand that some of the variables will be populated at run time and they will have no value at design time; hence the evaluation may not give you the exact result that will be created at run time. This evaluation of an expression is accomplished by the Expression Evaluator, which also determines whether expressions adhere to the rules of Expression grammar. As mentioned, property expressions can be applied to any of the exposed properties in Control Flow tasks and to some of the data flow components, as not all data flow components support property expressions. The data flow components that support property expressions expose their properties on the parent data flow task.

You can find functions and operators listed in the top-right pane of the Expression Builder. Integration Services Expression language has a rich set of syntax rules and functions that cannot be covered in detail in this book. However, detailed explanations for the expressions used will be provided wherever applicable. Refer to Microsoft SQL Server 2008 Books Online for more details on expressions.

Following is the brief discussion of these functions and operators.

- **Mathematical Functions** When you expand this folder, you will see 11 functions, such as ABS, LOG, SQUARE, and SQRT. These functions perform calculations based on numeric input values provided as parameters to the functions and return numeric values.
- **String Functions** This folder contains 14 string functions, such as LEN, LOWER, LTRIM, RIGHT, and SUBSTRING. String functions perform specified operations on string or hexadecimal input values and return a string or numeric value.
- **Date/Time Functions** Eight Date/Time functions here, such as DATEADD, GETDATE, and MONTH, perform operations on date and time values and can return string, numeric, or date and time values.
- **NULL Functions** Thirty functions, such as ISNULL and NULL, are available for particular data types. ISNULL returns a Boolean result based on whether an expression is null, whereas NULL returns a null value of a requested data type.

- **Type Casts** These twenty-nine functions help you perform data type conversions for a variable, a column, or an expression from one data type to another, such as string to integer or date/time to string.
- **Operators** About twenty-two operators, such as (+) add, (+) concatenate, (/) divide, and (>) greater than.

You will be using property expressions in various Hands-On exercises throughout this book, but just to tickle your brain, the following is a simple exercise using property expressions.

Hands-On: Using Expressions to Update Properties at Run Time

You need to create a directory to keep the update files you receive daily. But this time, you want to keep the daily update files in their own folder—you need to create a folder each night and want to include a date part in the name of the folder.

Method

You can do this in at least two ways. You can write an expression on the variable's value property to evaluate the new value and pass that value to the SourceConnection field. Or you can go back to the first part of the last Hands-On exercise, in which you used a connection manager for the SourceConnection field and provided a static value of C:\SSIS\FSTdirectory. As part of the second method, you can add property expression on the FSTdirectory Connection Manager to generate a connection string at run time. You will be using the second method in this exercise so that you can have an introduction to the Expression Builder.

Exercise (Configuring the File System Task with Hard-Coded Values)

The first part of this exercise is to open the Creating Directory project and change the settings to go back to a hard-coded value for the SourceConnection field.

1. Open the Creating Directory project with BIDS if it is not already open.
2. Double-click the File System Task to open the editor.
3. Click in the IsSourcePathVariable field and change the value to False.
4. Click in the SourceConnection field and choose <New connection…> from the drop-down list. Choose Create Folder in the Usage Type field of the File Connection Manager Editor. Type **SSISdirectory** in the Folder field. You don't need to provide a path here, as that will be evaluated at run time using the expression you are going to build. Click OK twice to close the File System Task Editor.

Exercise (Using Property Expressions to Evaluate ConnectionString at Run Time)

In this part, you will build a property expression on the ConnectionString property of the FSTdirectory Connection Manager to evaluate the directory name, similar to C:\ SSIS\FSTdirectory20091014, having the date attached as a suffix to the folder name.

5. In the Connection Manager area, right-click the FSTdirectory Connection Manager and choose Properties from the context menu.
6. In the Properties window, click in the Expressions field and then click the ellipsis button. This will open the Property Expressions Editor.
7. Click in the Property column and choose ConnectionString Property from the drop-down list. Click the ellipsis button shown on the right to open the Expression Builder.
8. In the top left pane of Expression Builder, expand Variables and drag and drop the User::directory variable in the Expression area. Then do the following in sequence:
   - Expand Operators and add the concatenate (+) operator.
   - Expand Type Casts and add **(DT_WSTR, <<length>>)**; then change <<length>> to **4** to make it **(DT_WSTR, 4).**
   - Expand Date/Time Functions and add **YEAR(<<date>>)**; then drop the GETDATE() function on the **<<date>>** part of the **YEAR(<<date>>)** function.

     Your expression should look like this:

```
@[User::directory] + (DT_WSTR, 4) YEAR( GETDATE() )
```

9. Click Evaluate Expression, and you will see a value of current year attached to the User::directory variable (which in my case is C:\ SSIS\FSTdirectory2009 at this point in time). Complete the expression by adding the following so that the complete expression looks like one shown in Figure 3-12:

```
RIGHT("0" + (DT_WSTR, 2) MONTH( GETDATE()  ), 2) +
RIGHT("0" + (DT_WSTR, 2) DAY( GETDATE()  ), 2)
```

10. Click OK twice to return to the Properties window. If you expand the Expressions field, you should see the expression you developed earlier.
11. Press F5 on the keyboard to execute the package. When the File System task has turned green, press SHIFT-F5 to stop debugging and go back to design mode. Choose File | Save All and then close the project.
12. Using Windows Explorer, check to see that the required folder has been created.

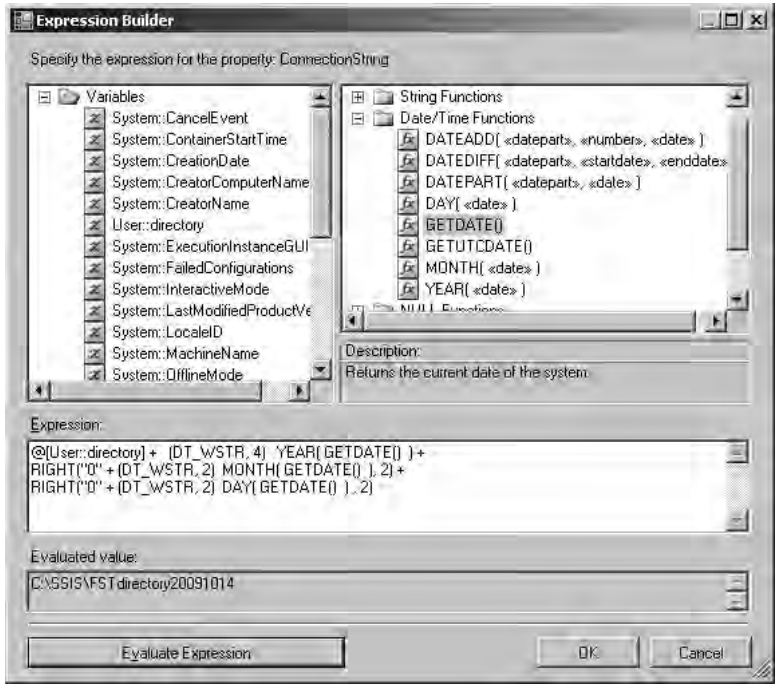


Figure 3-12 *Building a property expression*

Review

You've learned one of the core skills that will help you create packages that can update themselves using the current values at run time. You've used Expression Builder to write an expression. You understand that this can also be written directly without using Expression Builder's drag and drop feature to speed things up, but Expression Builder helps in providing quick syntax help.

Summary

This chapter covered the basic components used for building a workflow in an SSIS package. Integration Services provides connection managers that help various tasks in the package to establish connections to the external data stores. Variables provided in SSIS are different from those in DTS 2000; they can have scope and are categorized as system variables that are read-only which you cannot create and modify, or as the user-defined variables that you create to meet the needs of the package. Precedence

constraints are also enhanced beyond DTS 2000, as they can include expressions to consider along with execution results of the precedence executable. Finally, the most powerful feature of Integration Services is its ability to evaluate the properties dynamically at run time. This is achieved with Property Expressions. Integration Services Expression language is rich in syntax and can be used to create complex expressions using variables, columns, functions, and operators. After having learned about connection managers, system- and user-defined variables, precedence constraints, and the Expressions language, it's time for you to start building packages using these components.

Chapter 4

Integration Services Control Flow Containers

In This Chapter

- Integration Services Package
- Foreach Loop Container
- For Loop Container
- Sequence Container
- Task Host Container
- Summary

Integration Services (SSIS) uses containers to host functional objects that build packages. An SSIS container is an object that represents a unit of work and contains the tasks and precedence constraints. A container can contain other containers and tasks. So, while you are creating an SSIS package, you will be adding and configuring tasks inside a container that will provide necessary services to your tasks to accomplish the work that needs to be done. Functionally, SSIS containers provide repeating logic, the ability to enumerate over a set of items, and the ability to group tasks and child-containers to create an independent unit of work that can communicate with a parent container.

In this chapter, you will study the following five Integration Services containers:

- Integration Services package
- Foreach Loop Container
- For Loop Container
- Sequence Container
- Task Host Container

You will study another container available in SSIS, the Event Handler, in Chapter 8. Its primary function is to support a container based on the events happening at run time.

Integration Services Package

The Integration Services package is an XML document that consists of executables, control flow containers, tasks, data flow transformations, connection strings, variables, event handlers, and other components that all work together. The package object is the most important object and sits at the top of the hierarchy in the Integration Services object model architecture. This means that all other containers and tasks are contained and executed within the context of a package.

You can build a package using GUI tools, such as the Import and Export Wizard or Business Intelligence Development Studio (BIDS), or you can build it programmatically using an IDE or even Windows Notepad. However, you may find it easier to use GUI tools, which let you build a complex package in a short time. After a package is created in BIDS, you can save it on the file system as an XML file or to SQL Server in the sysssispackages table. After saving the package, you can connect to Integration Services using SQL Server Management Studio to manage your packages.

When you created your first project earlier in the book, you also created a blank package inside the solution. In the same exercise, you learned about various components and parts of a package. You can use other objects such as event handlers, variables,

package configurations, breakpoints, and log providers that can greatly enhance package functionality and give you complete control over what you want to achieve and how you want to achieve it.

By now, you have learned that every package will have a control flow and most will include a data flow. SSIS has separated the control flow and data flow in a package. This separation makes it easy for you to design, develop, debug, and maintain complex packages.

When you create a package in the BIDS, you see four tabs on the SSIS designer surface:

- **Control Flow tab** Helps you create package workflow using control flow components such as tasks, connection managers, and precedence constraints.
- **Data Flow tab** Helps you create a data flow using data flow components such as transformations, source adapters, and destination adapters.
- **Event Handlers tab** Helps you create a control flow that runs in response to an event raised by a task or a container inside a package.
- **Package Explorer tab** Shows you a hierarchical view of the elements used in the package.

Although an SSIS package sits at the top of the object model hierarchy and all the tasks and containers are executed within it, a package can be included as a child in other packages. Think of the SSIS package as a container that can contain a child-container—likewise, a package can contain a child-package. SSIS includes the Execute Package task that acts as a wrapper for a package, making it a child package. You can include this task (and hence the child package) in the control flow of a parent package. This functionality makes SSIS modular in design, which means that you can create child-packages as modules, which can then be combined to form a more complex enterprise-wide package. This functionality also reduces complexity in design, making it easier for you to debug and maintain smaller packages. You will learn more about the Execute Package task in the next chapter, where the Control Flow tasks are discussed.

Foreach Loop Container

While designing the workflow of an SSIS package, you may want to perform an operation on the members of a collection. For example, you may want to read the names of the files in a folder and set the values of your variables using those names, or you may want to perform logic on each of the records in a table. The Foreach Loop Container in SSIS provides this functionality by bringing the result set into the workflow; this lets you perform interesting work based on the data in the result set.

With a Foreach Loop Container you can create a repeating control flow in a package. This concept has been borrowed from programming languages. SSIS has made repeating work flow logic for each member of a collection a lot easier by providing the Foreach Loop Container that uses various enumerators to enumerate over different sets of collections.

Here are some examples for which a Foreach Loop Container can be used:

- Notifying your customers by sending out e-mails when an insurance policy is due for renewal
- Processing prospects that have made enquiries during the last month
- Automatically sending out welcome packs to new clients
- Processing dunning letters whenever a payment is overdue by a certain number of days

The following table lists the enumerators that SSIS provides:

Enumerator	Description
Foreach ADO	Enumerates rows in tables such as an ADO record set.
Foreach ADO.NET Schema Rowset	Enumerates the schema information about a data source. For example, enumerate all the tables in an SQL Server database.
Foreach File	Enumerates files in a folder. Can select files from the subfolders if you select Traverse Subfolders option while configuring this enumerator.
Foreach From Variable	Enumerates the enumerable objects that a variable contains. For example, you may enumerate values of an array.
Foreach Item	Enumerates the items in a collection such as list of files in an Excel sheet.
Foreach Nodelist	Enumerates the result set of an XPath expression. For example, the expression `/Products/vehicle[@Bodystyle='saloon']`
Foreach SMO	Enumerates SQL Management Objects (SMO). For example, tables or user collections in an SQL Server database.

The Foreach Loop Container repeats the work flow tasks built into the container for each member of the collection defined by the enumerator. For example, if you want to send out an e-mail to customers on the basis of a table that has 100 rows, the Foreach loop will iterate over the Send Mail task 100 times by selecting to a record, sending out the e-mail for that record, and then moving on to the next record.

You can implement any business logic that requires repetition using child containers and tasks as a single unit of work within a Foreach Loop Container. Once you have

chosen the enumerator to implement the repetition logic, the Foreach Loop Container will iterate multiple times over the business logic you want to repeat for the set of values defined by the enumerator.

Why you would want to do this? Suppose you have created a contacts table, and you want to send an e-mail to each of the contacts listed in the table. You can use Send Mail task provided in SSIS to send these e-mails; however, the challenge, to fill in the e-mail addresses one by one in the Send Mail task dynamically to be able to send an e-mail to each of the contacts, has to be overcome. This is where a Foreach loop can help: it enumerates each contact, passes the e-mail address of each using a variable to the Send Mail task, waits for the task to send the e-mail, and then moves on to the next contact until all the contacts have been sent e-mail. This functionality, which is necessary for customer relationship management (CRM) projects these days, used to mean that custom code had to be written. Now, the Foreach Loop Container makes it easy.

In the following Hands-On exercise, you will configure a Foreach Loop Container to send out e-mails to selected contacts.

Hands-On: Contacting Opportunities

You are tasked to send out first contact e-mail to all the persons who have made enquiries about the products your company makes in the month of October 2009.

Method

In this exercise, you will use the prospect table of the Campaign database to select persons who have made enquiries in the month of October 2009. By now, you must attach the Campaign database provided with the software for this book to your SQL Server 2008 database server; if you have not attached it yet, do that first so that you can complete this exercise. Attaching the Campaign database to your SQL Server 2008 is explained in the Appendix. For this exercise, you will be performing the following steps:

- Add an OLE DB connection manager for connecting to the Campaign database so that you can get data from the prospects table. Add another connection manager for SMTP server for sending out e-mails.
- Add an Execute SQL task in the control flow for selecting the prospects.
- Add a Foreach Loop Container to include the repetition logic.
- Within the Foreach Loop Container, add a Send Mail task for sending out e-mails. The Foreach Loop Container will iterate on the selected records and execute Send Mail task for each record to send out e-mail.

Exercise (Adding the Connection Managers)

In the first part of this exercise, you will create a new project and add connection managers to connect to the Campaign database and Simple Mail Transfer Protocol (SMTP) server. These connection managers will allow your package to interface with external objects to access data and send e-mails.

1. Start Business Intelligence Development Studio. Choose File | New | Project to create a new project. In the New Project window, click OK after specifying the following:

Template	Integration Services Project
Name	Contacting Opportunities
Location	C:\SSIS\Projects

2. In the Solution Explorer window, right-click Package.dtsx and choose Rename from the context menu. Rename the package **Mailing Opportunities.dtsx** and click OK in the pop-up confirmation dialog box. Choose File | Save All to save the newly created package.
3. Right-click in the Connection Managers area and choose New OLE DB Connection from the context menu. In the left pane of the Configure OLE DB Connection Manager dialog box (Figure 4-1), under Data Connections, you will see a list of connections that were created earlier. Choose localhost.Campaign from the list and click OK to close the window. The package gets connected to the Campaign database and the localhost.Campaign connection manager appears under the Connection Managers tab. If you don't find the localhost.Campaign connection manager listed under Data Connections, refer back to the "Using System Variables to Create Custom Logs" Hands-On exercise in Chapter 3 to learn how to create this connection manager.
4. Right-click in the Connection Managers area and choose New Connection from the context menu. In the Add SSIS Connection Manager window, select the SMTP connection manager type and then click Add.
5. In the SMTP Connection Manager Editor window Name field, type **My SMTP Server**. In the Description field, type **Connection to my SMTP Server**. In the SMTP Server field, type in the name of your SMTP server as shown in Figure 4-2.

   In the figure, you can see that I'm using localhost is used to send e-mails. You can use your corporate SMTP server or install SMTP service component of Internet Information Services (IIS) on the local machine and forward all the messages to the gateway SMTP server. You can install IIS from the Control Panel by clicking the Add or Remove Programs icon and then Add/Remove Windows Components.

Figure 4-1 *Adding the OLE DB Connection Manager*

If you are using a Microsoft Exchange Server that is configured not to accept unauthenticated connections, select the Use Windows Authentication check box. You can also choose to encrypt communication using Secure Socket Layer (SSL) when sending e-mails from this server. Whichever way you choose to send e-mails, make sure your SMTP server is prohibited from relaying e-mails for external servers and hence prevents any potential attacks from mass mailers.

Click OK to return to the SSIS Designer.

Exercise (Configuring Execute SQL Task)

In the second part of this exercise, you will configure an Execute SQL Task to select prospects that have enquired in the month of October 2009. While configuring this task,

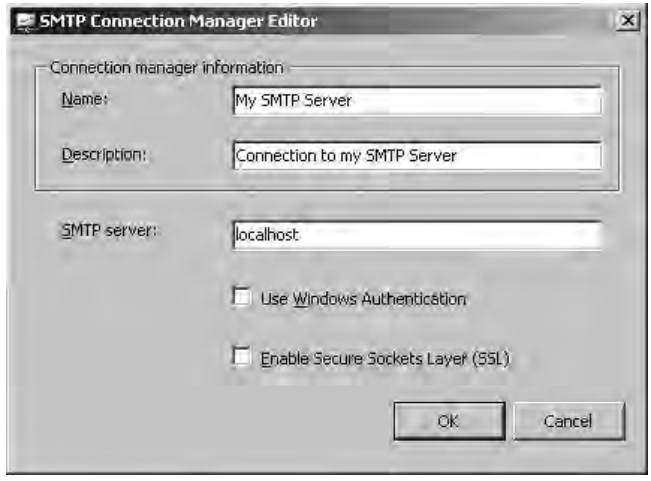


Figure 4-2 *SMTP connection manager configurations*

you will type an SQL statement to extract the required records and will add a variable to store those extracted records.

6. In the Control Flow tab of the designer, go to the Toolbox window and drag and drop an Execute SQL Task on the control flow surface. Double-click the Execute SQL Task icon to open the Execute SQL Task Editor. Type in the following values in the General tab of the editor window:

Name	Prospects
Description	SQL to choose persons who inquired in the month of October 2009.

   As a best practice you should always add a description to the SSIS tasks and components, as it is a convenient way to self-document the package. In the complex packages where you have multiple similar tasks, the description lets you quickly understand what each component is doing. You don't even have to open the task—just hover your mouse on the task in the Control Flow surface and the description will be shown to you as a task hint.
7. Click in the ResultSet field and choose Full Result Set from the drop-down list. The choice of the option in this box depends on the result set expected from your SQL statement. In this case, the select statement is going to return more than one row, so you will choose a Full Result Set option here. These result set options are explained in detail in Chapter 5.
8. Leave the ConnectionType set to OLE DB and from the drop-down list in the Connection field choose localhost.Campaign connection manager.

9. In the SQLSourceType field, leave Direct Input selected. You can see the list of available options, however, by clicking in the SQLSourceType field. You can either input your SQL statement directly in the task or input from a file or read from a variable. In this exercise, you will be entering your SQL in the task directly.
10. Click in the SQLStatement field and an ellipsis button appears in the right corner of the field. Click the ellipsis button and you will see the Enter SQL Query window. Type in the following SQL query and click OK to return to the Execute SQL Task Editor.

```
SELECT TITLE, FIRSTNAME, LASTNAME, EMAIL, ENQUIRYDATE
FROM PROSPECTS
WHERE ENQUIRYDATE BETWEEN '2009/10/01' AND '2009/10/31'
```

The Execute SQL Task Editor should look as shown in Figure 4-3.

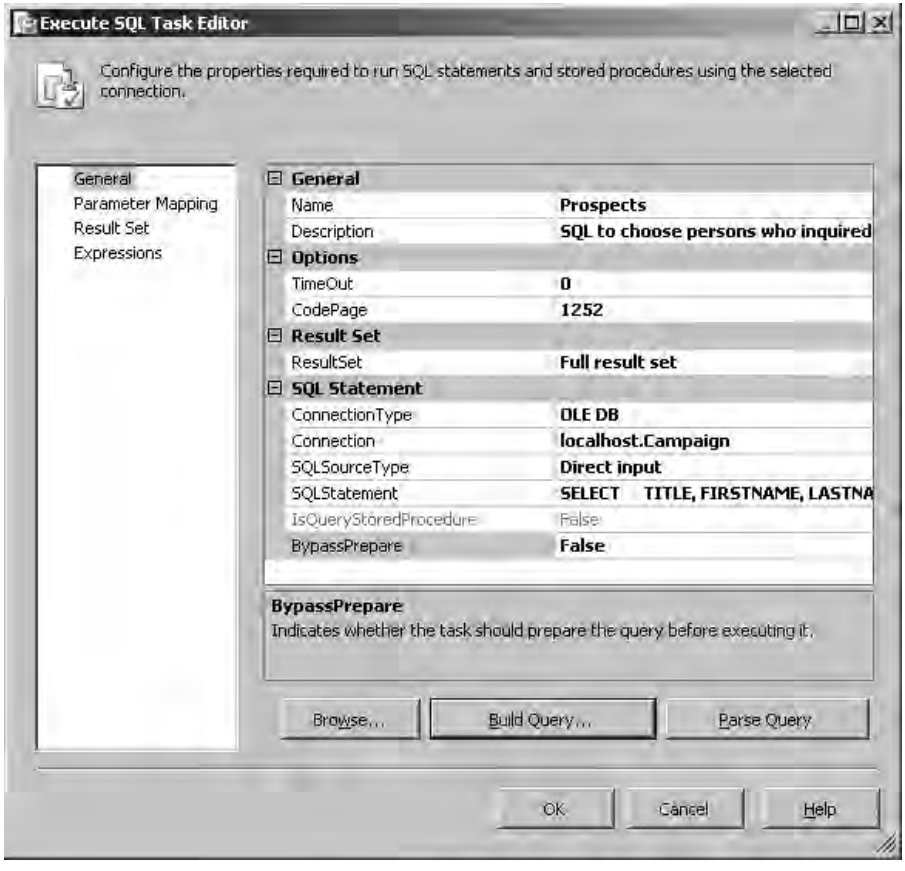


Figure 4-3 *Configuring the Execute SQL task*

11. In your package, you will store the result set to a variable and then use this variable in the Foreach Loop Container to let it iterate over the resultant records. The result set records will be held in a variable of the Object data type, as they can hold a dataset and will be read by the Foreach Loop Container one by one and fed into the defined tasks for further processing.
12. From the left pane of the Execute SQL Task Editor, click the Result Set page.
13. In the Result Set page, click Add. NewResultName will appear under the Result Name column. Delete it and type **0** (zero) in this field. Click in the field under the Variable Name column. Click the drop-down arrow to select <New variable…>. In the Add Variable window, in the Name field type **Opportunities**; leave the Namespace field set at User, and in the Value type field, select Object, as shown in Figure 4-4.

    Variables are case-sensitive, so when you type to add or to select a variable, pay attention to the case of variable. Click OK to return to the Task Editor window. You will see User::Opportunities added as a user variable under the Variable Name column.
14. Click OK to close Execute SQL Task Editor that will select the prospects at run time.

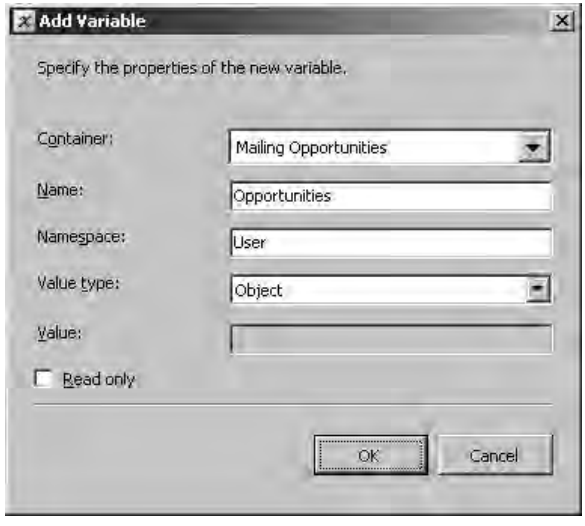


Figure 4-4 *Creating a user variable*

Exercise (Adding Foreach Loop Container)

Here, you will configure the Foreach Loop Container using the Foreach ADO enumerator that allows you to enumerate over records from a record set as provided by Opportunities variable.

15. From the Toolbox, drag and drop a Foreach Loop Container below the Prospects task. Now click on the Prospects and drag the green arrow appearing below it onto the Foreach Loop Container. This is the precedence control letting the Foreach Loop Container to proceed if the previous Prospects task has completed successfully.
16. Double-click the icon of the Foreach Loop Container to open the editor. In the General page, type in the following:

Name	Iterating October Opportunities
Description	This will iterate over the record set obtained from October Opportunities.

17. Click the Collection in the left pane to move on to the next page. Click in the Enumerator field to see the drop-down arrow, click it, and choose Foreach ADO Enumerator from the drop-down list. Note that Enumerator configuration area changes to suit the enumerator you choose.
18. In the Enumerator Configuration box, select User::Opportunities in the ADO Object Source Variable field, as shown in Figure 4-5. By specifying User::Opportunities here, you are telling the Foreach Loop Container to get the records stored in the variable. This is a good example of tasks communicating with each other—here, the Prospects task populates variable with values for the downstream components to use those values.
19. Since only one dataset will be generated in your example, select the Rows in the first table radio button, as shown in Figure 4-5. Click OK to close Foreach Loop Editor window.

Exercise (Adding Send Mail Task and Executing the Package)

Now configure the Send Mail Task so that you can send e-mails for each of the selected records.

20. From the Toolbox, drag and drop the Send Mail task *inside* the Foreach Loop Container.
21. Double-click the Send Mail Task icon to open the editor. In the General page, type in the following details:

Name	Mailing Opportunities
Description	It will send a mail for each of the records iterated by the Foreach Loop Container.

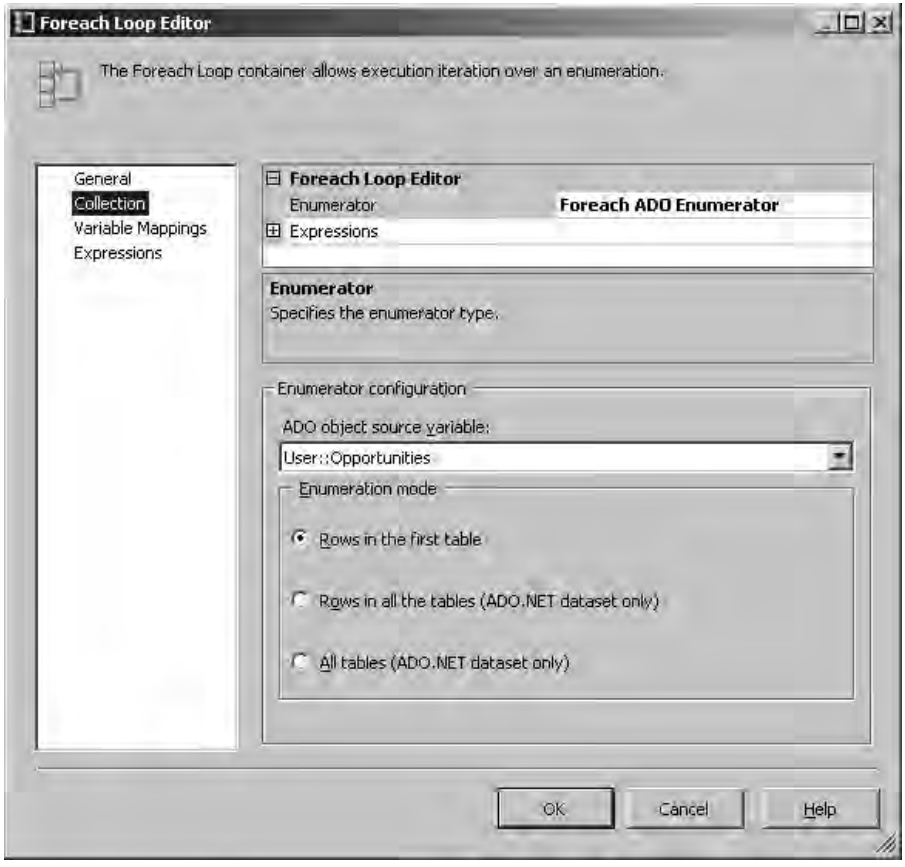


Figure 4-5 *Configuring the Foreach Loop using Foreach ADO Enumerator*

22. Go to the Mail page. From the drop-down list in SmtpConnection field, select My SMTP Server.
23. To keep your first package simple, you will be sending e-mails to yourself in this exercise. Type your e-mail address in the From and To fields and type **Your Enquiry** in the Subject field. You can actually read e-mail addresses from a database table for these two fields using property expressions, which you will study in Chapter 8.
24. In the MessageSourceType, leave Direct Input selected, as shown in the Figure 4-6. This field provides you three options: you can type your message directly in the task or get your message from a file or a variable.

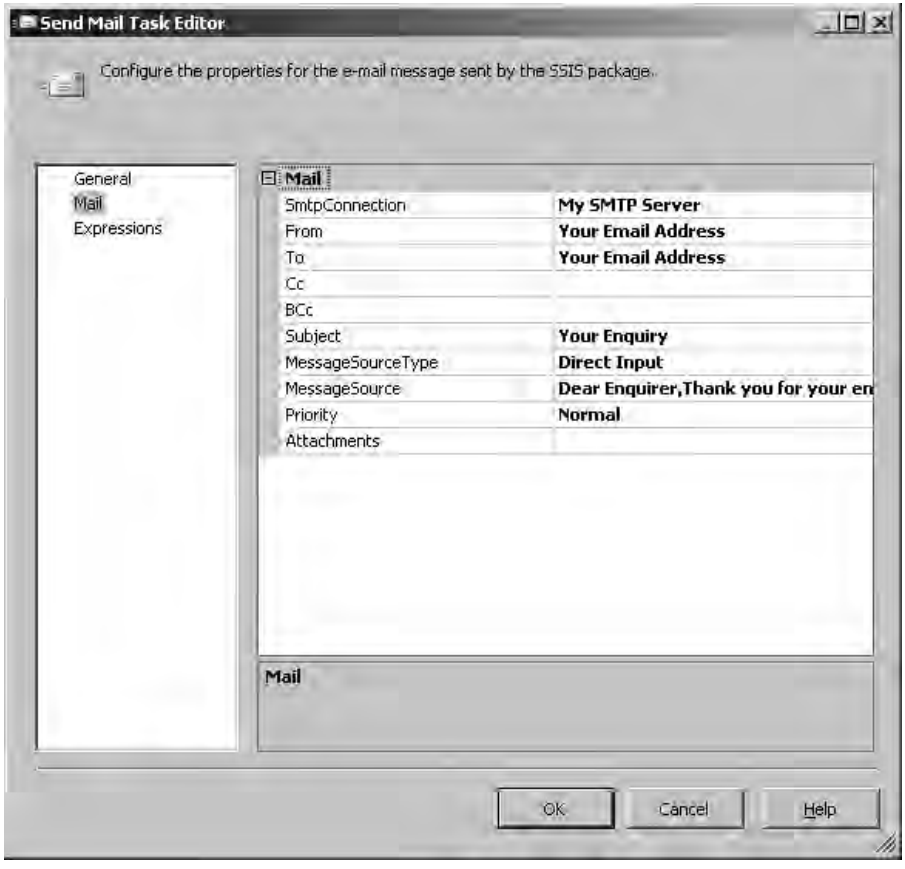


Figure 4-6 *Send Mail task configurations*

25. Click in the MessageSource field and then click the ellipsis button to start typing your message in the Message Source window. Here is a simple message you can use for this example. Type in the following message and click OK.

    Dear Enquirer,

    Thank you for your enquiry. One of our sales representatives will be in touch with you. In the mean time, please go to our web site for more information on our products.

    Thank you very much for showing interest.

    Kind regards,

    Sales Support Team

26. Leave Priority set to Normal, though High, Normal, or Low options are available as well. Leave the Attachments field blank. Click OK to close the dialog box.

    By now, your package should look like the one shown in the Figure 4-7.

27. Press F5 on the keyboard to start debugging. You will see the tasks changing colors. First, Prospects (Execute SQL Task) will turn yellow and then to green to indicate that the task has completed successfully. Then Iterating October Opportunities (Foreach Loop Container) and Mailing Opportunities (Send Mail Task) will turn from yellow to green quickly in turn for as many times as the number of records in the collection. This is a good visual experience to realize that the Foreach Loop Container iterates over each item in the collection and processes the logic you design for it. Check your Inbox to see the e-mails you have sent. If your mailing system has relay restrictions, you may have to adjust them to complete the execution of the package successfully.

28. Choose File | Save All and then choose File | Close Project to close the Contacting Opportunities project. Finally, exit from BIDS and it's time for a coffee break, but only after reading the review.

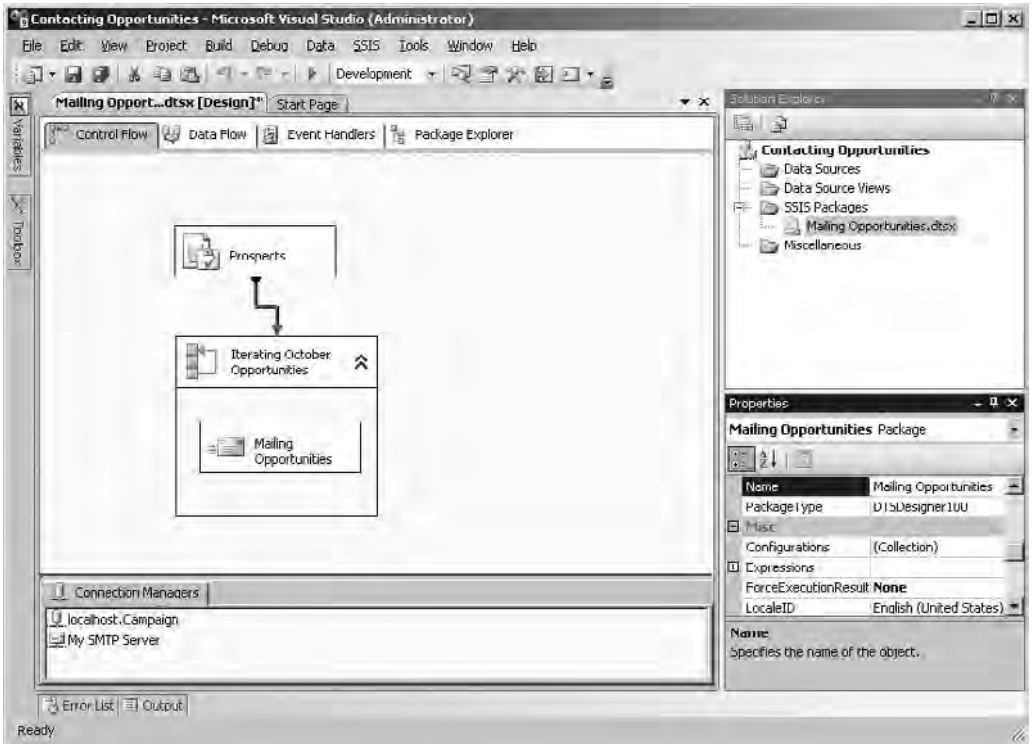


Figure 4-7 *Mailing Opportunities package*

Review

In this exercise, you created a workflow for selecting prospects and sent e-mails using the Execute SQL task, Foreach Loop Container, and Send Mail task. You have now seen most of the configuration settings for these tasks, but there are some interesting things that can still be done in this package—such as picking up e-mail addresses of the recipients from a data set; using a combination of title, first name, or last name in the addressing label; and specifying the reference by mentioning the enquiry date in the subject field. The good news is that all these can be done in Integration Services using property expressions. Property expressions help you dynamically update properties of a task such as the To address in the Send Mail Task. Property expressions are covered in more detail in Chapter 8, where we'll work with the advanced features of SSIS and extend this package to send personalized mails.

For Loop Container

The For Loop Container is easier than the Foreach Loop Container to work with. If you have worked with programming languages, you must be aware of popular Do While statements. Generally the syntax of these statements defines a numeric variable, with a starting value, the incrementing step after each iteration, and the ending value, along with statements to be executed repeatedly till the end point is reached. The For Loop Container provides a similar functionality of repeating a task or group of tasks for a defined number of times. The For Loop Container iterates over and over while evaluating an expression with each repeat until the expression evaluates to False.

To understand this, consider a Send Mail Task configured inside the For Loop Container. If you set to start the looping with counter value equal to zero, the increment step to one, and the maximum allowed value that can be reached to less than five, then the For Loop Container will iterate the Send Mail task five times, which in turn will send out five messages. The package starts with counter value equal to zero to be increased by one with each iteration. On each iteration, the current value of the counter is checked against the maximum possible value of five. If the current counter value is less than five, it will let the Send Mail task run and will increase the counter value by one before it evaluates again. When the counter value reaches five, it determines that the value has reached the maximum allowed and will not proceed beyond this point; it stops the execution. The evaluation expression at this point returns a value of False.

To do these iterations, the For Loop Container uses the following elements:

- **InitExpression** An optional initialization expression used to assign initial values to the loop counter.

- **EvalExpression** A required evaluation expression evaluated at the onset of each iteration to decide whether the loop should continue or stop. This is a Boolean evaluation, and when the expression evaluates to FALSE, the process exits looping.
- **AssignExpression** An optional iteration expression that increments or decrements the loop counter.

You must provide an evaluation condition, but initialization and assignment expressions are optional. A For Loop Container can have only one evaluation expression and runs all the control flow tasks included in it the same number of times. You can use either literals or expressions while specifying an evaluation condition, an initialization counter value, or an assignment step value. The property expressions can include variables, which can be dynamically updated at run time. You can use this feature quite effectively to change the number of iterations each time the package is run, depending upon the values of variables.

Let's compare the For Loop Container with Foreach Loop Container task, which you studied in the preceding section. The For Loop Container provides a looping functionality that conditionally iterates a control flow defined within the container, whereas the Foreach Loop Container provides a looping construct that enumerates files and objects in the control flow of a package—that is, it executes the control flow defined within the container for each item in the collection. Neither container provides any functionality; rather they provide only a structure in which you can build a repeatable control flow. Both containers can include a control flow with multiple tasks in addition to other containers. You can in fact build nested loops and implement complex looping in your packages using these containers.

Hands-On: Deleting Data Month by Month After Archiving

In your data warehouse, you have to keep data for the last five years to meet business needs. So, at the beginning of every year, you archive data to tape that is older than last five years and then delete from the data warehouse tables. However, your DBA does not allow you to delete all one year worth of rows in single SQL statement because it swells the transaction log too much and requires lot of free space. So, you decided to delete data month by month and, to achieve this further, decided to use For Loop Container.

Method

In this exercise, you will delete data month by month without going into the other details of archiving and managing a server. The Campaign database contains a Sales table that you will use to study the For Loop Container and then use to add iterations in your package. The Sales table in the Campaign database is provided with the

software for this book. So, before you start, make sure you have attached the Campaign database as mentioned in the Appendix.

Our Sales table has 24 rows for year 2005, with 2 rows belonging to each month. This table has been simplified for this exercise and the rows for other years have not been included for the sake of clarity. Run the following query against the Sales table in SQL Server Management Studio to see the record set:

```
SELECT * FROM [dbo].[Sales]
```

In this exercise you will delete all these rows using For Loop Container within the package you develop. You will iterate over the data 12 times, that is, once for each month, And in each iteration you will delete rows attached to that month, in our case 2 rows each time.

Exercise (Add a For Loop Container)

As a first step let us add a looping functionality in a new package to iterate for 12 times.

1. Start BIDS. Choose File | New | Project. Create a new Integration Services project specifying the following:

Name	Deleting data month by month
Location	C:\SSIS\Projects

   Click OK to create a new project.
2. Rename the Package.dtsx to **Deleting data month by month.dtsx** by right-clicking Package.dtsx in Solution Explorer.
3. Right-click in the Connection Managers area and choose New OLE DB Connection from the context menu. Select the localhost.Campaign OLE DB connection manager listed in the Data Connections area from the Configure OLE DB Connection Manager dialog box. Click OK to add this connection manager.
4. Right-click anywhere on the designer surface and choose Variables from the context menu. In the Variables window, add a variable with the following details:

Name	MonthCounter
Scope	Deleting data month by month
Data Type	Int32
Value	0

   You will set the value equal to 0 here, which will be changed during run time. Click OK to create this variable.

5. Drop the For Loop Container from the Toolbox on to the designer surface.
6. Right-click the For Loop Container and choose Edit from the context menu. Type the following in the For Loop page of the For Loop Container:

Name	Looping for deleting monthly data
Description	This container adds a looping structure for deleting monthly data.
InitExpression	@MonthCounter = 1
EvalExpression	@ MonthCounter <= 12
AssignExpression	@ MonthCounter = @ MonthCounter + 1

By setting these configurations, you are actually telling the For Loop Container to initialize the MonthCounter variable with a value equal to 1 and increase it by one after each iteration. The MonthCounter variable will be evaluated against the number of month in a year, as you are deleting one year's worth of data, to decide whether to continue to iterate the For Loop Container.

Click OK to close the For Loop Container. The For Loop Container configurations are shown in Figure 4-8.

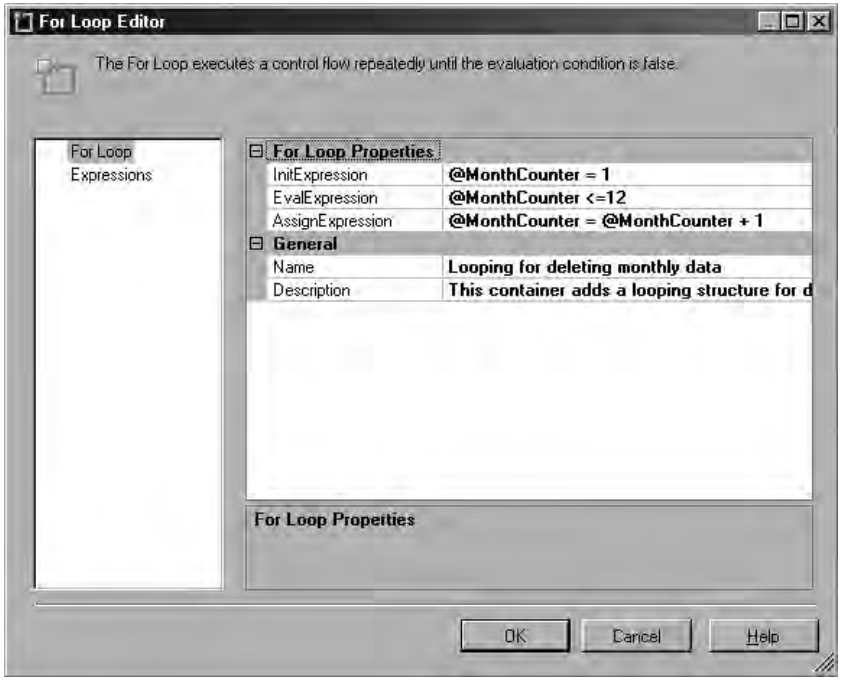


Figure 4-8 *Configuration settings of the For Loop Container*

Exercise (Delete Monthly Records)

Now you will add an Execute SQL task within the For Loop Container for applying the SQL script to delete records for the month specified in the form of a parameter by For Loop Container. After completing the development work, you will run the package and will see how the rows are deleted using breakpoints.

7. Drag and drop the Execute SQL task from the Toolbox inside the Looping for deleting monthly data For Loop Container. Double-click the new Execute SQL task to open the Execute SQL Task Editor window. Type the following in the General page of the Execute SQL task:

Name	Delete monthly records
Description	This task deletes monthly records from the table.

8. Leave all the other options as they are, choose the localhost.Campaign connection manager from the drop-down list in the Connection field as you have done in earlier tasks, and type the following query in the SQLStatement field (by now you know how to do that):

```
DELETE FROM [dbo].[Sales] WHERE YEAR(SalesOrderDate) = 2005 AND
MONTH(SalesOrderDate) = ?
```

   This SQL query deletes all the records belonging to the month specified by the MonthCounter parameter in a single iteration of the For Loop Container.

9. Go to Parameter Mapping page and click Add. Select User::MonthCounter variable as shown in Figure 4-9.

   This will pass the value of MonthCounter variable from the For Loop Container to the Execute SQL Task. As the For Loop Container iterates and increases the value of MonthCounter variable one by one, the Execute SQL Task deletes data from the Sales table month by month. Close the Execute SQL Task Editor by clicking OK.

10. Before we execute the package, let's add some breakpoints to the package to see what happens at run time. Right-click the Delete monthly records Execute SQL task and choose the Edit Breakpoints option from the context menu. You will learn a lot more about the breakpoints later in Chapter 8, but for now simply remember that they help you to see the run-time status of a package by pausing its execution and letting you to see the variable values and such. Choose the very first break condition that breaks the package OnPreExecute event (see Figure 4-10). Setting this breakpoint will pause the execution before each time the Execute SQL Task runs. Click OK to set the breakpoint.

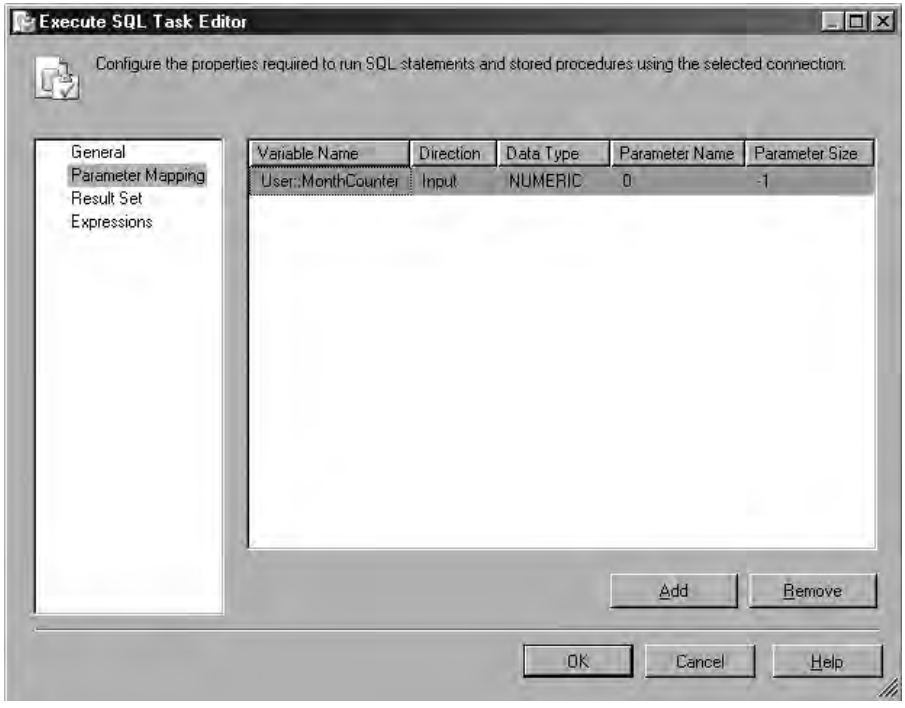


Figure 4-9 *Setting an input parameter for Execute SQL task*

11. You are now ready to Start Debugging from the Debug menu or by pressing F5. You will see the For Loop Container turning yellow and the package execution stopped at Execute SQL Task. Open the Locals window from the bottom left of the BIDS screen. If you don't find it there, you can also open it by clicking Debug | Windows | Locals menu bar command. Expand the variables and scroll down to locate the User::MonthCounter variable. Note that the value is set to 1. Now press F5 to continue processing. The Execute SQL Task will run and will turn green. After that the For Loop Container will start the second iteration by incrementing the value of User::MonthCounter to 2 and the package will break again before executing the Delete monthly records task as shown in Figure 4-11. Note that the value of User::MonthCounter variable is 2 now.
12. Switch to SQL Server Management Studio and run the following SQL Command to see the records left in the Sales table.

    ```
    SELECT * FROM [Campaign].[dbo].[Sales]
    ```

    Note that the records belonging to month 1 have been deleted from the table with only 22 records left in the table.

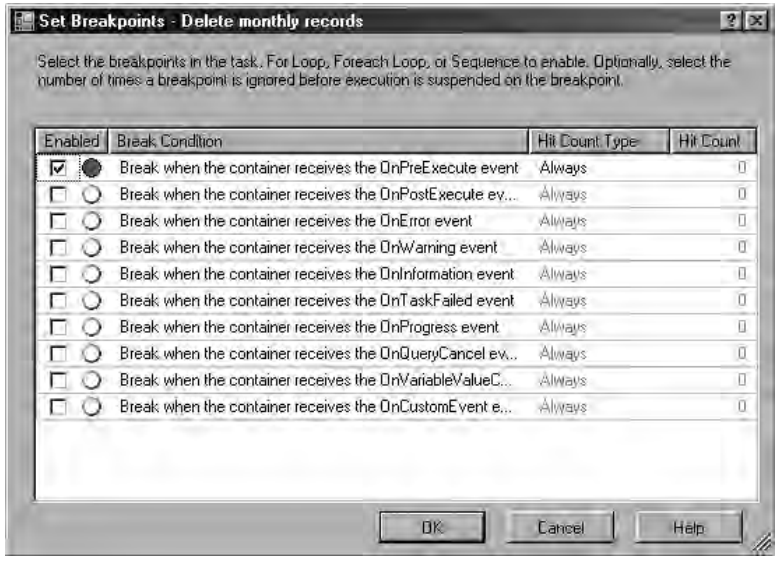


Figure 4-10 *Setting an OnPreExecute event break condition on the Execute SQL task*

13. Keep running the package to completion while checking the values and the records deleted from the Sales table. You will notice that the tasks turn from yellow to green and particularly that the Delete monthly records Execute SQL task inside the For Loop Container task turns yellow and green in turn, demonstrating that this task is running many times—to be precise, it runs 12 times, thus deleting one year worth of data. Once the package completes execution, stop debugging the package by pressing SHIFT-F5 and click the Execution Results tab to see how the package execution progressed (Figure 4-12).

    Look in the middle of the window to watch for the Looping for deleting monthly data section, which lists the For Loop Container execution results. Notice the *Start (12),* then 12 lines specifying the query task 100 percent complete, and then *Stop (12).* This signifies that the "Delete monthly records" task has been run 12 times successfully and the thirteenth time, the For Loop Container evaluation expression evaluated to False, causing the process to exit from the For Loop Container iterations.

14. Press CTRL-SHIFT-S to save all the objects in the package. Close the Project and exit from BIDS. Switch to SQL Server Management Studio and verify that at the end of the package all the records have been deleted from the table.

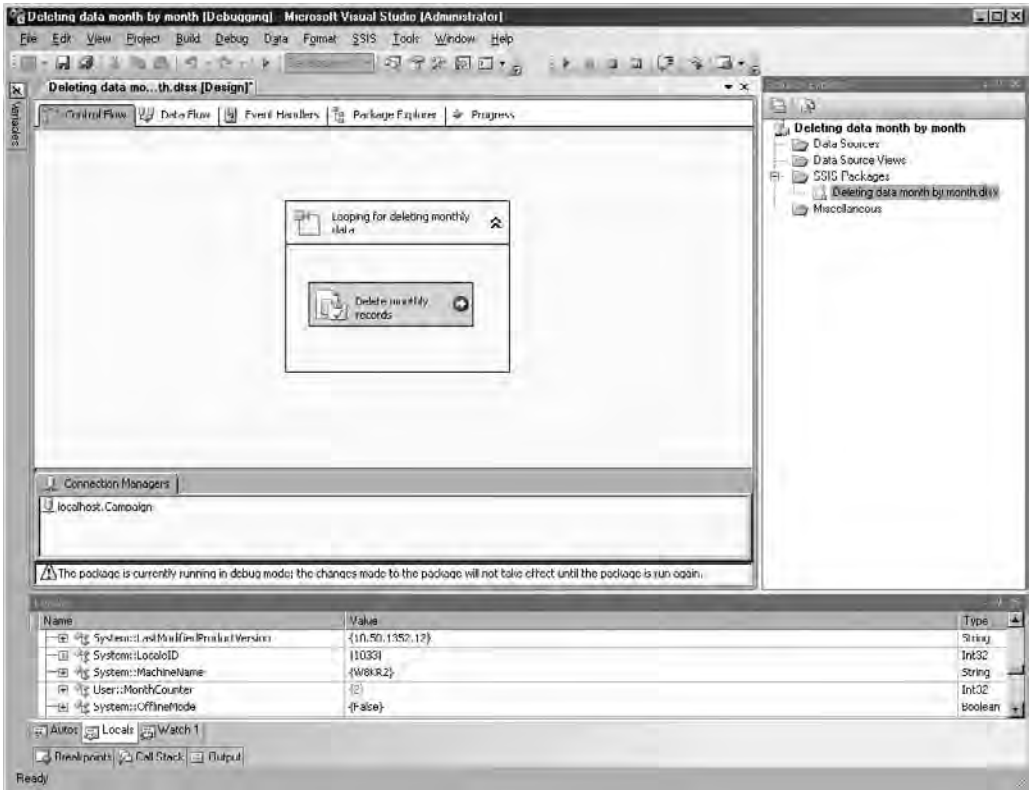


Figure 4-11 *Breaking the package and monitoring the values of variables*

Review

In this hands-on exercise, you learned how to use and configure the For Loop Container. You have used the For Loop Container to delete data from a table on a month-by-month basis. Whichever way you use the For Loop Container, the bottom line is that it allows you to add iterations—i.e., the looping functionality to your packages whenever you need to use them. If you want to run this exercise for a second time, you will have to bring Sales table to the original state, which you can do by using the Sales_Original table, which contains an initial copy of the Sales table.

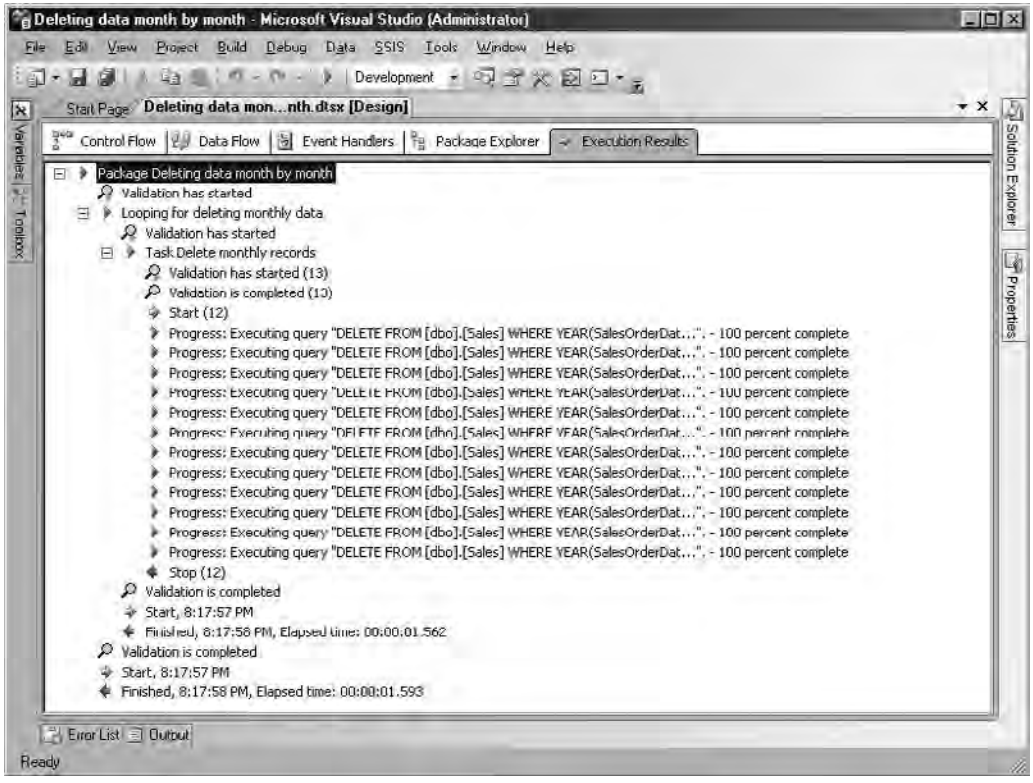


Figure 4-12 *Execution results of the Deleting data month-by-month package*

Sequence Container

The Sequence Container helps organize a complex package with several tasks by allowing the logical tasks to be grouped together. A Sequence Container is a subset of the package control flow containing tasks and containers within it. You can organize clusters of tasks with Sequence Containers and hence divide the package into smaller containers, rather than dealing with one huge container (Package Container) with lots of tasks.

It will be worth highlighting here that a similar grouping feature has also been provided in the Control Flow of an SSIS package. If you have a very complex package that consists of several tasks and containers, sometimes you may find it hard to locate the tasks that you want to work on; in such cases you may choose to group certain tasks together to simplify the visual look and feel of the Control Flow area. You can

group the tasks and containers together by first selecting them and then right-clicking one of them to select the Group option. Once grouped together, these groups can then be collapsed and expanded based on your need to free up the working area on the desktop. However, note that the grouping feature is a design-time feature only and is provided for to simplify the visual display of the Control Flow surface. Such groups have no properties or configurations assigned to them, and they do not affect the execution or run-time behavior of a package. On the other hand, a Sequence Container is a very powerful object that allows you to create multiple separate Control Flows in a package and not only group tasks and containers but also affect their run-time behavior with its own properties. For example, you can disable a Sequence Container to avoid execution of that section of the package, while you can't do the same with the simple grouping feature.

While developing your Integration Services project, you can add this container from the Toolbox into the Control Flow of your package. When you double-click the task, you won't see any user interface popping up where you can configure this task; rather, the context is changed to the Properties window. The Sequence Container has no custom user interface and can be configured using its Properties window; thus it behaves like a Package Container. You can build your control flow within the Sequence Container by dragging and dropping the tasks and containers within it.

Following are some of the uses for which Sequence Containers can be helpful:

- Organizing a package by logically grouping tasks (tasks focusing on one functional area of business) makes the package easy to work with. You may choose to run one subset only from the package, depending upon the need.
- Setting properties on a Sequence Container may apply those settings to multiple tasks within the container; this allows you to configure properties at one place instead of via individual tasks, making management easier. For example, you can use the TransactionOption property to manage the transaction support for all the tasks in the Sequence Container, that is, if one task fails, all tasks fail. This means you can build a rollback facility for one piece of logic.
- Such grouping of tasks makes it easy to debug the package. You can disable a Sequence Container to switch off functionality for some business area and focus on debugging the other areas.
- Sometimes you may need to isolate variables from certain tasks. In such cases, you can limit the scope of variables to a group of tasks and containers by keeping them inside a Sequence Container.

Task Host Container

You have worked with the package, the Foreach Loop Container, the For Loop Container, and the Sequence Container in BIDS by using their Editors or Properties windows. The benefits of having a container design in SSIS include sharing of variables between parent and child containers and being able to manage the events happening within a child container. The Task Host Container is not available in BIDS Toolbox as a Container, so you can't configure it directly and drop tasks in it; rather, it is a wrapper around each Control Flow task. The Task Host Container encapsulates each task when the task is embedded in a workflow on the designer surface. Encapsulating a task empowers the task to use features of a container such as Event Handlers. This container is configured when you configure the properties of the task it encapsulates.

Summary

The workflow in Integration Services consists of containers. Two containers, For Loop and Foreach Loop, provide the repeating logic, which before SSIS, was available to procedural languages only. With the advent of SSIS, you have at your disposal the For Loop Container to perform repeating logic to the tasks contained within it, the Foreach Loop Container to perform the repeating logic for each item in a collection, and the Sequence Container to group tasks and containers, allowing you to have multiple control flows by having multiple Sequence Containers under one package.

In the beginning of the chapter, when you learned about the Package Container, you read that within the Integration Services architecture design, a package sits at the top of the container hierarchy. Yet Integration Services provides a facility to embed a package in another (parent) package by wrapping the (child) package in a wrapper task called the Execute Package task. You will study this task and do a Hands-On exercise for the Execute Package task in the next chapter. Finally, you also learned that each task is treated like a container in Integration Services, as the Task Host Container encapsulates each task. This allows tasks to inherit the benefits of containers.

This page intentionally left blank

Chapter 5

Integration Services Control Flow Tasks

In This Chapter

- Categories of Control Flow Tasks
- Control Flow Tasks in Detail
- FTP Task
- Execute Process Task
- File System Task
- Web Service Task
- XML Task
- Execute SQL Task
- Bulk Insert Task
- Message Queue Task
- Execute Package Task
- Send Mail Task
- WMI Data Reader Task
- WMI Event Watcher Task
- Transfer Database Task
- Transfer Error Messages Task
- Transfer Jobs Task
- Transfer Logins Task
- Transfer Master Stored Procedures Task
- Transfer SQL Server Objects Task
- Back Up Database Task
- Check Database Integrity Task
- Execute SQL Server Agent Job Task
- Execute T-SQL Statement Task
- History Cleanup Task
- Maintenance Cleanup Task
- Notify Operator Task
- Rebuild Index Task
- Reorganize Index Task
- Shrink Database Task
- Update Statistics Task
- Summary

Now that you understand the architecture of Integration Services, how it uses system and user variables and how the control flow containers can be used in the work flow of a package, it is time to learn more about the control flow tasks that can fabricate a complex workflow to perform a sensible piece of work.

This chapter is divided into two main parts: the first covers tasks that play a direct role in building up workflow for your packages, and the second covers tasks that are designed to perform SQL Server maintenance, though they can also be used in the workflow. You have already used some of these control flow tasks, such as the Execute SQL task and the Send Mail task, in the packages you have built so far. The tasks in SQL Server Integration Services (SSIS) define the unit of work necessary to create the control flow in a package. This unit of work can comprise downloading files, moving files, creating folders, running other processes and packages, sending notification mails, and the like. You will be using some of these tasks to build a couple of control flow scenarios to demonstrate how these components can be used together. SSIS also provides a facility to create your own custom tasks if the existing control flow tasks don't fit the bill. Integration Services supports the Microsoft .NET Framework, so you can use any .NET-compliant programming language, such as Visual Basic .NET or C#, to write custom tasks.

In the latter part of this chapter, you will read about Maintenance Plan tasks that can be used to perform database maintenance functions for SQL Server 2000 and above. Using these tasks, you can create quite an elaborative control flow of a package that is designed to manage and maintain databases. Though you may find that these tasks do not directly contribute to building workflow for a package that is designed to solve a business problem, they can still be included in the control flow if required to perform maintenance of SQL Server objects, for example, rebuilding an index after a bulk load.

Let's wade through the Integration Services tasks first before swimming into deeper waters.

Categories of Control Flow Tasks

Though the Toolbox in Business Intelligence Development Studio (BIDS) shows Control Flow tasks in two main groups, the Control Flow tasks can be categorized on the basis of the functionality they provide. Read through the following introductions quickly to understand how these tasks are categorized and organized in the control flow. You can refer back this section whenever you need a quick description of a task.

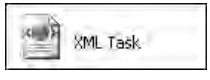
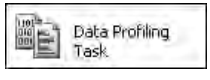
Data Flow Task

This main task performs ETL (extracting, transforming, and loading) operations using the data flow engine.

Control Flow Task	Description
	Perform the ETL functions—i.e., extract data from a source, apply transformations to the extracted data, and load this data to a destination. In BIDS, this task has its own designer panel and consists mainly of source adapters, transformations, and destination adapters. This task is covered in detail in Chapters 9 and 10.

Data Preparation Tasks

These tasks help in managing and applying operations on files and folders and identifying issues with data quality.

Control Flow Task	Description
	Perform operations on files and folders in the file system—i.e., you can create, move, delete, or set attributes on directories and files. You will be using this task in a Hands-On exercise later in this chapter.
	Download and upload files and manage directories—for example, you can download data files from your customer's remote server using this task to apply transformations on the data in these files using Data Flow task and then upload the transformed data file to a different folder on the customer's FTP server, again using the FTP task.
	Read data from a Web Service method and write that data to a variable or a file—for example, you can get a list of postal codes from the local postal company using this task and use this data to cleanse or standardize your data at loading time.
	Dynamically apply operations to XML documents using XSLT (extensible style sheet language transformation) style sheets and XPath expressions and merge, validate, and save the updated documents to files and variables—i.e., you can get XML documents from many sources, merge these XML documents to consolidate the data, and reformat for presentation in a report layout.
	You have already used this task in Chapter 2. Using this task, you can profile data in your data sources to identify the data quality issues before loading it into a data warehouse. The knowledge gained from data profiling will help you to design a data cleaning and standardization process in the scheme of your data warehouse.

Workflow Tasks

Workflow tasks communicate with other processes to execute packages, run programs or batch files, send and receive messages between packages, send e-mail messages, read Windows Management Instrumentation (WMI) data, and watch for WMI events.

Control Flow Task	Description
Execute Package Task	Run child packages—i.e., develop packages as modules of work items and choose a combination of such work item modules using this task to run as under a master package or break down a complex package into separate units of work and run them together.
Execute Process Task	Run a business application or a batch file to perform specific functions that can't be done easily in SSIS, whose output can then be included in the SSIS package.
Message Queue Task	Queue your messages if the destination is unavailable or busy and deliver such messages later. This task uses Microsoft Message Queuing (MSMQ) to send and receive messages.
Send Mail Task	Send messages from your packages, for example, on the basis of success or failure of the package or on the basis of an event raised during execution of the package.
WMI Data Reader Task	Run WQL (WMI Query Language) queries to get information from WMI about a computer system, enabling your package to decide what to do with the other tasks in the package.
WMI Event Watcher Task	Watch for WMI events by running WQL queries to specify the events of interest.

SQL Server Tasks

These tasks help in accessing and performing functions such as Copy, Insert, Delete, and Modify on SQL Server objects and data.

Control Flow Task	Description
Bulk Insert Task	Copy large amounts of data into SQL Server. This is the fastest method for importing data into SQL Server if you don't have to perform transformations while importing. You will use this task in a Hands-On exercise later in the chapter.
Execute SQL Task	Run SQL statements or stored procedures to perform operations such as create, alter, truncate, or drop tables; save a result set of a query or stored procedure into a variable; and so on. You have already used this task in earlier Hands-On exercises.

Scripting Tasks

These tasks help in extending the package functionality by allowing you to write your own code.

Control Flow Task	Description
Script Task	Write code to perform functions that otherwise cannot be performed using built-in tasks and transformations. Uses Visual Studio Tools for Applications (VSTA) Environment as its engine, for writing and running scripts with Microsoft Visual Basic 2008 or Microsoft Visual C# 2008. Scripting is covered in detail in Chapter 11.

Analysis Services Tasks

These tasks allow you to work with Analysis Services objects. These tasks are covered in detail in Chapter 12 while discussing data warehousing practices.

Control Flow Task	Description
Analysis Services Processing Task	Process Analysis Services objects such as cubes, dimensions, and mining models. You can process multiple objects in a batch and choose a processing sequence or can process them in parallel.
Analysis Services Execute DDL Task	Create, alter, or drop multidimensional objects (such as cubes and dimensions) or mining models. You create data definition language (DDL) statements in Analysis Services Scripting Language (ASSL) framed in an XML for Analysis Services (XMLA) command.
Data Mining Query Task	Run prediction queries in Analysis Services using defined data mining models. The prediction query is created using Data Mining Extensions (DMX) language, which provides support for using mining models.

Transfer Tasks

SQL Server Integration Services enables you to transfer databases, error messages, SQL Server Agent jobs, transfer logins, database objects, and master stored procedures using the built-in transfer tasks.

Control Flow Task	Description
Transfer Database Task	Move or copy databases from one instance to another instance of SQL Server. You can transfer database in an offline (Detach/Attach method) or online mode.
Transfer Error Messages Task	Transfer SQL Server error messages from a source SQL Server to a destination SQL Server.

Control Flow Task	Description
Transfer Jobs Task	Transfer SQL Server Agent jobs from one instance of SQL Server to the other instance.
Transfer Logins Task	Transfer logins from a source SQL Server to a destination SQL Server. You can choose to transfer all logins on the server, transfer all logins for the selected database, or transfer just the selected logins.
Transfer SQL Server Objects Task	Copy database objects between databases on a source SQL Server and a destination SQL Server. You can choose to transfer indexes, primary keys, secondary keys, triggers, users, logins and roles, and so on.
Transfer Master Stored Procedures Task	Transfer the stored procedures saved to the master database in a source SQL Server to a destination SQL Server.

Maintenance Tasks

SSIS has built-in tasks for administrative functions for databases, such as backing up and shrinking SQL Server databases, rebuilding and reorganizing indexes, and running SQL Server Agent jobs. Though these tasks are intended for maintaining SQL Server 2000 and SQL Server 2005 databases, they can also be used within SSIS packages.

Control Flow Task	Description
Back Up Database Task	You can perform different types of backups on one or more databases with this task.
Check Database Integrity Task	Check the allocation and structural integrity of all the database objects or indexes in the specified database. This task runs the **DBCC CHECKDB** statement.
Execute SQL Server Agent Job Task	Runs the jobs already created within the SQL Server Agent. Allows you to reuse the jobs already created in SQL Server Agent and provides a facility to perform administration from within a SSIS package.
Execute T-SQL Statement Task	Run T-SQL statements from within a SSIS package. This task is similar to the Execute SQL task, except this task supports only Transact-SQL version of SQL.
History Cleanup Task	Deletes historical data for the specified time period from MSDB tables related to backup and restore activities, SQL Server Agent jobs, and database maintenance plans.
Maintenance Cleanup Task	Deletes backup files or maintenance plan text reports based on the specified time period.
Notify Operator Task	Sends the notifications messages to SQL Server Agent operators via e-mail, pager, or Net Send communication channels.

Control Flow Task	Description
Rebuild Index Task	Rebuilds indexes in SQL Server database tables and views.
Reorganize Index Task	Reorganize indexes in SQL Server database tables and views with this task.
Shrink Database Task	Reduce the size of SQL Server database data file and log file with Shrink Database task. This task runs a **DBCC SHRINKDATABASE** command.
Update Statistics Task	Update information about the distribution of key values for one or more statistics groups in the specified table or indexed view with this task.

Backward Compatibility Tasks

Data Transformations Services (DTS) has been deprecated though backward compatibility support has still been provided. The following tasks and DTS 2000 will be removed in future versions of SSIS, so you should not be developing any new functionality or software code with these tasks. They have been provided purely to support the packages that are still to be migrated to Integration Services.

Control Flow Task	Description
ActiveX Script Task	Run existing DTS 2000 ActiveX code in SSIS packages. This task has been marked deprecated in the current SQL Server version; upgrade your scripts to use more advanced features provided by the Script task. Refer to Chapter 14 for more details on migration options.
Execute DTS 2000 Package Task	Run packages that were developed using the Data Transformation Services of SQL Server 2000 and include legacy packages in your Integration Services package. The task is covered in detail in Chapter 14.

Custom Tasks

The object model of Integration Services provides facilities you can use to extend and program any component or complete package by writing custom code. Whether you want to extend the functionality of a component that will be used only in a package or you want to reuse the developed functionality in various packages throughout your enterprise, you will use different techniques to extend and program Integration Services. The Script task provided in the control flow and the script component provided in the data flow environments allow you to create and program any functionality that is not available in the preconfigured tasks and components of Integration Services using a .NET-compliant programming language such as Visual

Basic or C#. However, these components are difficult to reuse in multiple packages. Alternatively, you can create your own custom tasks using any .NET-compliant programming language such as Visual Basic or C#.

After writing code for the custom task, you can create and register a user interface for the task in the SSIS Designer and reuse the developed and registered custom task or component in your packages as you would use any other built-in component. The development of custom tasks is detailed in Chapter 11.

Control Flow Tasks in Detail

In the preceding section, you've read brief description of each task; now you'll work through the Hands-On exercise or read about the options available through the task's GUI. While working through the details of each task and Hands-On exercise, you will not follow the categorization flow. This is to make it easier to learn and work with the tasks for business scenarios, followed by intuitive and step-by-step methods.

Your first project will require that you download, expand, and archive these zipped files, and then import them to an SQL Server table. This is a simple but generic scenario we all encounter in our day-to-day job functions. So, let's start our journey with the FTP task, which will help us in downloading files from a remote FTP server.

FTP Task

Running an FTP task makes your computer an FTP client and allows you to connect to an FTP server for transferring files. You can drag and drop this task onto the Control Flow Designer surface and double-click to open the FTP Task Editor to configure it. You can select to perform any of the following tasks from the options listed in the Operation field in the File Transfer page:

- **Send Files** Transfers files from the local computer to the remote FTP server.
- **Receive Files** Gets files from the remote FTP server.
- **Create Local Directory** Creates a folder on the local computer.
- **Create Remote Directory** Creates a folder on the FTP server.
- **Remove Local Directory** Deletes the specified folder on the local computer.
- **Remove Remote Directory** Deletes the specified folder on the FTP server.
- **Delete Local Files** Deletes the specified files on the local computer.
- **Delete Remote Files** Deletes the specified files on the FTP server.

It is easy to configure the FTP task as long as you understand how it works. During configuration of the FTP task, you may have to define the RemotePath, the LocalPath, and some other properties for both the local and the remote computers. However, the configuration options available to you change depending upon the type of operation you choose in the Operation field.

The FTP Task Editor connects to the FTP server using the FTP Connection Manager Editor. FTP Connection Manager is defined in the Connection Managers area on the Control Flow Panel in BIDS. In FTP Connection Manager Editor, you specify the server name/IP address, the server port number, the credentials for accessing the FTP server, and the options such as number of retries, time out, and so on. You can either specify the path to the folder on the FTP server directly in the RemotePath field or use a variable to pass the path to the FTP task. The FTP task can access or download multiple files from the FTP server using wildcards; however, when you want to send or upload files, the FTP task works slightly differently, depending on how you specify the LocalPath option. If you use a File Connection Manager in the LocalPath field to connect to a local file, the FTP task can access only one file in this case as the File Connection Manager can access only one file at a time. To send multiple files from the local computer to the ftp server, you need to specify the LocalPath using a variable, by first setting the IsLocalPathVariable property to True (shown later, in Figure 5-2). If you are using a variable, you can use wildcard characters such as * or ? for specifying multiple files. The FTP task behaves the same when you want to delete files from a local folder. So, if you want to send multiple files to an FTP server or want to delete multiple files on the local computer, you must use a variable to specify the LocalPath. An alternate approach could be to place the FTP task inside the Foreach Loop Container to enumerate across multiple files.

Preparations for the Hands-On Exercises in This Chapter

Let's do a Hands-On exercise with the FTP task to help you understand the various options. In this exercise, you will download files from an FTP server. But before you start, make sure you've completed the following steps to prepare yourself for the exercise:

1. Download the software and the files from the McGraw-Hill web site and copy them to C: drive as explained in the Appendix.
2. By now, you should have attached the provided Campaign database to your SQL Server 2005 database server; if you have not attached it, do that now so that you can complete the exercises. Attaching the Campaign database to your SQL Server 2008 is explained in the Appendix.
3. Install an FTP service on a second PC for this exercise, or have access to an FTP server. After that, create a folder called **Sales** on the FTP server root folder and copy the DealerSales01.zip and DealerSales02.zip files from the local C:\SSIS\RawFiles folder to the Sales folder on the FTP server.

Hands-On: Downloading Zipped Files

Dealers of Adventure Works Bikes submit their sales reports in zipped form to an FTP server, and these files need to be downloaded from the FTP server for import into an SQL Server database. You have been tasked with creating a project that downloads these zipped files.

Method

In this exercise, you will use the FTP task to download two files from the remote folder Sales on the FTP server. These files will be downloaded to the C:\SSIS\downloads folder on your local computer.

A couple of points to note: First, if you want to use the Downloading Zipped Files package that has been provided with this book, you will receive an error when opening the package. When you click OK on the pop-up error message, the package will load properly but without the connection string in the FTP task. This is because, by default, the sensitive information (passwords, connection strings, and so on) in the packages get encrypted using the User key (which is my user key in this case), and when another user tries to open the package, it throws an error and subsequently opens the package after removing the sensitive information. However, if you open the Downloading Zipped Files package after you've completed the work in this Hands-On, you will not get any such error.

Second, this package requires a connection to an FTP server. Many FTP sites are available from which you can download a file with this task—such as FTP sites for antivirus updates—and build the package for this exercise in case you don't have access to a test FTP server.

Exercise (Configure FTP Task)

You will be creating a new package here to work with FTP task. This package will be used in a later exercise as a child package.

1. Start BIDS and choose File | New | Project to open a New Project window. In this window, make sure Business Intelligence Projects is selected under Project Types, and then choose Integration Services Project from the Templates pane. Type in the following additional details and click OK to create a new project.

Name	Control Flow Tasks
Location	C:\SSIS\Projects

2. When the blank project is created, go to Solution Explorer in the BIDS and right-click the Package.dtsx package under the SSIS Packages folder; choose

Rename from the context menu. Type **Downloading zipped files.dtsx** to rename the package and click Yes in the Visual Studio confirmation box.

3. Drag the FTP task from the Toolbox onto the SSIS Designer. Double-click the FTP task icon to open the FTP Task Editor. On the General page, specify the following details:

Name	Download Files
Description	This task downloads sales reports files from the FTP Server.

Click in the FtpConnection field to show the down arrow button. Click the down arrow and choose <New Connection…> from the drop-down list. This will open the FTP Connection Manager Editor. In the Server Settings area, type the name of your FTP server (W8KR2, in my case) in the Server Name field and **21** in the Server Port field as shown in Figure 5-1. W8KR2 is the name of the FTP server used in the lab setup for creating the projects used in this book.

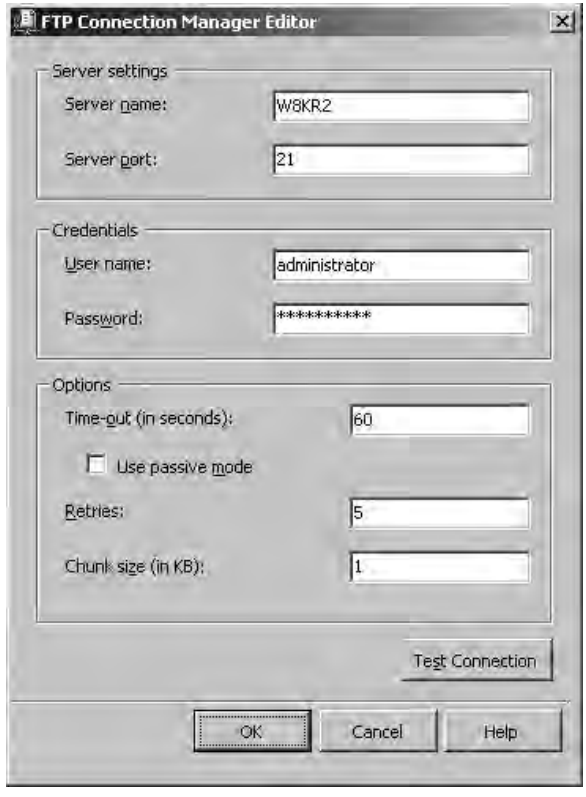


Figure 5-1 *Configuring the FTP Connection Manager*

In the Credentials area, type the user name and password that you can use to connect to the FTP server to download files. Leave other fields set to the default settings. Click the Test Connection button to test the connectivity to the FTP server. If the test reveals a successful connection, click OK twice to close the message window and the FTP Connection Manager Editor.

For more details on the FTP Connection Manager, refer to the discussion in Chapter 3.

4. Go to the File Transfer page from the left pane in the FTP Task Editor.
5. Select Receive Files in the Operation field. Note that the available options change with the choice of operation selected.
6. Select True for the IsTransferAscii field. For the benefit of those who are new to using FTP to download files, FTP uses two modes to transfer files—ASCII and Binary. ASCII mode is used to transfer text files—i.e., files that contain letters, numbers, and simple punctuation characters. Binary files, on the other hand, are structured differently and hence require a different mode of transfer—i.e., Binary mode. Examples of binary files include images, executable files, and algorithmically generated packages.
7. In the Remote Parameters section, click in the RemotePath field and you will see an ellipsis (…) button on the far-right corner. Click this ellipsis button to open Browse For File window. This window shows the directory structure of the remote FTP server. Choose the path where you copied the files earlier while preparing for this exercise; or, if you have copied the files to the Sales folder on the FTP server, choose the /Sales folder and then choose the DealerSales01.zip file and press OK. You will see /Sales/DealerSales01.zip listed in the RemotePath field. You want to select both files listed in the folder, but you can't do that using the graphical interface. The FTP task allows the use of wildcard characters such as * and ? to specify filenames to enable you to select multiple files. Change the preceding path in the RemotePath field to **/Sales/*.zip** to select both the files.
8. In the Local Parameters section, click in the LocalPath field, click the down arrow, and choose <New Connection…> from the drop-down list. This will open the File Connection Manager Editor for you to specify an existing folder path where you want to download the files. Click the Browse button next to the Folder field and select the C:\SSIS\downloads folder by browsing to the correct path. Create this folder if this doesn't exist already. Click OK and you will see *downloads* listed in the LocalPath field. You should also be able to see a File Connection Manager *downloads* created under the Connection Managers area (see Figure 5-2).
9. Select True for the OverwriteFileAtDest field, since you will be running this package many times. However, while configuring this task for your production servers, carefully consider using this option.

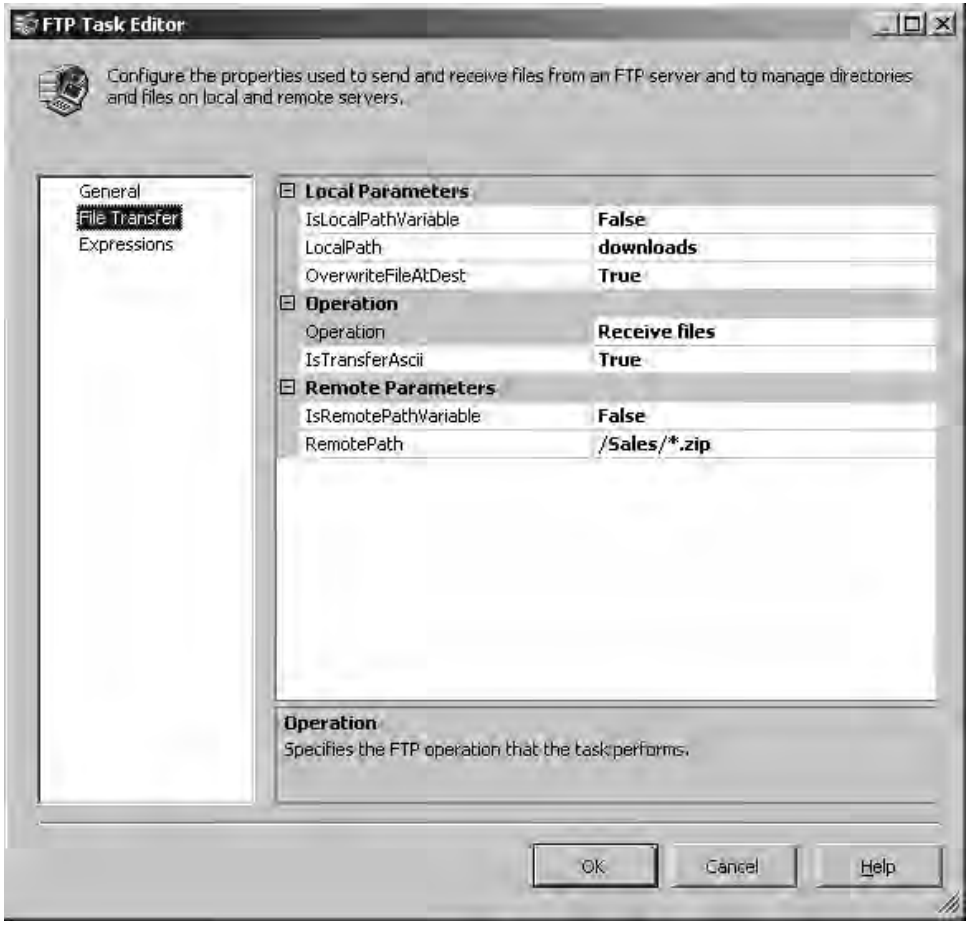


Figure 5-2 *Configuring the FTP task in the Task Editor*

10. Click OK to complete the configuration and close the editor. Press F5 to run the package and you will see that the task turns yellow for a while; once the file downloading is completed, it will turn green. (If your network firewall or local computer's firewall [for instance, the Windows firewall] is blocking the outbound connection to the FTP site, you may have to allow this connection by changing your firewall settings. See your network administrator for help on changing firewall settings.)
11. Navigate to the C:\SSIS\downloads folder and you'll notice that the two Zip files have been downloaded to that folder.
12. Press CTRL-SHIFT-S to save all the files in this solution and then choose File | Close Project.

Review

In this Hands-On exercise, you learned to use the FTP task to download multiple files from a remote FTP server. You have hard-coded file paths for the RemotePath and LocalPath fields in this exercise. However, you can use variables to change these folders dynamically during run time and download or upload files to different folders each time the package is run. For example, you can download files to the folders derived on the basis of date—for instance, the folder name includes the date as a part of the name (such as Zip20091011). So, in that case, when the package is run, the path variable is calculated dynamically based on the current date and loads the files to the current date folder. Variables are not the only means to derive values dynamically. SSIS enables you to derive values dynamically at run time, based on the outcome or values generated by other tasks, using property expressions as well that do not necessarily contain variables in their expressions. You have been introduced to the property expressions in Chapter 3 and will learn more about them in Chapter 8.

Execute Process Task

Generally enterprises have home-grown or custom-developed applications to apply business processes to the information they deal with. These applications might be to make sure that the information goes to right people in the right format after corporate standards are applied on to it, or it might be a custom application to transform a mainframe encrypted and compressed (such as EBCDIC format) file into a standard text format file. The company might wish to reuse the functionality that already exists and has been accepted within the corporate setting. Integration Services allows you to leverage your existing investment and bring those processes in the workflow within the packages by using the Execute Process task. With this task, you can run a business application or a batch file to perform specific business functions that already exist or can't be developed easily in SSIS; the good thing about this is that the output from running the business application or batch file can then be included in the SSIS package workflow. You could unzip files, call custom programs to break a huge file into smaller files that can be easily handled for transformation operations, run a custom report generation application from within SSIS, and then use a Send Mail task to distribute those reports. To keep it simple, we will use this task to unzip the files you have downloaded from the FTP server in the preceding Hands-On exercise.

You have begun your journey to understand the SSIS workflow. In the last Hands-On exercise, you downloaded zipped files from an FTP server in the C:\SSIS\ downloads folder. In the next exercise, you will unzip the downloaded files using the Execute Process task, which will run a batch file to do the trick.

Hands-On: Expanding Downloaded Files

Adventure Works Bikes dealers' reports need to be downloaded and extracted so that they can be imported into the SQL Server database. You have already downloaded these files; here you will extract flat files from the downloaded files.

Method

You have downloaded zipped files DealerSales01.zip and DealerSales02.zip in the C:\SSIS\downloads folder but can't use these files, as SSIS can't read zipped files. You need to extract flat files from the zipped files to be able to import them into SQL Server using SSIS. You will run a batch file to extract the flat files from the zipped files and will deploy an Execute Process task to run this batch file at run time. Because you want to unzip more than one file, you will enumerate the files you want to unzip; an enumeration function will be provided by a Foreach Loop Container. In the last chapter, you used a Foreach Loop Container with the Foreach ADO enumerator to enumerate over rows; now you will be using the Foreach File enumerator to enumerate over files. The steps for this will be:

- Create a batch file to expand the downloaded Zip file.
- Use a Foreach Loop Container to enumerate over multiple Zip files.
- Add an Execute Process task to call a batch file from within the SSIS package to extract the flat files.

Exercise (Creating Batch File)

The downloaded files have been compressed using a Freezip freeware package that has been included in the distribution software provided for this book. Go to C:\SSIS\Freeware\Freezip to find the software files there. However, if you use some other software utility to compress files, you can use a command-line interface of that utility to unzip files in the same manner as shown in this exercise.

Open a blank text file using Notepad and type the following commands:

```
DEL C:\SSIS\downloads\%1.txt
C:\SSIS\Freeware\Freezip\UNZIP %1.zip %1.txt
```

The first line deletes the previously present text file, and the second line extracts a text file from the Zip file specified as an argument in the command. Save the file as C:\SSIS\RawFiles\ExtractFiles.bat. Make sure that the file has been saved with the .bat extension and not with the .txt extension. This file should already be available in the RawFiles folder, as it has been included in the distribution software provided for this book.

Exercise (Enumerating Multiple Zipped Files)

You will add and configure a Foreach Loop Container to enumerate over the Zip files in the downloads folder and populate the *fname* variable with the filename.

1. Start BIDS and open the Control Flow Tasks project. Go to Solution Explorer and right-click the SSIS Packages folder in the Control Flow Tasks project and choose New SSIS Package from the context menu. This will add a new package called Package1.dtsx in the project.
2. Right-click Package1.dtsx and rename it as **Expanding downloaded files.dtsx**.
3. Drop a Foreach Loop Container from the Toolbox onto the designer surface and open the editor by double-clicking it. Type the following in the General page:

Name	Enumerating Zip Files
Description	This task enumerates Zip files in the C:\SSIS\downloads folder.

4. Go to Collection page and make sure Foreach File Enumerator is selected in the Enumerator field.
5. In the Enumerator configuration area, specify C:\SSIS\downloads in the Folder field either by typing it in directly or by using the Browse button to select the folder. You can enumerate multiple files of the same type with the Foreach Loop Container at one time. For this, you can use the Files field to limit or select precisely the files of one type you want to access in a folder containing different types of files. In the Files field, type ***.zip** to specify the types of files you want the Foreach Loop Container to enumerate on (refer Figure 5-3). Use the Files field to limit or select precisely the files you want to access in a folder containing different types of files.
6. The most interesting bit in this page is the Retrieve File Name section. This is where you will choose how the filename should be retrieved from three options. Before you jump into selecting one of the options, you must understand what these options are and how they work. The filename is treated in SSIS as consisting of three parts: the path pointing to the location where the file might be stored, the name portion of the file, and the extension of the file indicating its type.

Fully Qualified	This option will return the path, the filename, and the extension all as a single string. The path portion of the filename can be a full folder path in a universal naming convention (UNC) or absolute form. For example, the fully qualified name retrieved can be *ComputerName*\Sales\DealerSales01.zip, or it can be in the form C:\SSIS\downloads\DealerSales01.zip, depending on what you have specified in the Folder field.
Name And Extension	Choosing this option returns the name portion of the filename along with its extension—for example, DealerSales01.zip.
Name Only	Choosing this option will return only the name portion of the filename—for example, DealerSales01.

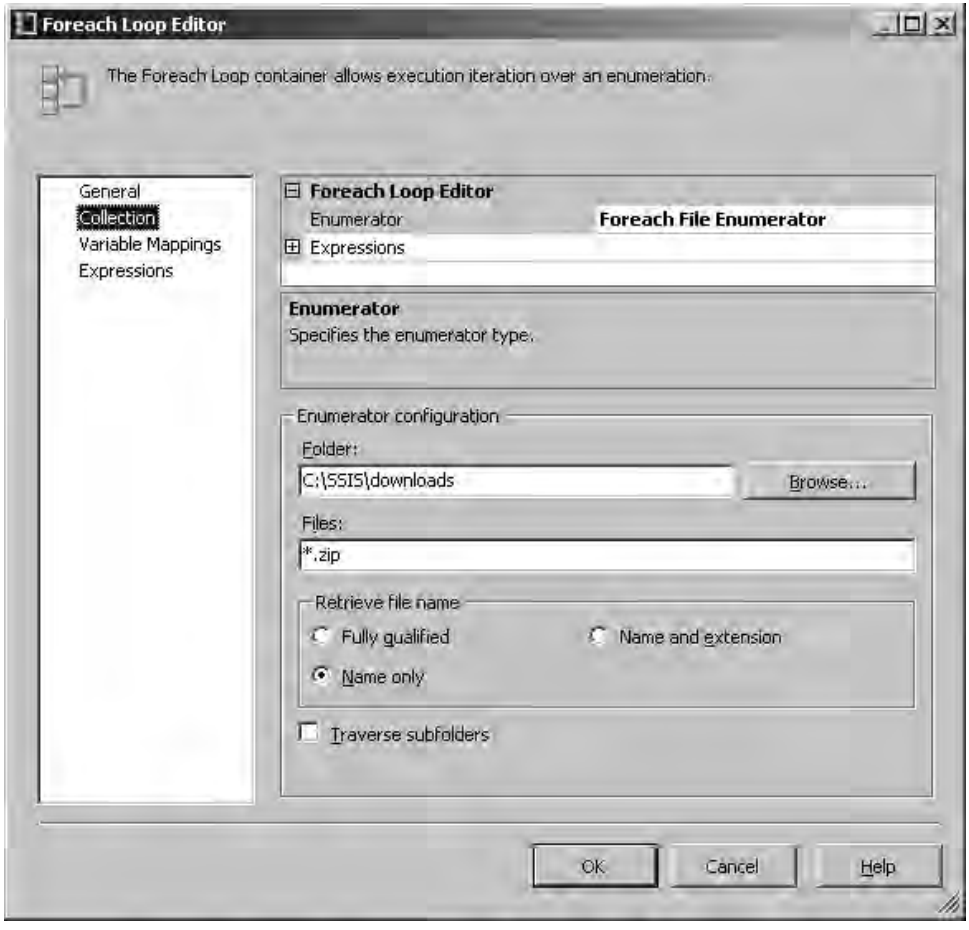


Figure 5-3 *Configuring the Foreach Loop Container with the Foreach file enumerator*

Select Name only, as you will need only the name of the file to be passed as an argument to the batch file.

7. Go to the Variable Mappings page, click in the Variable field, and select <New Variable…> to create a new variable. Name the variable **fname** in the Expanding Download Files Container with the **string** type value and return to the Foreach Loop Editor. Note that the Index field has automatically been allocated a value of *0* for this variable. Make sure that you type the variable name in lowercase—*fname*—as the variables are case sensitive. Click OK to complete the Foreach Loop Container's configuration.

Exercise (Calling Batch File Using Execute Process Task)

Now you will add Execute Process task inside the Foreach Loop Container. This task will use the *fname* variable populated by Foreach Loop Container to pass as an argument to the ExtractFiles.bat file.

8. Drag and drop the Execute Process task from the Toolbox within the Foreach Loop Container and double-click it to open the Execute Process Task Editor. Type the following in the General page of the task editor:

Name	Call ExtractFiles batch file
Description	This task extracts DealerSales flat files.

9. Go to Process page to configure options for this task. Verify that the RequireFullFileName field has the default True selected. This means that the task will fail if it cannot find the batch file at specified location.
10. In the Executable field, specify C:\SSIS\RawFiles\ExtractFiles.bat to point the task to the batch file.
11. You need to specify the filename as an argument to the batch file. The filename argument has been populated in the *fname* variable in the earlier part of this exercise. However, Arguments field doesn't allow use of a variable, so you will be using property expressions in the next step to attach the fname variable to the Arguments field. Leave the Arguments field blank for the moment and in the WorkingDirectory field, specify **C:\SSIS\downloads**, as shown in Figure 5-4.

    In the next three fields, you can use variables for the input, output, or error values as explained here:

    - **StandardInputVariable** Specify a variable to provide the input to the process.
    - **StandardOutputVariable** Specify a variable to capture the output of the process.
    - **StandardErrorVariable** Specify a variable to capture the error output of the process.

    You are not using these fields for this exercise, so leave them blank. Using the FailTaskIfReturnCodeIsNotSuccessValue field, you can configure the task to fail if the process exit code does not match the value you provide in the SuccessValue field. Also, you can specify a time-out value in seconds and choose to terminate the task after time-out period. Finally, if it is important, you can choose the window style in which the task starts a process. Leave these fields set at their default values and move on to the Expressions page.

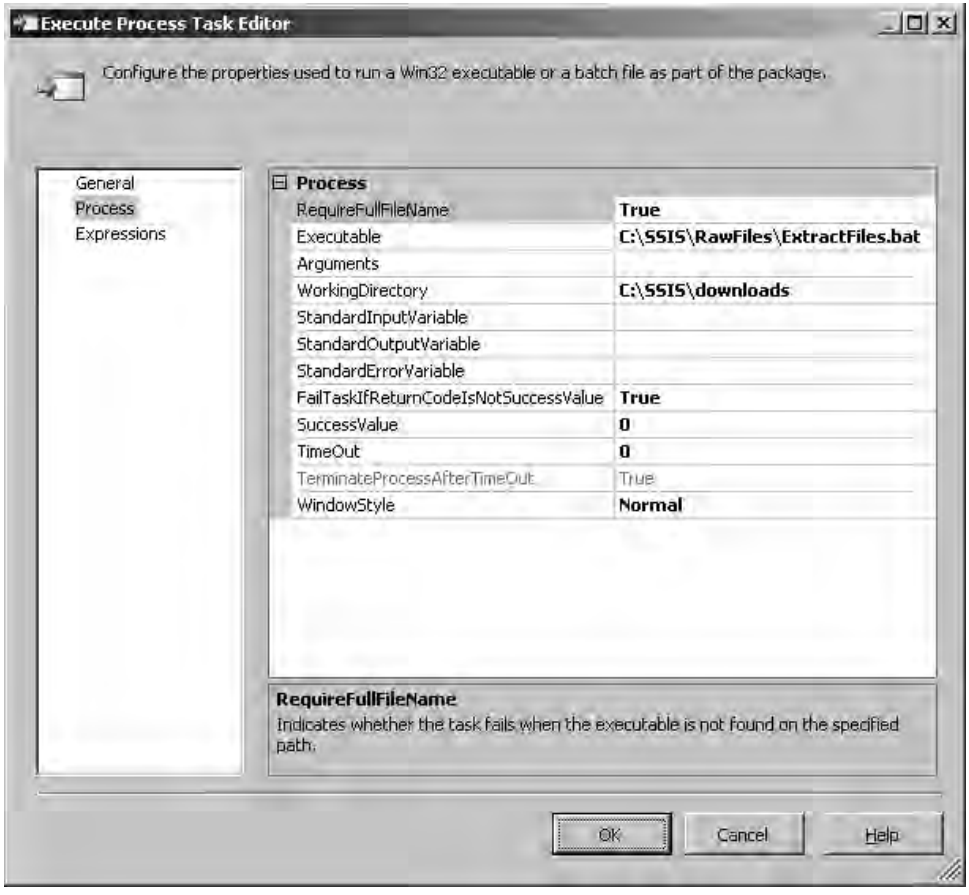


Figure 5-4 *Configuration of the Execute Process task*

12. In this step, you will pass the value of the *fname* variable to the Arguments field using property expressions. Property expressions are explained in detail in Chapter 8, but here for a quick introduction, property expressions will apply the run-time values of the fname variable on to the Arguments property. Click the ellipsis button in the Expressions field to open the Property Expressions Editor. Click in the Property field and select Arguments from the drop-down list. Click the ellipsis button in the Expression field to open an Expression Builder dialog box. Expand Variables in the left pane of the window and locate the *User::fname* variable in the list. Drag this variable and drop it in the Expression box. Click OK three times to close the task editor.
13. Press F5 to start debugging the task. You should see the tasks turning quickly from yellow to green (see Figure 5-5).

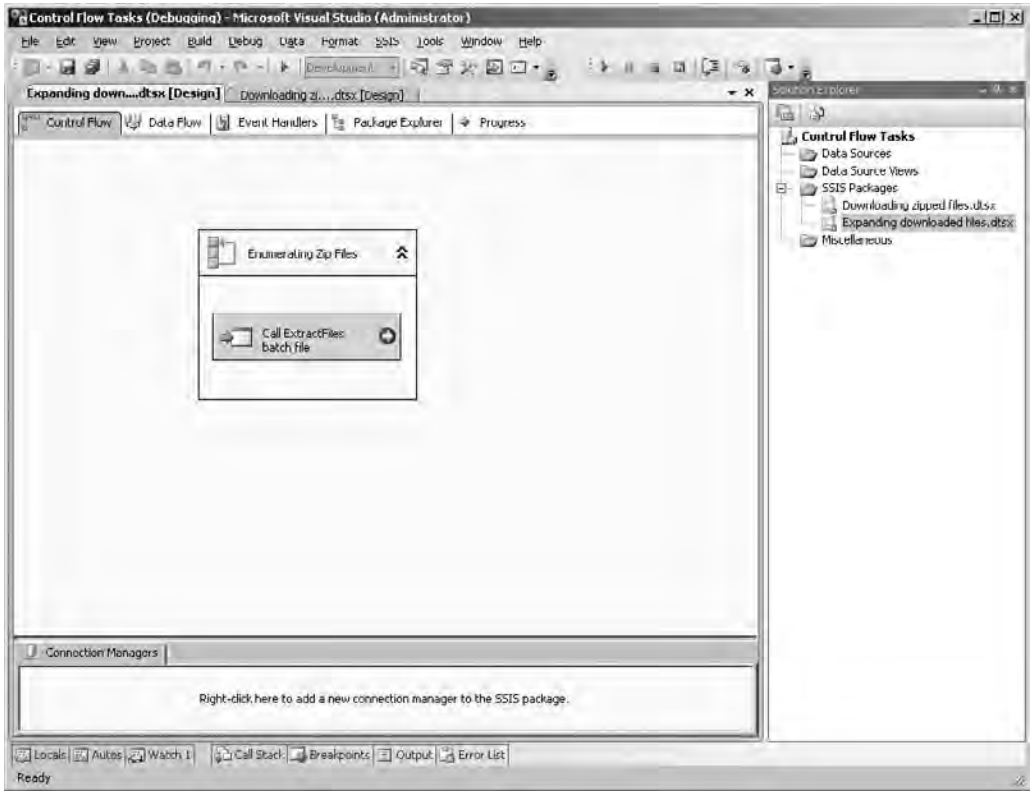


Figure 5-5 *Executing the Expanding downloaded files.dtsx package*

13. Press CTRL-SHIFT-S to save all the files in this project and then choose File | Close Project.
14. Go to the downloads folder and see that the DealerSales01.txt and DealerSales02 .txt files have been extracted.

Review

You have learned how to use the Execute Process task to expand the zipped files in this Hands-On exercise. Now you know that SSIS provides an option to run the custom-built applications that you need to use for your day-to-day functions. Also, not only can you run custom-built applications, but you can also use the results of running such an application by using the StandardOutputVariable property and use this value in other places in the package such as building a workflow expression.

File System Task

The File System task allows you to perform operations on the files and directories in the file system. You will be using this task in your packages to perform operations such as copy, delete, and move on files and folders. If you have used DTS 2000, you might have written ActiveX code to rename, copy, or move files or folders. Though you may still have to write a piece of code in the Script task to perform some of the complex operations, most of your requirements for performing operations on files and folders can be met by using the File System task and hence less code to maintain.

In the Task Editor of File System task, you will see an Operation field that offers a drop-down list of operations it can perform. Following are the brief descriptions of the operations that the File System task can perform:

- **Copy directory** Copies a folder from one location to another.
- **Copy file** Copies a file from one location to another.
- **Create directory** Creates a directory in a specified location.
- **Delete directory** Deletes a directory from a specified location.
- **Delete directory content** Deletes all the files and folders in the specified directory.
- **Delete file** Deletes the specified file.
- **Move directory** Moves a folder from one location to another.
- **Move file** Moves a file from one location to another location.
- **Rename file** Renames a file in the specified location. You can actually move the file to a new location while renaming by specifying a new path in the name, though you cannot rename the file while moving it to a new location.
- **Set attributes** Sets Archive, Hidden, ReadOnly, and System attributes on the files and folders. If you do not specify any attribute in your selections, the Normal attribute will be applied, which is exclusive to all the other attributes.

The File System task has a dynamic layout in which the available fields change depending on the operation you choose. For example, the Delete Directory, Delete Directory Content, and Delete File Operations do not use a destination, and hence the DestinationConnection field doesn't appear when you choose any of these operations. The SourceConnection, however, is used by all of the operations. The SourceConnection and the DestinationConnection can be specified by using a File Connection Manager to refer to a file or a folder or by providing a variable that contains the connection strings.

The following Hands-On exercise for this task will help you understand the configuration options. In the earlier exercises, you have downloaded and unzipped the dealer sales files and hence have compressed Zip files and uncompressed text files in the C:\SSIS\downloads folder. Your next problem is to archive the Zip files for historical purposes.

Hands-On: Archiving Downloaded Files

The downloaded reports of dealers of Adventure Works Bikes are to be archived for historical purposes. You want to keep the downloaded zipped files in the Archive folder.

Method

In this exercise, you will use the Foreach Loop Container and the File System task to copy these files one by one to the C:\SSIS\downloads\Archive folder. The step-by-step method is as follows:

- Configure the Foreach Loop Container to enumerate over files and house the File System task in it. Your File System task will copy one file with each iteration to the Archive folder.
- Configure the File System task to move files to the Archive folder one by one with each iteration of the Foreach Loop Container.

Exercise (Configure Foreach Loop Container)

In first part of the exercise you will configure the Foreach Loop Container to enumerate files using Foreach File enumerator and populate a variable with the filename. This variable will be used by File System task to determine the name of the file to be copied over.

1. Open the Control Flow Tasks project in BIDS. Go to the Solution Explorer, right-click the SSIS Packages folder, and choose New SSIS Package from the context menu. This will add a new package, Package1.dtsx, to the project.
2. Right-click Package1.dtsx and rename it as **Archiving downloaded files.dtsx**.
3. From the Toolbox, drag and drop the Foreach Loop Container onto the Control Flow panel.
4. Make sure that the Foreach Loop Container is selected; open the Variables window from the default position on the left side of the BIDS interface. Alternatively, click the Foreach Loop Container and choose View | Other Windows | Variables.

Click Add Variable, the left-most button in the toolbar of the Variables window. Specify the following for this variable:

Name	Fname
Scope	Foreach Loop Container
Data Type	String
Value	Xyz

The value you've assigned to fname variable will be changed at run time when the Foreach Loop Container reads the filename and populates this variable during each iteration. The value assigned is just a placeholder value, so you can realize the Fname variable changes values dynamically. Last, be careful with the case of the Name field, as SSIS variables are case-sensitive.

5. Right-click the Foreach Loop Container and choose Edit from the context menu to open the Foreach Loop Editor. In the General page of the container, type the following:

Name	Enumerating Files
Description	This container enumerates the files and populates the fname variable.

6. Go to the Collection page. Make sure it shows the Foreach File Enumerator in the Enumerator field and specify the options shown in Figure 5-6.

Enumerator	Foreach File Enumerator
Folder	C:\SSIS\downloads
Files	*.zip
Retrieve file name	Fully qualified

7. Move on to Variable Mappings page. You will map fname variable to the filename retrieved by the Foreach Loop Container so that later on in the package you can get the filename by the fname variable. Click in the highlighted area under the Variable column and then click the drop-down arrow that appears. From the drop-down list, select the variable *User::fname*, configured earlier in the exercise. A *0* will appear under the Index field. Index indicates the position of the string within the collection. As you will be accessing only one filename at a time, Index will be set to 0. Click OK to close the Foreach Loop Editor.

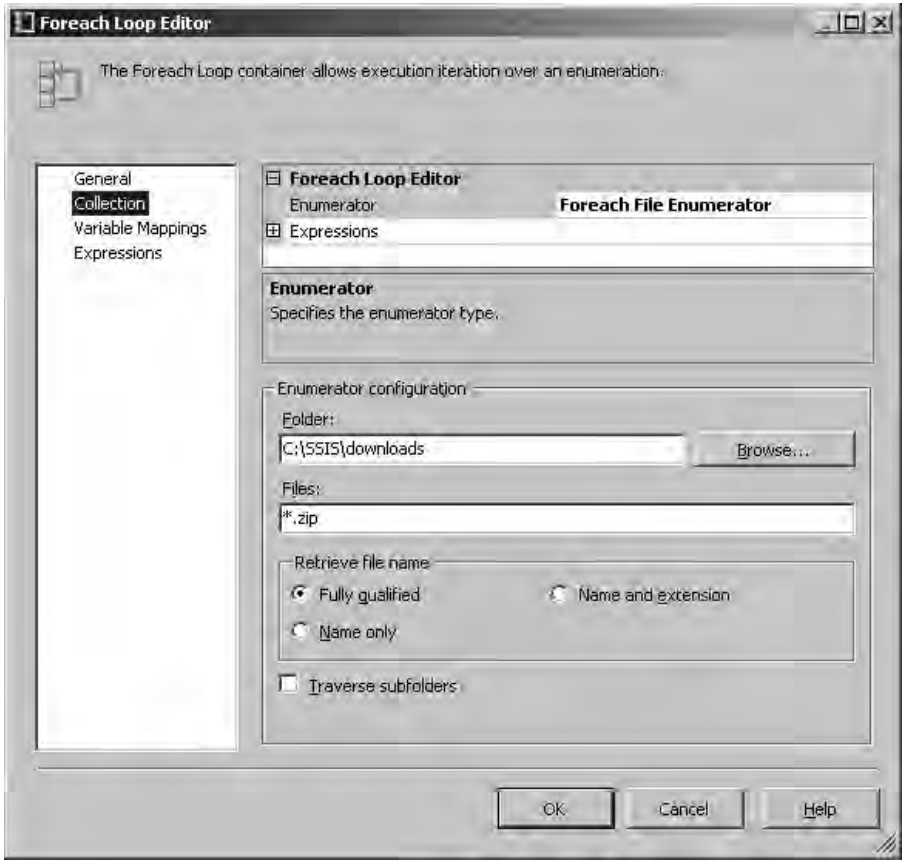


Figure 5-6 *Configuring the Foreach Loop Container to enumerate fully qualified filenames*

Exercise (Configure File System Task)

In this final part, you will configure the File System task to move the downloaded zipped files from the C:\SSIS\downloads folder to the C:\SSIS\downloads\Archive folder. This task will move the files one by one with each iteration of the Foreach Loop Container. It will use the variable *User::fname*, populated by Foreach Loop Container, to determine the source filename.

8. Drag and drop the File System task from the Toolbox within the Enumerating Files Container.
9. Double-click the File System Task icon to open the File System Task Editor. Select Move File in the Operation field. In the General area on the right pane, fill in the following details:

Name	Archive downloaded files
Description	This task copies downloaded files from the 'downloads' folder to the 'Archive' folder.

10. In the Source Connection section, set IsSourcePathVariable to True.
11. Click in the SourceVariable field and then click the down arrow to see the drop-down list. Choose User::fname as shown in Figure 5-7.
12. In the Destination Connection section, verify that IsDestinationPathVariable is set to False.

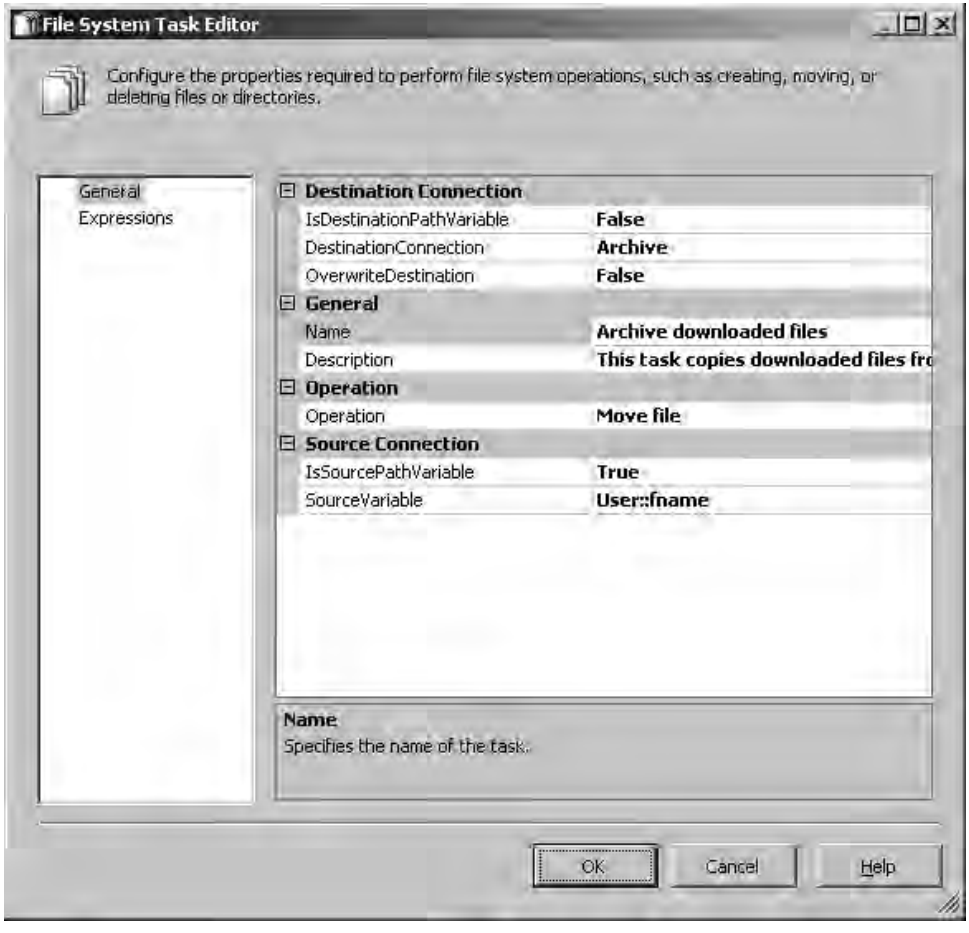


Figure 5-7 *Configuring the File System task for moving files*

13. Click in the DestinationConnection field and then click the down arrow to see the drop-down list. Choose the <New Connection…> to open File Connection Manager Editor. In the Usage type field, select Existing Folder and type **C:\ SSIS\downloads\Archive** in the Folder field. Note that a File Connection Manager named Archive has been created.
14. The OverwriteDestination field allows you to overwrite the files with the same name at destination folder. Be mindful while configuring this option in the production environment. Leave it set at the default value of False. Click OK to close the File System Task Editor.
15. Now that your package is ready to be run, press F5 on the keyboard to run the package and notice how the Enumerating Files Container changes from yellow followed by Archive Downloaded Files task changing from yellow to green. This cycle is repeated twice before both the objects stop processing and turn green to declare success of the operation. Each time Archive Downloaded Files task changes color from yellow to green, one file has been moved. Stop debugging the package by pressing SHIFT-F5.
16. Run Windows Explorer to check that the files have been moved from C:\SSIS\downloads folder to the Archive subfolder in this directory.
17. Press CTRL-SHIFT-S to save all the files in this solution and then choose File | Close Project.

Review

You have configured the Foreach Loop Container to enumerate over files in a folder and pass the filenames via a variable to the File System task. The variable passed by the Foreach Loop Container was used to set the source filename in the File System task, which was configured to move files from a dynamic source to the hard-coded destination Archive folder. In this exercise, you have seen the functionality provided by SSIS components to run in synchronization, where one component was reading the files one by one and passing the information to the other component that was moving those files to a different folder as it receives the filenames from the parent container.

Web Service Task

You can read data from a Web Service method and write that data to a variable or a file using the Web Service task. For example, you can obtain a list of postal codes from the local postal company, write it to a flat file using the Web Service task, and then do the lookup against this postal codes file to clean or standardize your data at loading time.

Web Service task uses the HTTP Connection Manager to connect to the web service. HTTP Connection Manager specifies the server URL, user credentials, optional client certificate details, time-out length, proxy settings, and so on.

The Web Service Description Language (WSDL) is an XML-based language used for defining web services in a WSDL file, which lists the methods that the web service offers, the input parameters that the methods require, the responses that the methods return, and how to communicate with the web service. Thus, a web service requires a WSDL file to get details of settings to communicate with another web service. The HTTP Connection Manager can specify in the Server URL field a web site URL or a WSDL file URL. If you specify the WSDL file URL in the Server URL field, the computer can download the WSDL file automatically. However, if you are specifying the web site URL, you must copy the WSDL file to the local computer.

XML Task

Whenever you are working with XML data, you will be most likely using the XML task to perform operations on the XML documents. This task is designed to work with the XML documents from the workflow point of view, whereas if you want to bring XML data, i.e., the content of an XML document in the data flow, to apply transformations, you will be using the XML Source adapter while configuring your Data Flow task. The XML Source adapter is available in the Data Flow Sources section in the Toolbox when you're working with the Data Flow task on the Data Flow panel.

Using the XML task, you can perform the following operations on XML documents:

1. Retrieve XML documents and dynamically modify those documents at run time.
2. Select a segment of the data from the XML document using XPath expressions similar to how you select data using an SQL query against database tables.
3. Transform an XML document using XSLT (extensible stylesheet language transformations) style sheets and make it compatible with your application database.
4. Merge multiple documents to make one comprehensive document at run time and use it to create reports.
5. Validate an XML document against the specified schema definition.
6. Compare an XML document against another XML document.

The XML task can automatically retrieve a source XML document from a specified location. To perform this operation, the XML task can use a File Connection Manager, though you can directly enter XML data in the task or specify a variable to access the XML file. If the XML task is configured to use a File Connection Manager, the connection string specified inside the File Connection Manager provides the information of the path of the XML file; however, if the XML task is configured to use a variable, the

specified variable contains the path to the XML document. At run time, other processes or tasks in the package can dynamically populate this variable. Like the retrieval process of XML documents, the XML task can save the result set after applying the defined operation to a variable or file. By now, you can guess that to write to a file, the XML task will be using a File Connection Manager.

The XML Task Editor has a dynamic configuration interface that changes depending upon the type of operation you choose to apply to the XML documents. Following are the descriptions of these configuration areas.

Input Section

As mentioned, the XML task can retrieve the source document that is specified under the Input section in the XML Task Editor. You can choose from three available SourceType options: Direct Input allows you to type in XML data directly in the Source field; File Connection allows you to specify a File Connection Manager in the Source field; and Variable allows you to specify a variable name in the Source field.

Second Operand Section

This section defines the second document required for the operation to be performed. The type of second document depends on the type of operation. For example, the second document type will be an XML document if you are merging two documents, while the second document will be an XSD (XML Schema Definition) document if you are trying to validate an XML document against an XSD schema. Again, like the Input section, you can choose between the three types—Direct input, File connection, and Variable—in the SecondOperandType field and, based on your choice, specify the document details in the SecondOperand field.

Output Section

In this section, you specify whether you want to save the results of the operation performed by running the XML task. You can save the results to a variable or a file by using the File Connection Manager to specify the destination file. You can also choose to overwrite the destination.

Operation Options Section

This section is dynamic and changes with the option selection. For example, for a Diff operation, this section will change to the Diff Options section (see Figure 5-8), and for Merge operation, this will become the Merge Options section with its specific fields relevant to the operation. The two operations XSLT and Patch do not have this section at all.

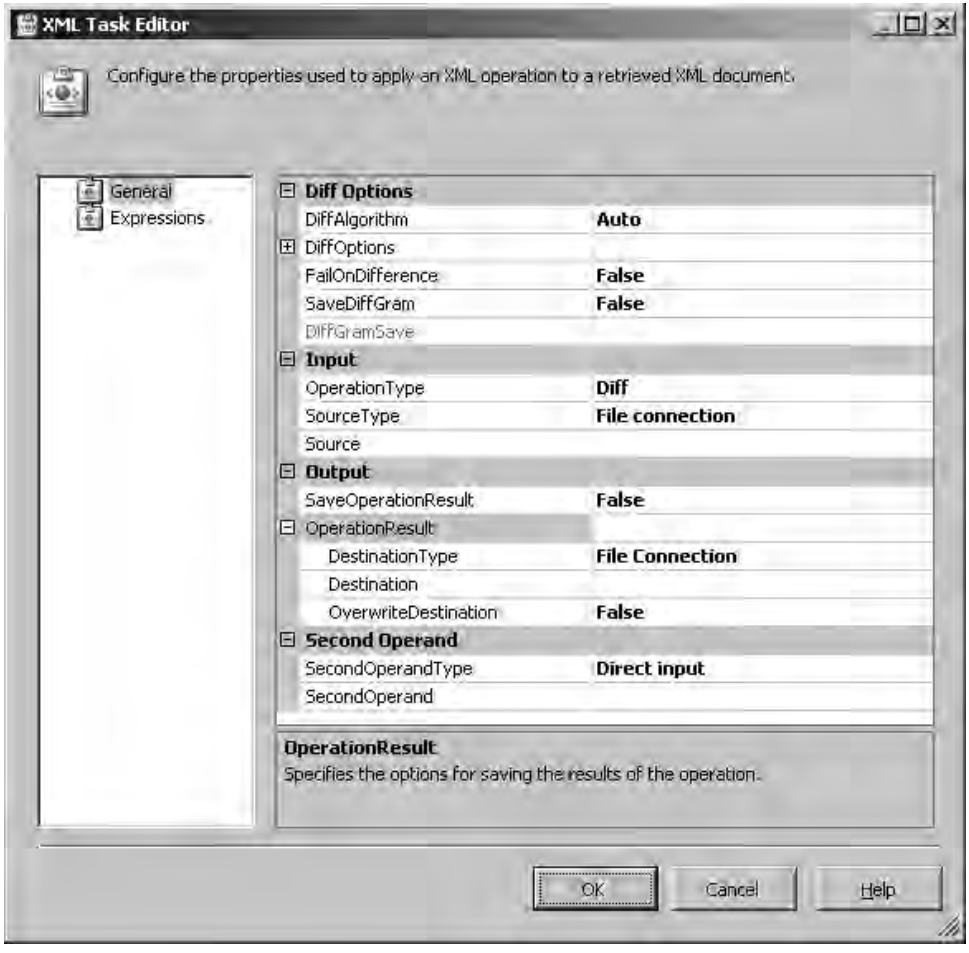


Figure 5-8 *The XML Task Editor*

The XML task has six predefined operations for you to use. The configuration layout of the options changes as soon as you select a different operation.

Validate

You can validate the XML document against a Document Type Definition (DTD) or XML Schema Definition (XSD) schema. The XML document you want to validate is specified in the Input section in the Editor, and the schema document is specified in the Second Operand section. The type of schema document depends upon what you specify for ValidationType—XSD or DTD. With either type of ValidationType, you can choose to fail the operation on a validation failure in the FailOnValidationFail field.

XSLT

You can perform XSL transformations on the XML documents using XSLT style sheets. The Second Operand should contain the reference to the XSLT document, which you can type directly into the field or specify by using either the File Connection Manager or a variable.

XPATH

Using this operation, you can perform XPATH queries and evaluations on the XML document. The Second Operand should contain a reference to the second XML document, which you can type directly into the field or specify by using either the File Connection Manager or a variable. You can select the type of XPATH operation in the XPathOperation field. The XPathOperation field provides three options.

- **Evaluation** Return the results of an XPath function such as sum().
- **Node list** Return the selected nodes as an XML fragment.
- **Values** Return the results in a concatenated string for text values of all the selected nodes.

Merge

Using this operation, you can merge two XML documents. This operation adds the contents of the document specified in the Second Operand section into the source document. The operation can specify a merge location within the base document.

One thing to note here is that the XML task merges only the documents that have Unicode encoding. To determine whether your documents are using Unicode encoding, open the XML document with any editor or using Notepad and look at the beginning of the document to find `[encoding="UTF-8"]` in the declaration statement. UTF-8 indicates the 8-bit Unicode encoding.

Diff

Using this operation you can compare the source XML document to the document specified in the Second Operand section and write the differences to an XML document called a *Diffgram* document. The Diff operation provides a number of options to customize this comparison:

- **DiffAlgorithm** Provides three choices: Auto, Fast, and Precise. You can choose between comparison algorithm to be fast or precise. The Auto option lets the Diff operation decide whether to select a fast or precise comparison based on the size of the documents being compared.

- **IgnoreComments** Specifies whether comment nodes are compared.
- **IgnoreNamespaces** Indicates whether the namespace URI (uniform resource identifier) of an element and its attribute names are compared.
- **IgnorePrefixes** Specifies whether prefixes of element and attribute names are compared.
- **IgnoreXMLDeclaration** Specifies whether the XML declarations are compared.
- **IgnoreOrderOfChildElements** XML documents have hierarchical structure, and this option specifies whether the order of child elements is compared.
- **IgnoreWhiteSpaces** Specifies whether white spaces are compared.
- **IgnoreProcessingInstructions** Specifies whether the processing instructions are compared.
- **IgnoreDTD** Specifies whether the DTD is ignored.
- **FailOnDifference** Specifies whether the task fails if the Diff operation fails, e.g., an XML document fails to validate according to the validation schema.
- **SaveDiffGram** Choose to save the comparison result in a Diffgram document.

Patch

Using this operation, you can apply the Diffgram document you saved earlier in the package during the Diff operation to an XML document. By doing this, you actually create a new XML document that includes the contents of the Diffgram document created earlier by the Diff operation.

Execute SQL Task

The Execute SQL task is the main workhorse task to run SQL statements or stored procedures and uses the power of the underlying relational database. If you have used DTS, you may have used this task. Typically in DTS, once you have loaded data into a database and you apply transformations using the Execute SQL task. These transformations vary from generating salutations to lookup transformations, deriving columns, or applying business rules using the SQL Server relational engine. The design philosophy used in SQL Server 2008 Integration Services allow you to perform many of these tasks during the loading phase while data is still in memory, thereby increasing performance by reducing the repeated and inefficient transformations that require data to be staged or involved read/write operations on hard disks. The power of the Execute

SQL task is still available in SSIS in a more usable form by providing ability to use variables, to create expressions over the properties of the task, and to return a result set to the control flow that can be used to populate a variable.

Using the Execute SQL task, you can perform workflow tasks such as create, alter, drop, or truncate tables or views. You can run a stored procedure and store the result to a variable to be used later in the package. You can use it to run either a single SQL statement or multiple SQL statements, parameterized SQL statements, and save the rowset returned from the query in to a variable. You have already used this task in the "Using System Variables to Create Custom Logs" Hands-On exercise in Chapter 3, where you used a parameterized SQL statement, and in the "Contacting Opportunities" Hands-On exercise in Chapter 4, where you saved the resulting rowset to a variable, which then got enumerated over by a Foreach Loop Container.

If you scroll to the Maintenance Plan Tasks in the Control Flow Toolbox, you will see a similar task, the Execute T-SQL Statement task. The Execute T-SQL Statement task has a more simple interface than the Execute SQL task and is focused on performing maintenance tasks on SQL Server databases using T-SQL. It doesn't give you any facility to run parameterized queries and direct the result set to the work flow, whereas the Execute SQL task has a more complex interface and is designed for use in a relatively complex workflow where you need to use SQL Statements against not only the SQL Server but a variety of sources, deal with variables, run parameterized queries, or direct the result set to the data flow.

Keep this task open in front of you and try various selections as we go through each option, as the task contains dynamic fields that change depending upon the choices you make in certain fields.

The Execute SQL Task Editor includes General, Parameter Mapping, Result Set, and Expressions pages.

General Page

In this page, you define a Name and Description for the task under the General section. In the Options section, you can specify a TimeOut value in seconds for the query to run before timing out. By default, the TimeOut value is 0 (zero), indicating an infinite time. The CodePage field allows you to specify the code page value.

In the Result Set section, you choose one of four options based upon the result set returned by the SQL statement you specify in this task. Based on the type of SQL statement—i.e., whether it is a SELECT statement or INSERT/UPDATE/DELETE statement—the result set may or may not be returned. Also, the result set may contain

zero rows, one row, or many rows, and the following four options in the Execute SQL Task Editor ResultSet field allow you to configure them:

- **None** Use this value when you use INSERT, UPDATE, or DELETE SQL statement that returns the result set containing zero rows.
- **Single Row** When the SQL statement or a stored procedure returns a single row in the result set.
- **Full result set** When the query returns more than one rows.
- **XML** When the SQL statement returns a result set in the XML format.

In the SQL Statement section ConnectionType field, options are the EXCEL, OLE DB, ODBC, ADO, ADO.NET, and SQLMOBILE connection manager types, used for connecting to a data source. Depending on the type of connection manager you've chosen, the Connection field provides a drop-down list of already configured connection managers of the same type or provides a <New Connection…> option to let you add a connection manager of the appropriate type. The interface provided by the <New Connection…> option changes to match your selection of the connection manager specified in the ConnectionType field.

Depending on the data source type, you can write a query using an SQL statement in the dialect the specified data source can parse. Further, you can specify the source from where the SQL statement can be read for execution in the SQLSourceType field. The selection of the source in the SQLSourceType field changes the next field dynamically (which is coupled to it) to match the SQLSourceType choice. The options available in the SQLSourceType field and how it affects the coupled field are explained here:

- **Direct input** Allows you to type an SQL statement directly in the task. This changes the coupled field to SQLStatement, which provides an interface in which to type your query.
- **File connection** If you have multiple SQL statements written in a file, you can choose this option to enable the Execute SQL task to get the SQL statements from the file. Selecting this option changes the coupled field to FileConnection, which allows you to specify a File Connection Manager to connect to an existing file containing SQL statements.
- **Variable** Enables the Execute SQL task to read the SQL statement from a variable. This option changes the coupled field to the SourceVariable field, which provides a drop-down list of all the system and user variables.

If you have chosen ADO or ADO.NET Connection Manager in the ConnectionType field earlier, the IsQueryStoredProcedure field becomes available for use and you will be able to specify that the SQL statement to be run is a stored procedure.

When you select OLE DB Connection Manager in the ConnectionType field, you can specify in the BypassPrepare field whether the task should skip preparing the query step before running it. When SQL queries are run, the SQL statement is parsed and compiled in the first step and then executed based on the execution plan prepared earlier in the compilation step. By default the BypassPrepare field is set to True, which means when the query is run the first time, it parses and compiles the SQL statement in the first step and keeps the execution plan in the cache of the database engine and in the next step uses this execution plan to execute the query. So when the query is run the next time, it doesn't parse and compile the statement but rather uses the existing execution plan, saving time required to parse and compile. If you're running a query multiple times, it is faster and efficient if you use a prepared execution—however, if you're running the query only once, it is not a recommended option.

Parameter Mapping Page

If you are running a parameterized SQL statement, you can map variables to the parameters in this page. The interface on this page is relatively simple and provides self-explanatory fields and options. You can click Add to add a parameter mapping and then click in each field to select the available values from the drop-down lists; however, in the Parameter Name field, you have to type in a value. The configuration of this page has been covered in the "Using System Variables to Create Custom Logs" Hands-On exercise in Chapter 3.

Result Set Page

If you are using a query that will return a result set and you have selected a value other than None in the ResultSet field on the General page, you can use this page to map the row set to a variable, which can be used later in the package.

Expressions Page

Using this page, you can build a property expression using SSIS expression language to update the property dynamically at run time. In the Expression Builder, you can use System or User variables, functions, type casts, and operators exposed by expression language to build an expression.

Bulk Insert Task

When you get large amounts of data in the flat files, say from the mainframe systems or from third parties, and you want to import this data from the flat files into an SQL Server, you can use the Bulk Insert task. This task is the quickest way to copy large amounts of data into an SQL Server table or view.

Note the following when deciding whether to use this task:

- The Bulk Insert task does not perform any validation or transformation on the data while importing data from the source text file to the SQL Server database table or view. So, it is more suitable in cases where the data is coming from a reliable source or you intend to apply business processes in later stages. A similar component—SQL Server Destination—is provided in the data flow task that can provide the performance of a bulk insert. You will be using SQL Server Destination if you want to perform transformations on data before loading and your target server is the local server. SQL Server Destination is covered in Chapter 9 in detail.
- When you embed this task in an SSIS package Control Flow, only the members of the sysadmin fixed server role can run the package.
- You can use an XML- or non-XML-formatted file in the Bulk Insert task. A formatted file is used to store format information for each field in a data file in relationship to a specific table and provides all the format information that is required to bulk-import data. This format file must be located on the server executing the SSIS package containing the Bulk Insert task.
- During the import process, some of the rows may fail to import and in turn can fail the task. You can still run this task successfully by increasing the allowed number of errors by specifying a value in the MaxErrors option. However, the rows that fail to import cannot be logged or extracted out. If you want to capture the failing rows, you need to consider alternative ways of importing data using the Data Flow task and capture the failing rows in the exception files by way of error outputs.

Message Queue Task

Microsoft Message Queuing (MSMQ) is a Windows service that allows applications or SSIS packages to send and receive messages using a queue. Message queuing is available in Windows 2000 and later released as a standard operating system component that has the advantages of Active Directory Integration. This service can be extended using the Message Queuing Connector to heterogeneous data sources such as CICS (IBM's Customer Information Control System) or UNIX.

In the SSIS world, message queuing means one package will be sending a message to a queue while another package will be receiving that message from the queue. These packages can be on different servers and may not be running simultaneously. Using MSMQ, you can reliably exchange data between packages that may not be running on the same server and might be separated in time throughout your enterprise. Following are some of the scenarios in which you may be using the Message Queue task:

- Send a message from an executing package to the package that is waiting for it to complete so that it can start running.
- If you have a small window of time to finish a large workload and you decide to distribute your workload across many servers to utilize the processing power, you can coordinate the operations on different servers using message queuing between packages.
- SSIS packages can communicate with the applications that utilize message queuing.
- Send output from a processing package to a waiting package on the other computer, where the data enclosed in the message will be processed further.

You can configure this task into either send mode or receive mode. Select Send Message in the Message field to configure it into send mode or select Receive Message in the Message field to configure it into receive mode. The package that is running the message queue task in the receive mode can be configured to either run and wait for the message in the queue using a looping construct or run on a schedule that is later than when the message is dropped in the queue just to pick up the message. SSIS is more suitable for the applications that can be satisfied with the latter implementation. If you need to wait for messages, for example, your application is sending data in message format as the transactions happen, Biztalk is more suitable for such implementations. Biztalk server is designed to support such requirements and puts negligible load on the server if it has to wait for messages, whereas SSIS will consume resources, as it has to keep looping and stay in memory. The message queue task is provided in SSIS to transfer the information that is relevant for the remaining ETL to complete rather than read or load data from a transactional system.

You can choose from one of the different types of messages to send or receive using the Message Queue task: Data file message, String message, or Variable message. When the task is configured in the receiving mode, you also get an additional String message to variable MessageType choice. As in other Integration Services tasks, the

available fields in this task are dynamic and change depending upon your choice to send or receive a message or on message type.

- **Data file message** Used to send or receive a file that contains the message. To send a data file message, you specify Data File Message in the MessageType field and specify the path of the file in the DataFileMessage field on the Send page of the Message Queue Task Editor. When receiving this type of messages, you select Data File Message in the MessageType field and specify the name of the file in the SaveFileAs field into which you want to save the message. Also, you can choose to overwrite the existing file, and optionally apply a filter to receive the message only from the specified package. You specify the full path and package name in the Identifier field on the Receive page.
- **Variable message** Used to send or receive one or more variables. To send variables, you specify Variable Message in the MessageType field and one or more variables in the VariableMessage field on the Send page of the Message Queue Task Editor. While receiving the variables, you specify Variable Message in the MessageType field, specify the name of the variable in the Variable field to receive the message into, and optionally choose to apply a filter to receive the message only from the package specified in the Identifier field on the Receive page.
- **String message** Used to send a text string. To send a text string, you specify String message in the MessageType field and type a text string in the StringMessage field on the Send page of the Message Queue Task Editor. While receiving the text string, you specify String message in the MessageType field and optionally specify to compare the incoming string in the Compare field with a user-defined string specified in the CompareString field on the Receive page. The string comparison options in the Compare field can be Exact Match for an exact comparison, Ignore Case for case-insensitive comparison, or Containing for a substring match.
- **String message to variable** Used to pass the source message that has been sent as a string to a destination variable and is available only when receiving messages. To configure a text string message to be passed to a variable, you specify String Message To Variable in the MessageType field, specify the name of the variable in the Variable field to receive the text string into, and optionally specify to compare the incoming string in the Compare field with a user-defined string specified in the CompareString field on the Receive page. The string comparison options in the Compare field can be Exact Match for exact comparison, Ignore Case for case-insensitive comparison, or Containing for a substring match.

To use the Message Queue task to send and receive messages, you first need to install the Message Queuing service and then create messaging queues. You can create either a public or a private queue, depending upon whether you have installed the Active Directory Integration component of the Message Queuing service. A public queue is created in an Active Directory environment to publish its properties and description to the Active Directory. A private queue does not publish itself to the Active Directory but works on the local computer that holds the queue.

The Message Queuing service has the following components available in Windows 2008 Server for installation:

- **Message Queuing Server** This component enables you to perform Message Queuing functions such as guaranteed message delivery, efficient routing, improved security, support for sending messages within transactions, and priority-based messaging. It can be used to implement both synchronous and asynchronous messaging solutions.
- **Directory Service Integration** Provides integration with Active Directory whenever the computer belongs to a Windows domain (Windows 2000 and later). With this component, you can publish queue properties to the directory, authenticate and encrypt messages using digital certificates registered in the directory, and route messages across directory sites. Public queues are configured and used when using this component.
- **Message Queuing Triggers** Associates the arrival of messages at a queue with triggering functionality of a COM component or a standalone application, depending upon the filters that you define for the incoming messages in a given queue.
- **HTTP Support** Enables sending or receiving of messages over HTTP transport with proxy settings configured using the proxycfg.exe tool.
- **Multicasting support** Enables queuing and sending of multicast messages to a multicast IP address.
- **Routing Service** Available in Windows 2008 Server to provide routing of messages using the store and forward mechanism in Active Directory transport between different sites or within the same site.
- **Windows 2000 Client Support** Available in Windows 2008 Server to provide support for Message Queuing clients on Windows 2000 computers in the domain.
- **Message Queuing DCOM Proxy** Enables a computer to act as a DCOM client of a remote Message Queuing Server.

For proper functioning of the Message Queue task, make sure you have installed the Message Queuing Server and the SQL Server Integration Services service. When you install SQL Server 2008 without specifically selecting the Integration Services service, you may still be able to use BIDS to design and run Integration Services packages. However, as this is a partial installation of SSIS, not all tasks will run properly, and the Message Queue task is one of those that won't. For the Message Queue task to be functioning properly, you must install Integration Services fully during installation of SQL Server 2008.

Next, you will import the expanded files into the Campaign database in SQL Server using the Message Queue task and the Bulk Insert task in the following Hands-On exercises.

Hands-On: Importing Expanded Files

Dealers of Adventure Works Bikes submit the sales reports to an FTP server, and they need to be downloaded from this server and imported into the SQL Server database. You have already downloaded, expanded, and archived these report files in the previous exercises, and now you have DealerSales01.txt and DealerSales02.txt files in the Downloads folder ready to be imported to the Campaign database. However, you want to run this new package independent of earlier packages, and this may occur at a different time during the day.

Method

We can meet this objective in several ways: We can use the Bulk Insert task to import the data to our existing table—but we have multiple files, so we can use the Foreach Loop Container to read the filenames one by one and pass them to the Bulk Insert task to import multiple files one at a time. However, as we want to keep the packages independent from one another and want to run the second package only when the first has completed successfully, the use of messages from the first package to pass the information to the other will be a better solution. Though in this case you will not be following the shortest and easiest solution option to achieve your goal, the solution you are going to follow is quite interesting, and I'm sure it will be relevant in real-life scenarios.

You will add the Message Queue task at the end of the Archiving Downloaded Files package to send filenames in the messages, which will be read by the Importing Expanded Files package using the Message Queue task in the receiving mode. As the package won't have to read filenames from the file system, you will not be using

the Foreach Loop Container; instead, you will use the For Loop Container to read multiple messages one by one. Here's the step-by-step method:

1. Install the Message Queuing service and create message queues.
2. To keep it simple and applicable to most of the users, for the sake of this exercise I've used a Windows Server 2008 machine with a private queue only. If you've a different environment, refer to Microsoft SQL Server 2008 Books Online for more details to create a message queuing environment suitable to your requirements.
3. Configure the Archiving Downloaded Files package to send filenames. In this step, we will add Message Queue task in the already created Archiving Downloaded Files package.
4. Build the Importing Expanded Files package. And the final step is to execute this package and check that the two messages have been picked up from the queue and the text files have been imported into the SQL Server Campaign database.

Exercise (Install Message Queuing Service)

Let us start this exercise by installing the Message Queuing service. You will also create message queues while learning about private and public queues.

1. From the Server Manager, click the Add Features link to start the Add Features Wizard.
2. In the Select Features dialog box, expand the Message Queuing group and select the Message Queuing Server from the list. Click Next to go to Confirm Installation Selections page.
3. Verify the selection and click Install to install the Message Queuing Server component.
4. When the installation is completed successfully, click the Close button in the Results page to close the wizard.
5. Again in the Server Manager, expand the Features node in the left pane of the window to see the installed Message Queuing feature.
6. Right-click the Private Queues folder and choose New | Private Queue from the context menu. This will open the New Private Queue window. In the Queue name field, type **SSISprivQ** and then click OK.

   Note that another difference between a private queue and a public one is that the *private$* is attached to the full name of a private queue. The New Private Queue window shows that the queue will be created in *your computer name* with *private$* added in front of the name you type in. So, the full path of the queue will be *ComputerName\private$\SSISprivQ*. If this were the public queue in which we specify the queue name as *SSISpubQ*, the full path would have been *ComputerName\SSISpubQ*.

7. Expand the Private Queues folder under the Message Queuing from the left pane of the Computer Management window. Further expand the SSISPrivQ folder and you will see a Queue messages subfolder. This is where the Message Queue task will be delivering the messages and the messages will wait for a receiving Message Queue task to pick them up.

Exercise (Configure Archiving Downloaded Files Package to Send Filenames)

Remember that in the Archiving Downloaded Files package, the Foreach Loop Container reads the filenames of zipped files lying in the C:\SSIS\downloads folder and then passes those names to the File System task, which moves the zipped files to the Archive subfolder. To accomplish this, the package uses a variable named *fname* to pass the zipped files names. Because you want to import text files, not the zipped files, you will derive a new variable called *txtfname* to contain the text files names from *fname* variable. This new variable will then be sent through the Message Queue task as a variable message.

8. Run BIDS and open the Control Flow Tasks project. In the Solution Explorer window, double-click the Archiving downloaded files.dtsx package to open it.
9. Open the Variables window. Click the Auto Hide pushpin to dock the Variables window on the left side of the screen. On the Control Flow surface, click the Enumerating Files Foreach Loop Container and switch to the Variables window. Add a new variable named *txtfname* in the Enumerating Files Scope and set Data Type to string. Press F4 to open the Properties window. The Properties window will open up by default on the right side of the screen, showing properties of the *txtfname* variable. (Sometimes you may not see properties for the object in which you are interested—this is because of your selection of the items on the designer. If this happens, make sure you've selected the Enumerating Files Container and then clicked the *txtfname* variable to see the properties for this variable.)

   Scroll through the properties and locate the *EvaluateAsExpression* property. This property allows you to enable the variable to use the results of an expression as its value. That is, you can write an expression, which will be evaluated at run time and the result of this evaluation will be used as the value of the variable.

   Select True for the EvaluateAsExpression property. Type the following text in the Expression property:

   ```
   SUBSTRING(@[User::fname] , 1, LEN(@[User::fname]) -3) + "txt"
   ```

   This expression will evaluate *txtfname* as C:\SSIS\downloads\DealerSales01.txt from the fname variable when it equals C:\SSIS\downloads\DealerSales01.zip. If you look further down in the properties window in the Value field, you will notice

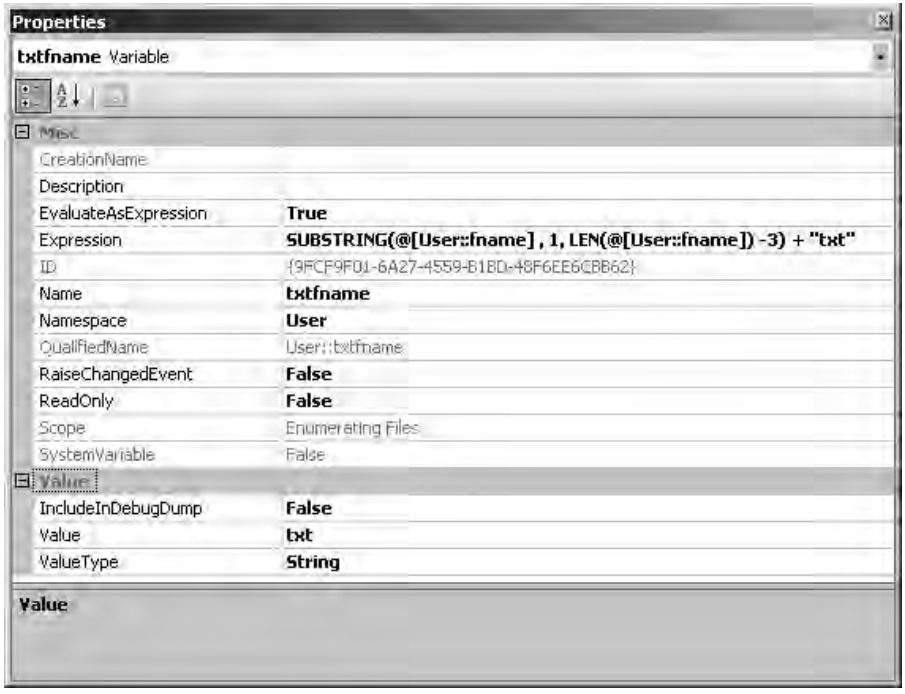


Figure 5-9 *Deriving the txtfname variable using property expressions*

(see Figure 5-9) that the full filename in fact has been resolved as txt. You can see that nothing appears in the name portion of the file path name because the *fname* variable has not yet been populated (by the Foreach Loop Container) in the design mode. However, at run time, the Foreach Loop Container provides a value for the *fname* variable and the name portion of the filename gets populated in the file path name.

10. Drag the Message Queue task from the Toolbox and drop it in the Enumerating Files (Foreach Loop) Container below the Archive Downloaded Files (File System) task. Drag and drop the green arrow, the precedence constraint, from below the Archive Downloaded Files task on to the Message Queue task. The package will look as shown in Figure 5-10 after you've configured the Message Queue task.
11. Right-click the Message Queue task and choose Edit from the context menu. Type the following on the General Page of the task editor:

Name	Send variable message
Description	This task sends the txtfname variable to the Importing expanded files.

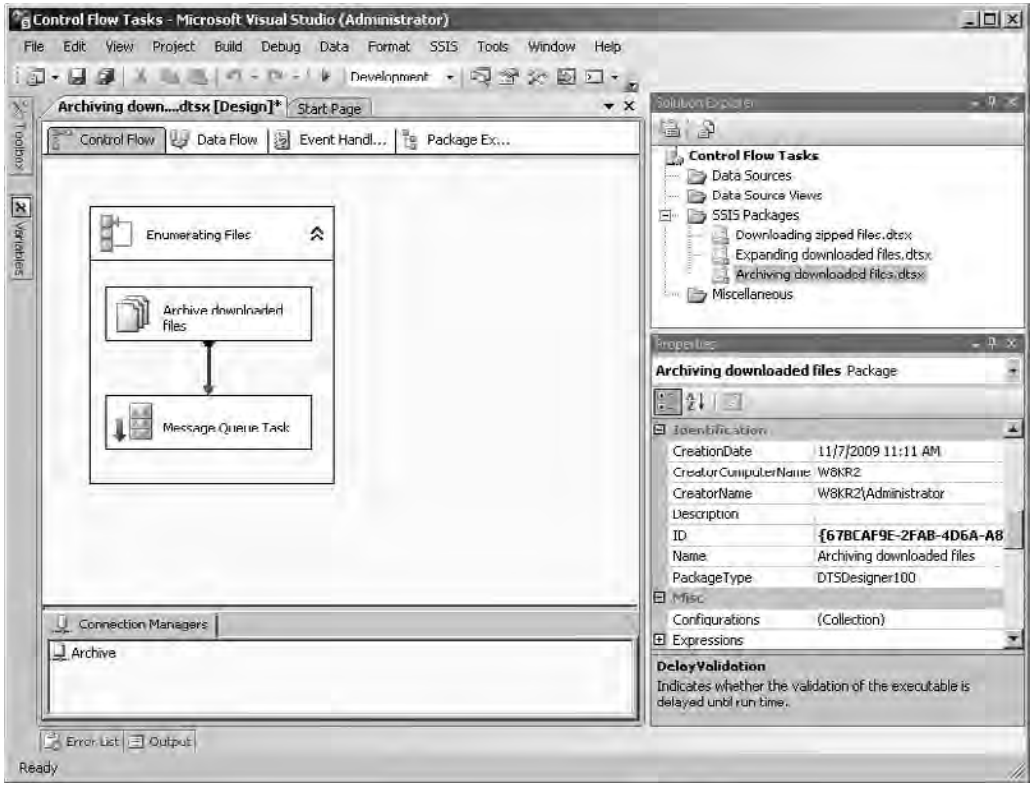


Figure 5-10 *Archiving the downloaded files package after adding Message Queue task*

If you want to send this message in the format that is acceptable to SQL Server 2000 Message Queue task, you can use the Use2000Format field. For now, leave the default False value set in this field.

12. In the MSMQConnection field, select to create a new connection. In the MSMQ Connection Manager Editor, type the following:

Name	SSISprivQ
Description	Connection Manager for private Queue SSISprivQ
Path	*YourComputerName*\private$\SSISprivQ

You can use a dot (.) instead of *YourComputerName* to indicate your local computer.

You may want to test the connection by clicking Test; once you get the "Test connection succeeded" message, click OK twice to return to the Message Queue Task Editor.

13. Leave the Send Message option selected in the Message field. The settings in General page should be like the ones shown in Figure 5-11.
14. Go to the Send page. Leave UseEncryption set to False. This field can be used to encrypt your messages using an encryption algorithm (RC2 or RC4) specified in the EncryptionAlgorithm field.

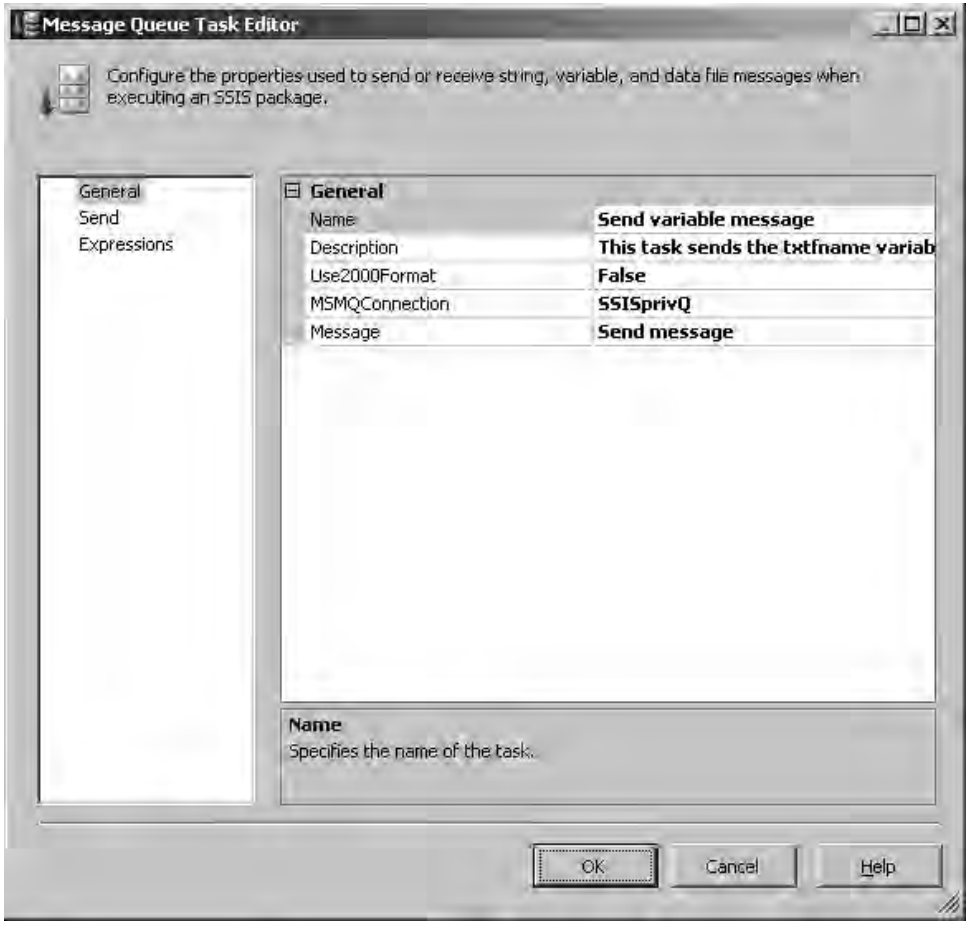


Figure 5-11 *Message Queue task configured to send messages*

15. Select Variable message in the MessageType field, as you will be sending a variable value in the message. Select User::txtfname in the VariableMessage field. Click OK to close the editor window.
16. Save the package by pressing CTRL-SHIFT-S.
17. Using Windows Explorer, move the DealerSales01.zip and DealerSales02.zip files from the C:\SSIS\downloads\Archive folder to the C:\SSIS\downloads folder that have been moved to the Archive folder in earlier exercise.
18. Switch to BIDS and press F5 on the keyboard to start debugging. When the package runs, notice that the tasks change color from yellow to green twice, indicating that the package has moved two files and sent out two messages. When all the tasks turn green, stop debugging from the Debug menu or by pressing SHIFT-F5. Close the project by choosing File | Close Project.
19. Switch to the Server Manager and notice two messages in the Queue messages folder under SSISprivQ. (You might have to refresh the folder if the window was already open.)

Exercise (Build Importing Expanded Files Package)

In this part of the exercise, you will build a new package, which will pick up variable messages sent by the Archiving Downloaded Files package earlier from the SSISprivQ messaging queue.

20. In the Solution Explorer, add a new package and rename it **Importing expanded files.dtsx**.
21. Start building the package by dropping the For Loop Container on the Control Flow Designer surface.
22. Right-click anywhere on the Designer surface and choose Variables from the context menu. Create three variables: *filename*, *maxcounter*, and *mincounter* with the details shown in the Figure 5-12.

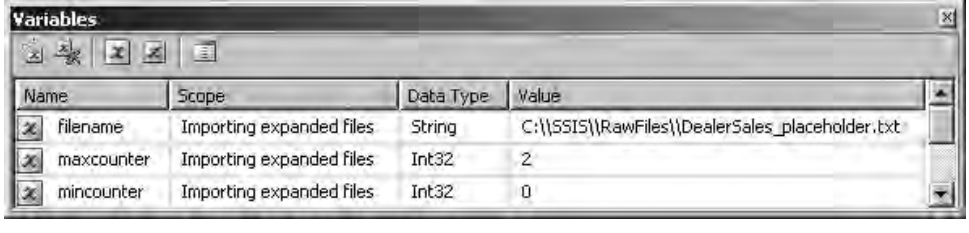


Figure 5-12 *Creating variables for the text filenames and For Loop Container*

Be sure to specify a value of C:\\SSIS\\RawFiles\\DealerSales_placeholder.txt to the *filename* variable and a value of 2 to the *maxcounter* variable. The *mincounter* and *maxcounter* variables will be used in the For Loop Container, while the *filename* variable will be used to receive the variable value from the messages sent by Archiving Downloaded Files package using the Message Queue task.

23. Double-click the For Loop Container icon to open the For Loop Editor. Type the following in the General page of the editor:

Name	Import Loop
Description	This For Loop Container imports DealerSales files one by one.
InitExpression	@mincounter = 0
EvalExpression	@mincounter < @maxcounter
AssignExpression	@mincounter = @mincounter + 1

As the initial value assigned to the *mincounter* variable in the InitExpression field is 0 and the maximum value specified for *maxcounter* variable while creating it is 2, the Import Loop will loop twice at run time, after which the expression defined in EvalExpression field will become false and the loop will stop running. Click OK to close the For Loop Editor window.

24. From the Toolbox, drag and drop the Message Queue task within the Import Loop (For Loop) Container. Open the Message Queue Task Editor window by double-clicking its icon and type the following on the General page:

Name	Receive variable message
Description	This task receives variable messages sent by 'Archiving downloaded files' package

25. Leave the Use2000Format field set to the default False value.
26. Create an MSMQ connection manager in the MSMQConnection field exactly as specified in Step 3 of the preceding series of steps.
27. Select Receive Message in the Message field. Note that Send changes to Receive on the left pane of the window. Go to the Receive page.
28. On the Receive page, select True in the RemoveFromMessageQueue field. This value lets you delete the message from the queue after the Message Queue task has received it. You may prefer to keep the message in the queue in situations when you have multiple subpackages to read a message.
29. You can choose to display an error message if the task fails with timeout in the ErrorIfMessageTimeout field. For now, leave it set to False. However, if you select True in this field, you will be able to specify a time-out value in seconds in the TimeoutAfter field.

30. Choose Variable Message in the MessageType field and No Filter in the Filter field. If you select the From Package option in the Filter field, you can specify the package in the Identifier field.
31. Specify User::filename in the Variable field, as shown in Figure 5-13.
32. Click OK to close the Message Queue Task Editor.
33. Drag and drop the Bulk Insert task from the Toolbox in the Import Loop Container below the Receive variable message task. Connect the Receive variable message task to the Bulk Insert task using the on success (green colored line) precedence constraint.

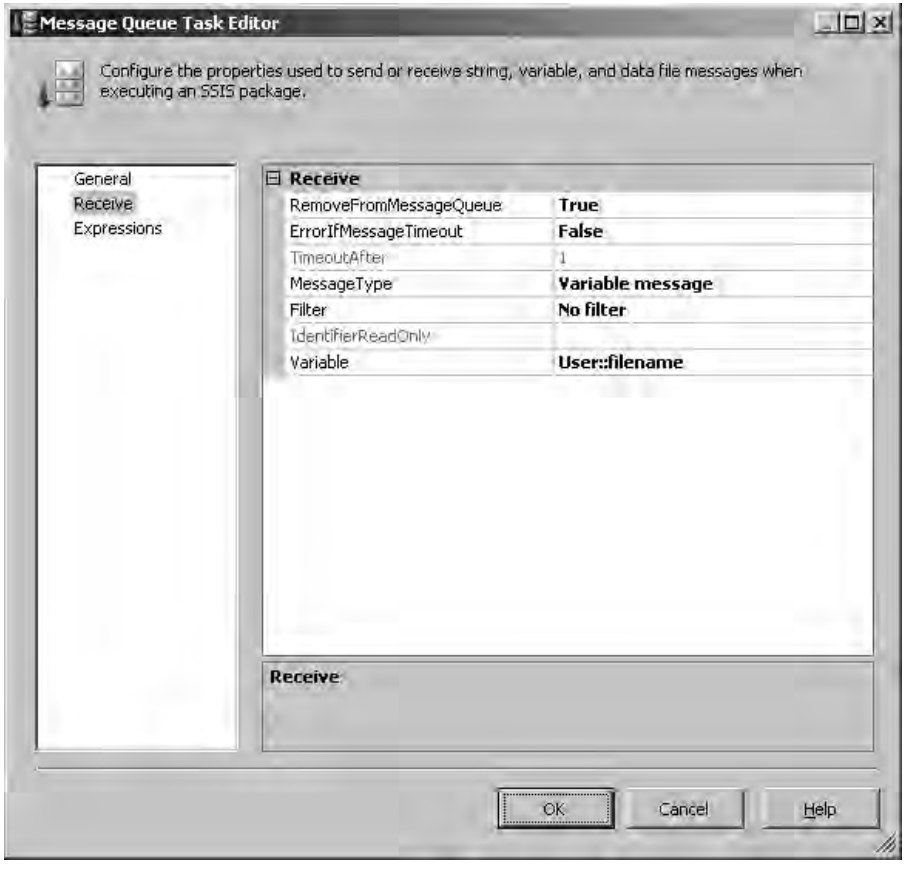


Figure 5-13 *Configuring the Message Queue task in receive mode*

34. Open the Bulk Insert Task Editor and type the following in the General page:

Name	Importing DealerSales
Description	This task imports DealerSales files into Campaign..DealerSales table.

35. Go to the Connection page, click in the Connection field under the Destination Connection group of options, and choose <New Connection…> to open the Configure OLE DB Connection Manager window. Choose an OLE DB connection manager from the Data Connections list to connect to Campaign database, which you created in an earlier Hands-On exercise. The good thing about the OLE DB Connection Managers is that they are available for reuse in other packages, as you have seen here. If you skipped the earlier Hands-On exercises and do not see an OLE DB Connection Manager in this window, you will have to create a new connection manager by clicking New. For details on how to create an OLE DB Connection Manager, refer back to the "Contacting Opportunities" Hands-On exercise in Chapter 4.
36. Click in the DestinationTable field and then click the drop-down arrow to see the list of tables in the Campaign database. OLE DB Connection Manager provides this list by establishing a connection to the database using the settings specified in the connection manager. Select the [Campaign] .[dbo].[DealerSales] Table from the list.
37. You can choose to specify the format of the file to be imported either directly in the task or by using a file. If you choose Use File in the Format field, you have to specify the name and location of the file in the FormatFile field, which appears on selection of the Use File option. For this exercise, choose Specify in the Format field. Leave the RowDelimiter field set to {CR}{LF} and choose Vertical Bar {|} from the drop-down list in the ColumnDelimiter field.
38. Under Source Connection options group, choose <New Connection…> to specify a File Connection Manager. As you will be using filenames provided by the Receive Variable Message task in the form of variables, you will specify a filename here as a placeholder. Specify C:\SSIS\RawFiles\DealerSales_ placeholder.txt in the File field of the File Connection Manager Editor (see Figure 5-14). You can create a blank DealerSales_ placeholder.txt file in case you're having difficulty specifying it in the File Connection Manager.
39. Go to the Options page, where you can specify the code page of the text file in the CodePage field. Leave it set to the default RAW.
40. When the data is coming from various database systems, the data type may be in different formats. The Bulk Insert task provides four default data formats that can be imported. You can specify the data type of the input file using the DataFileType field. The options are Character, Unicode (wide) Character, Native, and Unicode (wide) Native formats. Leave it set to char.

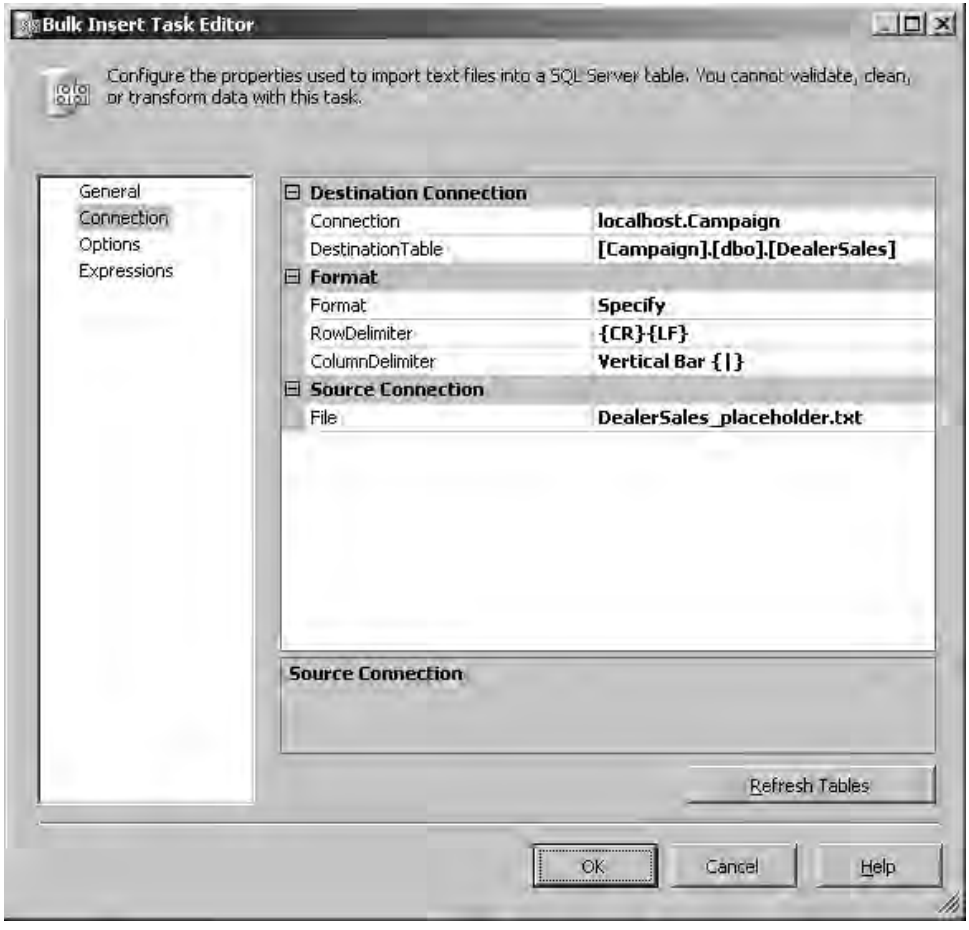


Figure 5-14 *Configuring the Bulk Insert task to import text files*

41. You can specify the number of rows in a batch in the BatchSize field. The rows specified in a batch are treated together and copied to the server as one transaction. The Bulk Insert task copies files by committing all the rows in the batch as one transaction and moving over to the next batch for another transaction. In case of an error and failure of the task, all the rows in the batch will be rolled back. For example, if you set the BatchSize equal to 10,000 rows for a table of 50,000 rows and the task fails at row number 25,001, the task will fail with 20,000 rows inserted to the table. A default value of 0 implies that all the rows will be treated as a single batch—i.e., fail or commit totally in one transaction.

42. In the LastRow field, you can specify the value of last row at which the task should stop inserting data to the table. Leave it set at the default value of 0 that means all the rows from the file will be inserted to the specified SQL table.
43. The FirstRow field is quite useful in situations where you have a large amount of data to import and the quality of data results in failing the process in between. By specifying where to start inserting the rows, you can avoid re-importing the rows that have already been imported. Using this option in conjunction with the BatchSize option helps in achieving high levels of input performance with less rework, even though the data quality may not be good. Leave the option selected to the default value of 1.
44. Moving to the Options section, you can choose any of the five options. You can also select more than one option here, which then will be listed in a comma-delimited list.
    - **Check Constraints** Checks the table and column constraints.
    - **Keep Nulls** Imports blank columns from the text file as Null values.
    - **Enable Identity Insert** Inserts explicit values into the identity column of the table.
    - **Table Lock** Locks the table during import process.
    - **Fire Triggers** Fires any existing triggers on the table while importing the data.

    Though none of these options will be of much help in this case, you can select Keep Nulls for this exercise.
45. You can specify the names of columns on which to sort the data in the SortedData field. This is effectively the ORDER BY clause in the bulk insert SQL statement. Leave it blank, which means do not sort on any column.
46. The rows that cannot be imported by Bulk Insert task are counted as errors. You can specify a maximum number of errors—i.e., the number of rows to fail before the task fails—in the MaxErrors field. Click OK to close the task.

    You have configured all the tasks and options within the tasks apart from specifying which files to import. You can do this by mapping the connection string of the placeholder File Connection Manager to the value of the variable received by the Receive variable message task. Let's see how to do this.
47. Right-click the DealerSales_placeholder.txt File Connection Manager and choose Properties from the context menu. In the Property window, click in the Expressions field and then click the ellipsis button on this field. In the Property Expressions Editor, click in the field below the Property column and select ConnectionString from the drop-down list. Then click the ellipsis button under the Expression field to open the Expression Builder.

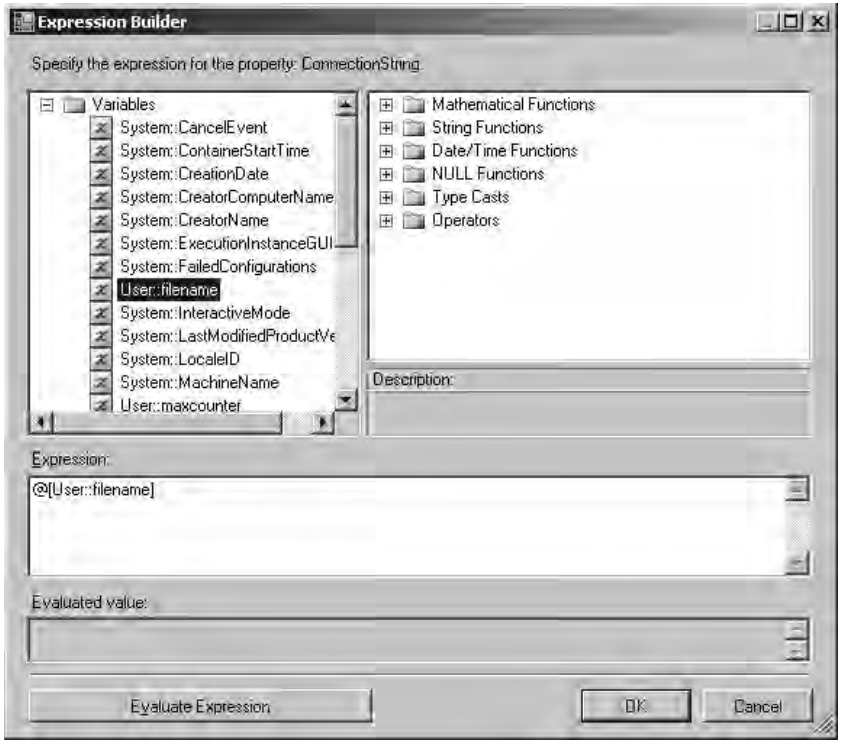


Figure 5-15 *Using Expression Builder to create a property expression for dynamically altering the connection string of the file connection manager*

48. In the Expression Builder, drag User::filename from the Variables list to the Expression field, as shown in Figure 5-15. Click OK twice to finish the configurations.
49. As a last step before you run the package, make sure that you have two messages queued, which were sent by the Archiving Downloaded Files package, in the message queue folder in SSISprivQ. Also, check that the DealerSales table in the Campaign database does not have any record.
50. In a real-life scenario, you will be running such a package under a schedule that wakes up after the message has been dropped in the queue just to read the message. For our exercise, press F5 to run the package.

    Notice that the Import Loop task turns yellow, followed by the Receive Variable Message task turning yellow and then green, indicating that it has successfully received the first variable message. Then Importing DealerSales turns yellow and stays yellow for some time before turning to green, indicating that it has successfully imported DealerSales01.txt file (Figure 5-16). This completes the first iteration of the Import Loop task and the process repeats itself for the second iteration, after which all the tasks turn green and stop processing.

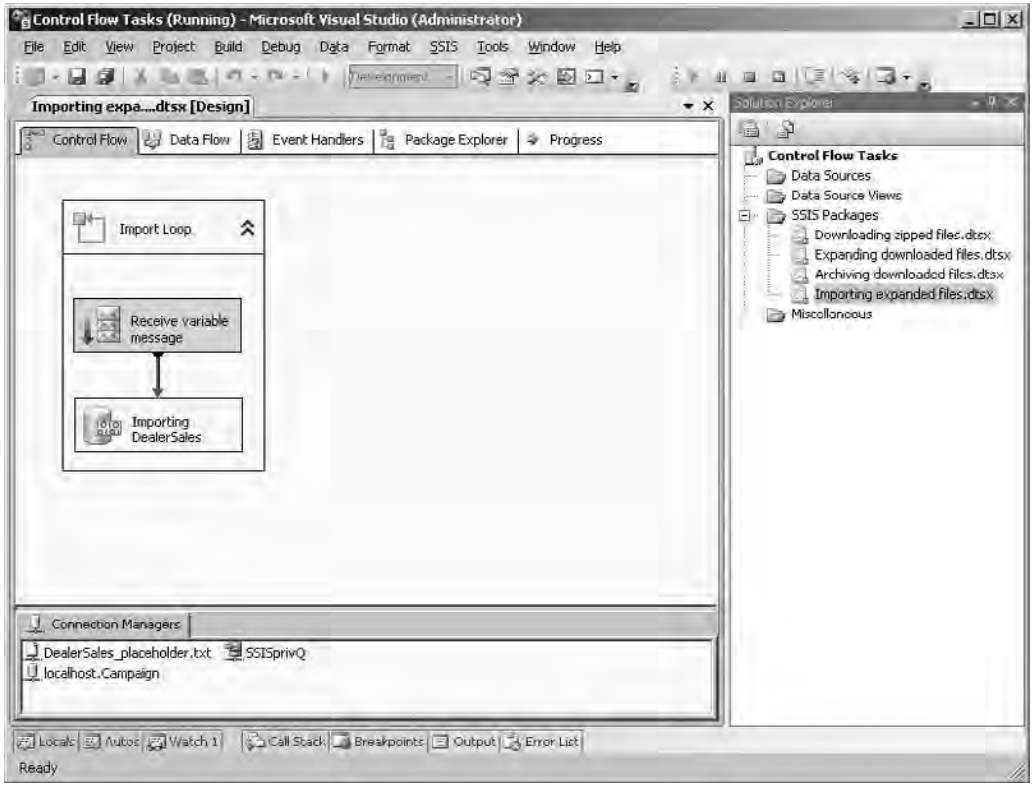


Figure 5-16 *Executing the Importing expanded files package*

To close the loop, check the Queue messages folder under SSISprivQ by switching over to the Server Manager and notice that the messages have been deleted. Go to SQL Server Management Studio and run the following query against the DealerSales table to see the imported data and total number of records:

```
SELECT * FROM DealerSales
```

You should see 242,634 rows displayed.

51. Press CTRL-SHIFT-S to save all the files in this solution and then choose File | Close Project.

Review

In this exercise, you used the Message Queue task to send and receive variables from one package to another using the Windows Message Queuing service. This can be quite useful for enterprise-wide implementations in which data and servers are scattered

all over the network, and it makes sense to use all the processing power underutilized in those servers to help you complete the nightly processes within the allocated time. You also used the Bulk Insert task to import files to an SQL Server table. This is the fastest method of importing data to an SQL Server table. In this exercise, you had encountered the For Loop Container and used it to iterate twice to receive two messages using the Message Queue task and then import two text files into the SQL Server table using Bulk Insert task. Last but not the least, and perhaps the most important thing you've learned in this exercise, you used property expressions to change ConnectionStrings of the File Connection Managers dynamically at run time with the help of variables.

Execute Package Task

The Execute Package task brings the benefits of modular design to SSIS packages. It allows an SSIS (parent) package to run the other (child) packages as part of a workflow. This task can be used to run packages stored either in the SQL Server MSDB database or on the file system.

While developing a solution for a business problem, you tend to build smaller packages meeting specific needs. This approach is recommended, as it helps your solution to be modular, which is easier to debug. This also helps you achieve quick results on some of the specific requirements. Once you have built all the modules of your complete solution, you can use the Execute Package task to fit all the smaller packages together to form an enterprise-wide solution.

Another benefit of following a modular design for SSIS packages is less work when you have to modify a child package that is used in multiple parent packages. For example, if you want to modify a child package that is used in five parent packages, you can modify your child package only once and all the five parent packages will pick up the modified child package.

While developing your packages with the modular design concept, you can reuse packages developed as modules to perform specific functions in other packages. You may, for example, develop a send mail package that reads from a table and sends a broadcast mail to all the members in the table. This functionality can be used in any package that requires sending mails. Another example could be auditing for which you can develop a package that records environment values and variable values to an audit table. You can then attach this package as a child package in all of your packages wherever you want to record auditing information.

Using the Execute Package task, you can have better security controls in place. You can control access to sensitive data by dividing your package into packages that can have public access and other packages that can be accessed by administrators or management only. For example, you can separate out salary processing package from your main SSIS

package and run that as a child package using the Execute Package task in the main SSIS package and hence avoid access to salaries data.

Probably the biggest benefit of having an Execute Package task is that the packages are able to communicate with other packages and have an effect on the success or failure of other packages. To clarify, consider the container hierarchy in which errors and events get flagged up the hierarchy with the package being at the top of the hierarchy. When we use a child package in a parent package, the Execute Package task actually contains the child package and becomes a parent container for that child package. Any event occurring in the child package gets passed on to the Execute Package task, which it can share with other Execute Package tasks used in the parent package. Now think of transactions that cause tasks to commit or roll back as a unit within a package. Using this task, transactions can actually span across packages and can cause multiple packages to commit or roll back as a unit. SSIS provides you the benefits of dealing with individual tasks and lets you use them as an independent unit, yet at the same time you are also able to integrate the packages together to work as a unit.

You can run a child package in its own process or in the process of the parent package using the ExecuteOutOfProcess option. If you are following a modular design for your package and want your parent and child packages to commit or fail together as a single unit, you will be using the in-process method—i.e., you will run the child package in the same process as that of the parent package. For this configuration, you will not be running additional processing threads and the memory required by the process will, of course, be less. In this case, the context switching will not happen and you will experience less processing overhead. However, the down side is that if a child package crashes due to some problem, that may also kill the parent package. In addition, if your system has more than 4GB memory, SSIS won't be able to use it, as a single process in 32-bit systems can use maximum of 2GB of virtual memory (or 3GB if you use /3GB switch in boot.ini file). Chapter 13 discusses memory utilization of SSIS packages and pros and cons of using 32-bit versus 64-bit systems in more detail.

Alternatively, if you want to make use of the full memory resources available on the system, or you want to protect the parent package from crashes in the child package due to bugs or other issues, or you want your parent package not to depend on the success or failure of a child package, then you may prefer to use the out-of-process method—i.e., the parent and the child packages will run in their own processes. As the parent package will be using multiple processes, you will see more context switching, due to the overhead of maintaining multiple processes, and the memory usage will also be more—in fact, memory utilization can grow to more than 4GB on a 32-bit system if available on the computer. The following exercise is designed to help you understand how to use the Execute Package task and the implications of connecting various tasks in a package.

Hands-On: Consolidating Workflow Packages

The packages you have developed so far to download, expand, archive, and import files are independent units of work and isolated, too. You want to consolidate all the packages into one package with a defined sequence and configure them with different precedence constraints to achieve a more desirable work flow.

Method

In this exercise, you will use the Execute Package task to embed the given packages in the parent package and join these packages with success constraint. As a second part to this exercise, you will change the constraints and see the effects on execution results.

Exercise (Building Consolidated Package)

In this part, you will create a new parent package in to which you will add four child packages created in the earlier exercises using Execute Package Tasks.

1. Start BIDS and open the Control Flow Tasks project. Create a new package in the Solution Explorer with the name **Consolidating workflow packages.dtsx**.
2. Drag and drop the Execute Package task from the Toolbox on to the designer surface.
3. Double-click the Execute Package task to open the Execute Package Task Editor.
4. Type the following in the General page of the editor:

Name	Downloading zipped files
Description	This task executes the named package.

5. In the Location field on Package page, select File System from the drop-down list. The other option, SQL Server, could be chosen if the package were stored in the MSDB database of SQL Server. As your packages are stored in the file system, you will not be using the SQL Server option here.
6. Click in the Connection field and select <New connection…> to open the File Connection Manager Editor. Leave Existing File in the Usage Type field and type **C:\SSIS\Projects\Control Flow Tasks\Downloading zipped files.dtsx** in the File field to point the Execute Package task to the Downloading Zipped Files package.
7. The PackageName field is available when you select SQL Server in the Location field to allow you to choose the package from the list of packages stored in the MSDB store. In your case, the field is disabled, as the package name has already been provided in the Connection field.

8. You can specify the password in the Password field if the package has been protected with a password. Leave it at the default setting for now. Package protection levels have been covered in detail in Chapter 7.
9. As we are not using any transactions across the packages, change the ExecuteOutOfProcess field value to True as shown in Figure 5-17. Click OK to close the Execute Package Task Editor window.

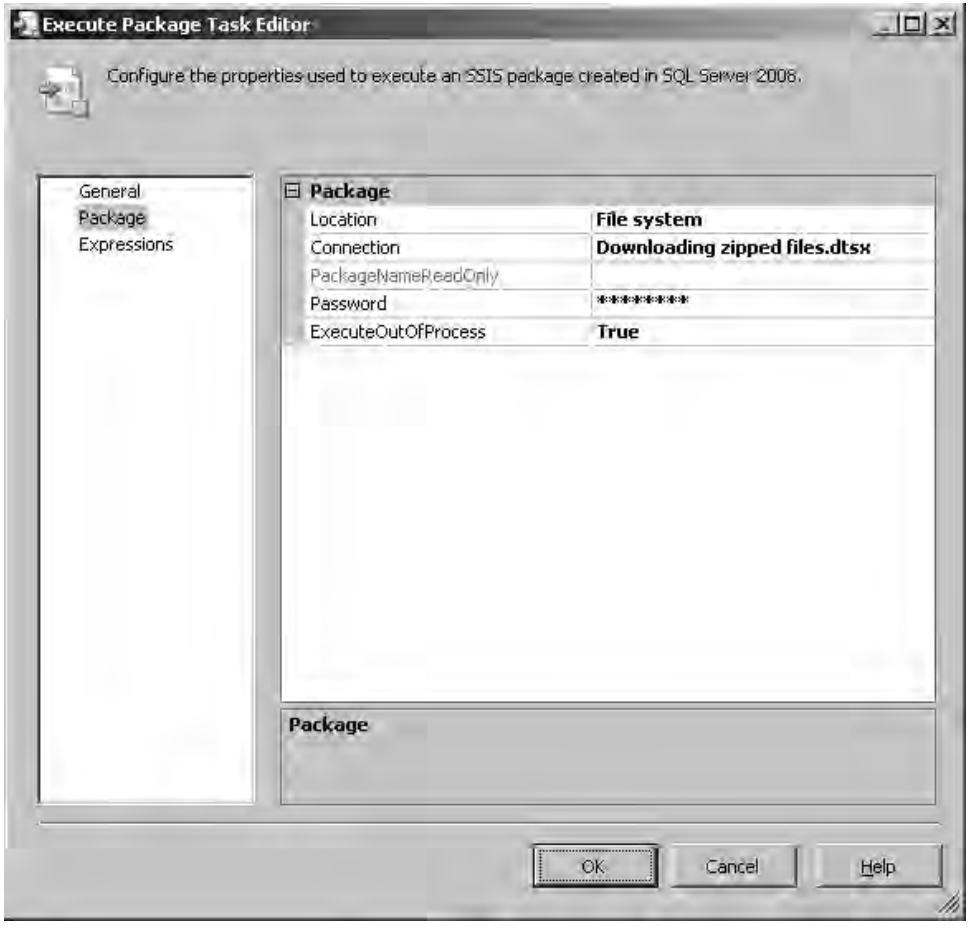


Figure 5-17 *Configuring the Execute Package task*

10. From the Toolbox, drop another Execute Package task on the designer surface just below the Downloading Zipped Files task. Stretch the green arrow from the Downloading Zipped Files task and join it to the new Execute Package task. Now, following Steps 3 to 9, configure this task with the following settings:

Name	Expanding downloaded files
Description	This task executes the named package
Location	File system
Connection	C:\SSIS\Projects\Control Flow Tasks\Expanding downloaded files.dtsx
ExecuteOutOfProcess	True

11. Similarly, add the following packages using the Execute Package task with the following details and connect them using the green arrows.

    For the Archiving Downloaded Files task:

Name	Archiving downloaded files
Description	This task executes the named package
Location	File system
Connection	C:\SSIS\Projects\Control Flow Tasks\Archiving downloaded files.dtsx
ExecuteOutOfProcess	True

    For the Importing Expanded Files task:

Name	Importing expanded files
Description	This task executes the named package
Location	File system
Connection	C:\SSIS\Projects\Control Flow Tasks\Importing expanded files.dtsx
ExecuteOutOfProcess	True

    Your package should look like the one shown in Figure 5-18.

12. Before we run this package, make sure the zipped files are still available on the FTP server in the Sales folder. After checking this, delete the DealerSales01 .txt and DealerSales02.txt files from the C:\SSIS\downloads folder and delete DealerSales01.zip and DealerSales02.zip from the C:\SSIS\downloads\Archive folder. Using SQL Server Management Studio, run the following commands to delete all the rows from the DealerSales table:

```
TRUNCATE TABLE [Campaign].[dbo].[DealerSales]
```

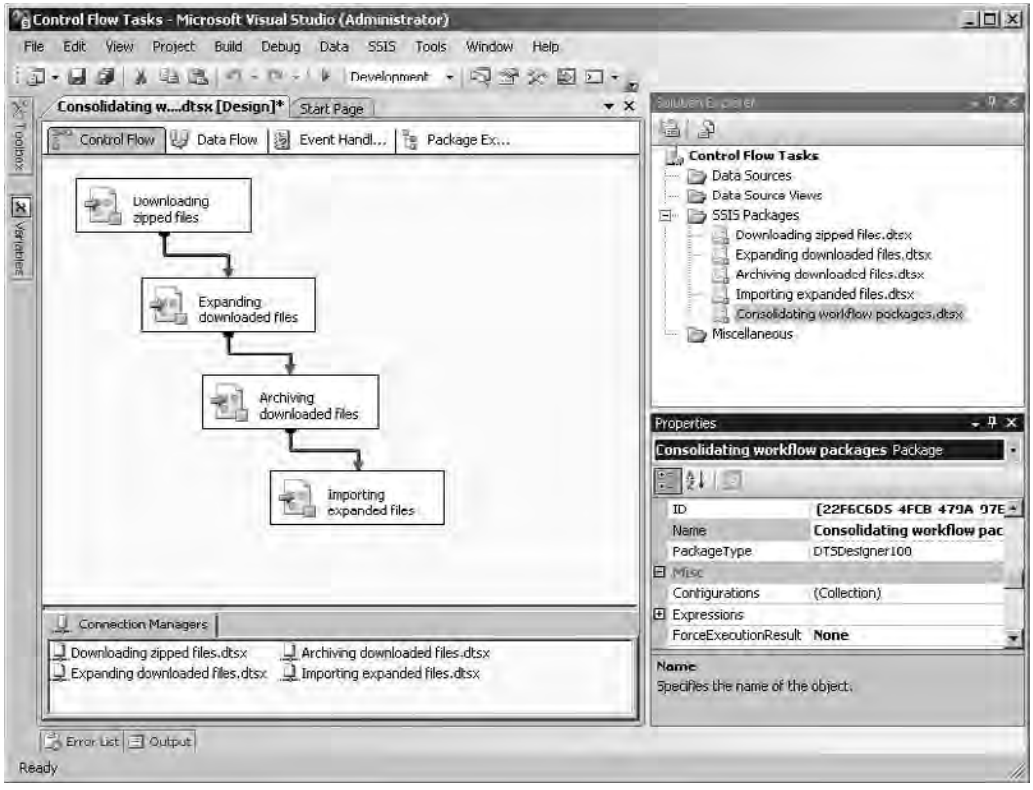


Figure 5-18 *Consolidating the Workflow Packages package*

Now that all the previous files and data have been deleted, run the package by pressing F5. You will see that the defined packages are opened and executed in sequence one after another. Once all four packages have been executed successfully, check the folders to see the files at expected places and the rows loaded into DealerSales table—242,634 in total.

Exercise (Understanding Precedence Constraints)

In this part of the exercise, you will make changes in the ExtractFiles.bat file to fail Expanding Downloaded Files task and then study the behavior of the package execution using success and completion constraint.

13. You have seen that the packages execute successfully. Make the following changes in the ExtractFiles.bat file to fail the Expanding Downloaded Files task:

```
REM DEL C:\SSIS\downloads\%1.txt
C:\SSIS\UNZIP %1.zip %1.txt
```

As you can now understand, when the ExtractFiles.bat file is called, it won't be able to find the unzip.exe file in the C:\SSIS folder and hence will fail in operation.

Using Windows Explorer, delete the DealerSales01.zip and DealerSales02.zip files from C:\SSIS\downloads\Archive folder, but do not delete DealerSales01.txt and DealerSales02.txt files from the C:\SSIS\downloads folder.

14. Right-click the Consolidating Workflow Packages package in the Solution Explorer window and choose the Execute Package command from the context menu. You will see the Downloading Zipped Files package appearing on the screen and being executed successfully, followed by Expanding downloaded files package being executed but failing as expected. Note that after the failure of this child package, the parent package Consolidating Workflow Packages stops immediately and doesn't execute tasks down the line. Stop debugging the package by pressing SHIFT-F5. Note that in the second to last line in the Output window, the package is declared finished with a failure. The exact message is "SSIS package 'Consolidating workflow packages.dtsx' finished: Failure." (See Figure 5-19.)

    If you don't see the Output window, you can open it by pressing CTRL-ALT-O.

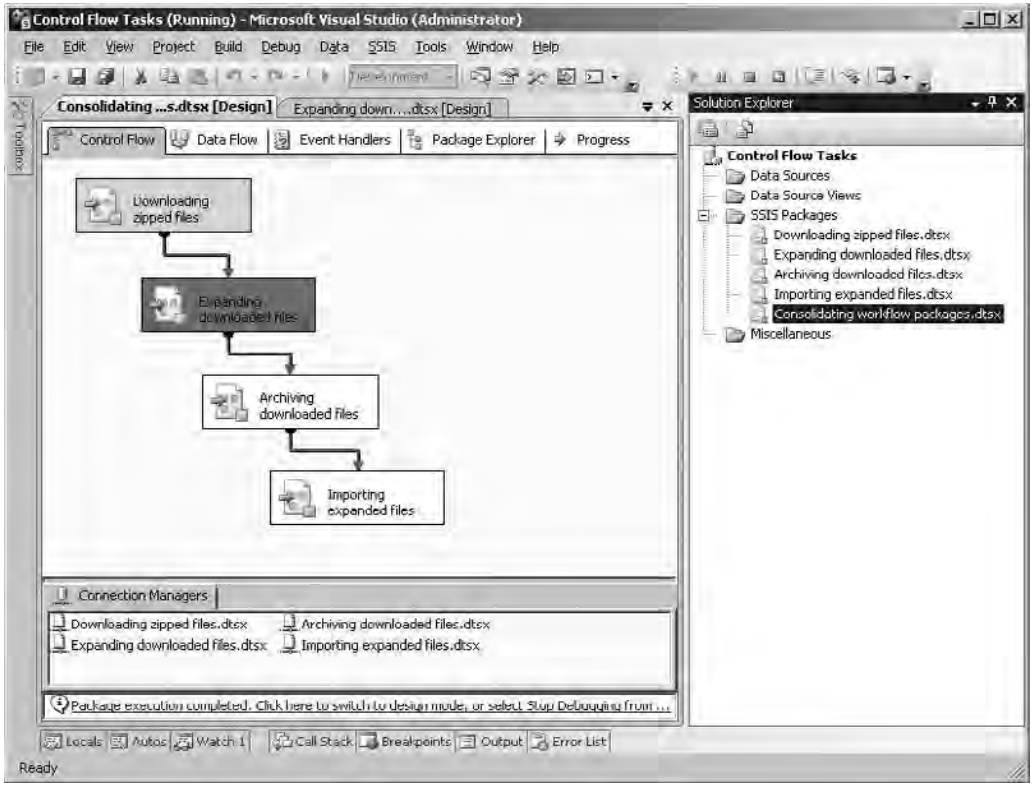


Figure 5-19 *Failing Consolidating Workflow Packages with the Success constraint*

15. Having seen the package fail when using the Success constraint, you will now change the Success constraint to a Completion constraint for the Archiving Downloaded Files task to see how the package behaves. Changing this constraint actually specifies that Archiving Downloaded Files task should run when the Expanding Downloaded Files task completes without regard to success or failure of Expanding Downloaded Files package.

    Right-click the green arrow from the Expanding Downloaded Files task to the Archiving Downloaded Files task and click Completion in the context menu. The green arrow will change to blue. This blue arrow signifies the On Completion constraint for the Archiving Downloaded Files task.

    Execute the Consolidating Workflow Packages package again. This time you will see the first task, Downloading Zipped Files, completing successfully, and then the Expanding Downloaded Files task failing as expected. But your parent package doesn't stop this time; instead, it goes on to run the remaining tasks successfully and loading records in the table as the text files were available (which you didn't delete in Step 13). This explains how the package behaves in case of a Completion constraint compared to a Success constraint. If you check the Output window for status, you will still see the same message you saw last time for the package being finished with a failure, but you do know for sure that this time the last two tasks ran successfully (Figure 5-20).

Review

In the first part of this exercise, you used the Execute Package task to include child packages in the parent package and consolidated all the different modules into one integrated package with all the features you built separately. In the second part you learned the behavior of execution of a package with the Success constraint and later with the Completion constraint. If a task fails during run time for any reason, the following tasks that are using the Success constraint for this package will not be executed and the package will fail immediately. On the other hand, if the failing task connects with following tasks using the Completion constraint, the downstream tasks get executed with no regard to success or failure, though the tasks that depend upon the processing of the failing task may be affected due to unavailability of data that would have been otherwise provided by the failing task. You also learned that the final message for a package might not tell you a true story about the execution status of the package. So, you definitely need to configure logging for your packages to know more about the execution status of the various tasks and the reasons of failure, if any. One thing more about precedence constraints is that you can actually evaluate expressions and use them along with the constraints to determine the execution for the subsequent tasks in a package.

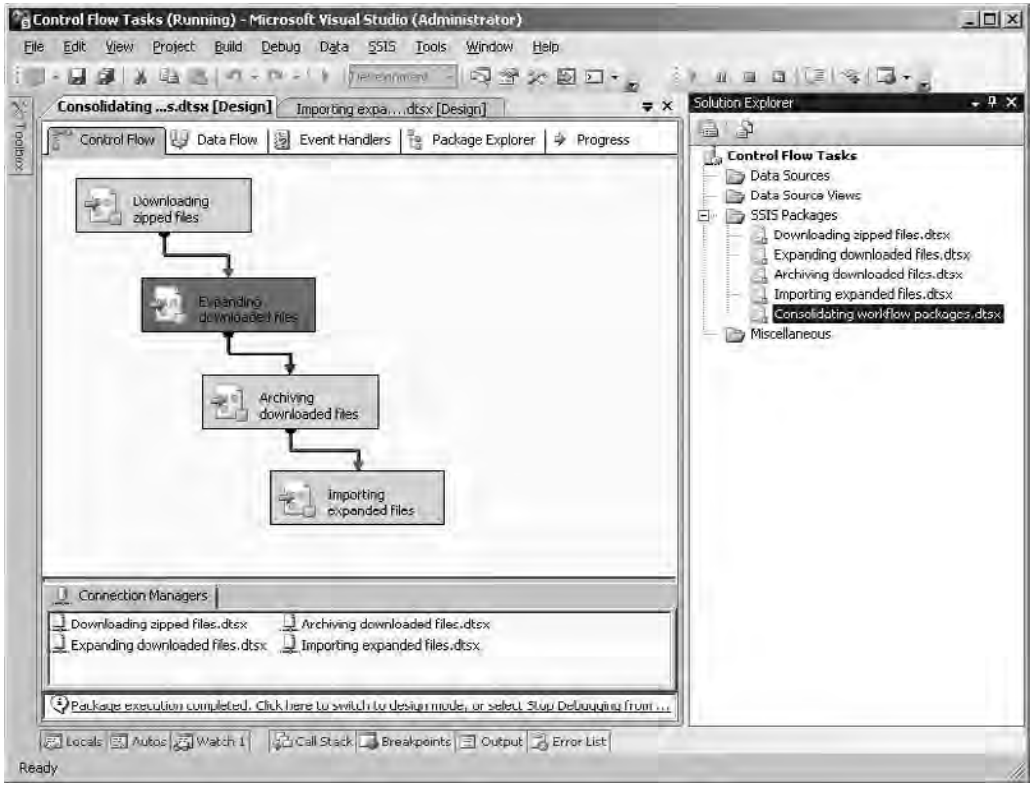


Figure 5-20 *Failing Consolidating Workflow Packages with the Completion constraint*

Send Mail Task

Using the Send Mail task, you can send messages from your packages such as on the basis of success or failure of the package or on the basis of an event raised during execution of the package. This task uses the SMTP Connection Manager to send mails using the SMTP server. You can either specify the message directly in the task, let the task read from a file, or choose a variable to be sent as a message. You can use this feature to pass messages or variables between SSIS packages running on different servers. You can also use the Send Mail task to send notification messages about success or failure of other tasks. You have already used this task in the "Contacting Opportunities" Hands-On exercise in Chapter 4.

WMI Data Reader Task

For the benefit of those who haven't used Windows Management Instrumentation (WMI), it is explained here along with a brief background to give you a head start. The Distributed Management Task Force (DMTF) is an industry consortium that is involved with the development, support, and maintenance of management standards for computer systems and is involved with management technologies such as Common Information Model (CIM) and Web-Based Enterprise Management (WBEM). CIM, a standard for describing management information, allows different management applications to collect the required data from a variety of sources and is platform independent. WBEM uses browsers and applications to manage systems and networks throughout the enterprise. WBEM uses CIM as the database for information about computer systems and network devices. Microsoft has implemented the DMTF's CIMV2 and WBEM standards in WMI.

The WMI schema is logically partitioned into namespaces for organizational and security purposes. You should use the WMI Control (Server Manager | Configuration | WMI Control | Properties) or the Wmimgmt.msc, Microsoft Management Console (MMC) snap-in to view and modify the security on WMI namespaces. A namespace actually groups a set of classes and instances that are logically related to a particular managed environment. For example, CIMV2 groups a set of classes and instances that relate to aspects of the local Windows environment. Though DMTF has defined a lot of namespaces within WBEM, Microsoft has chosen to instrument the various classes and properties that fall within the CIMV2 namespace.

The Windows operating system provides management information through the WMI component. WMI can be used to query computer systems, networks, and applications that can be further extended to create event-monitoring applications.

Using the WMI Data Reader task, you can run WQL queries to get the information from WMI such as the presence, state, or properties of hardware components, Windows event logs, and installed applications; using this you can build some sort of intelligence within your packages to decide, based on the results of a WQL query, whether the other tasks in the package should run.

The WMI Query Language (WQL) is a subset of ANSI SQL with minor semantic changes to support WMI. You can write data, event, and schema queries using WQL. Data queries are most commonly used in WMI scripts and applications to retrieve class instances and data associations, whereas schema queries are used to retrieve class definitions and schema associations and event queries are used to raise event notifications.

The good news is that writing a WQL query is similar to writing an SQL query because they use a common dialect. WQL is modified to support WMI event notification and other WMI-specific features. However, the tough bit is that WMI classes vary between versions of Windows and the WQL queries used here may not work on your system, as they have been tested only on a Windows Server 2008 machine.

Let's do a quick and short Hands-On exercise to demonstrate how to configure the WMI Data Reader task and write WQL queries.

Hands-On: Reading the Application Log

You are required to copy the application log error messages for all the failing SSIS packages to a text file.

Method

In this exercise, you will use the WMI Data Reader task to read an application log for error messages generated by Integration Services after January 1, 2010.

Here's the step-by-step procedure:

1. Create a WMI Connection Manager to connect to the server.
2. Write a WQL query to read the application log.
3. Complete configurations of the WMI Data Reader task.

Exercise (Create WMI Connection Manager)

Now you know the steps you have to use, so let us get going.

1. Open the Control Flow Tasks project in BIDS. Add a new package to the SSIS Packages folder in the Solution Explorer window. Rename the package **Reading Application Log.dtsx**.
2. Drag the WMI Data Reader task from the Toolbox onto the SSIS Designer. Double-click the task icon to open the WMI Data Reader Task Editor. On the WMI Options page, click in the WmiConnection field and choose <New WMI Connection…> from the drop-down list box. This will open the WMI Connection Manager Editor. Type the following in the Connection Manager Information area:

Name	Localhost
Description	WMI connection to localhost

Leave the following default settings in the Server and namespace area (refer to Figure 5-21):

Server name	\\localhost
Namespace	\root\cimv2

Select the Use Windows Authentication check box and click Test to test the connection. When you receive the success message, click OK to close the message and click OK again to close the WMI Connection Manager Editor.

Figure 5-21 *The WMI Connection Manager configured to connect to CIMV2*

3. To specify the source from where the task can access the WQL query, you can choose from the Direct input, File connection, or Variable options. Leave the Direct Input option selected in the WqlQuerySourceType field.
4. Type in the following query in the WqlQuerySource field and click OK to return to the task editor.

```
SELECT ComputerName, EventCode, Logfile, Message, SourceName,
TimeGenerated, Type
FROM Win32_NTLogEvent
WHERE Logfile = 'Application'
AND SourceName = 'SQLISPackage100'
AND Type = 'Error'
AND TimeGenerated > '20100101'
```

5. In the OutputType field, choose Data Table.
6. In the OverwriteDestination field, choose Overwrite Destination, as you will be running this test package many times. However, you should choose this option carefully in the production environment.
7. In the DestinationType field, you can specify to write the data to a file or to a variable. Choose File Connection in this field (see Figure 5-22).
8. In the Destination field, choose <New Connection…> and select C:\SSIS\RawFiles\ApplicationLog.txt as the existing file in the File Connection Manager Editor. This is a blank file used as a placeholder for Application Log data

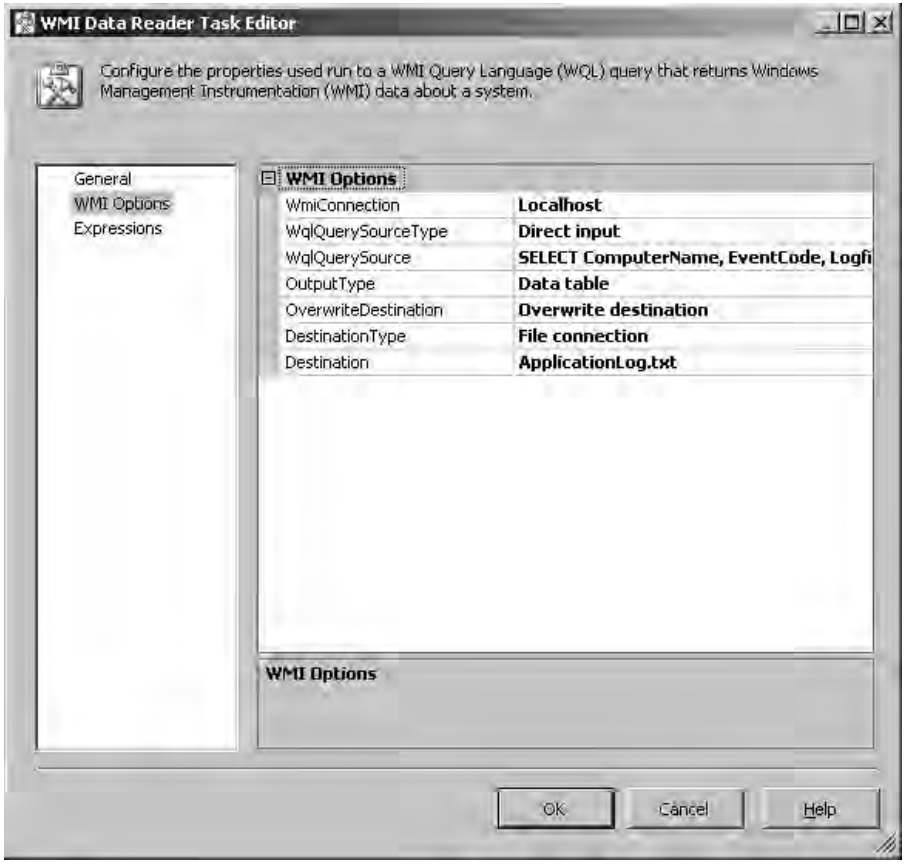


Figure 5-22 *Configuring the WMI DataReader task to read an application log*

enclosed within the folder. If you don't find this file, create a blank text file in this folder using Notepad. Click OK twice to complete the configurations.

9. Press F5 to start debugging the package. If you have not made any typos and the WQL configured is compatible with your system, the task will run successfully. Explore to the C:\SSIS\RawFiles folder and open the ApplicationLog.txt file; you will see the contents of the Windows Application log extracted in the text format.

Review

In this exercise, you learned how to configure and use the WMI Data Reader task. You also learned about some basic concepts and terminology related to WMI. Developers use Visual Basic scripting to exploit benefits of WMI. However, while scripting for WMI, keep in mind that the different versions of Windows operating system have

different sets of WMI classes in the repository. Using the WMI Data Reader task, you can harness the benefits of WMI yet avoid VB scripting using much simpler WQL language to write queries. One example could be building a workflow within your packages based on the environment conditions; for instance, you might want to check available free memory or free space on your hard disk at run time and build precedence constraints accordingly. The following two sample queries check the freespace on the disk or free available memory.

```
SELECT FreeSpace FROM Win32_LogicalDisk WHERE DeviceID = 'C:'
SELECT FreePhysicalMemory FROM Win32_OperatingSystem
```

Selecting OutputType as the Property value and the DestinationType as the Variable in the WMI Data Reader Task Editor will allow you to capture the value to a variable. Later, you can use this variable in creating an evaluation expression within precedence constraints to decide whether to continue or stop or divert the workflow to a different branch within the package. This is quite underutilized yet a powerful feature that allows you to test the environment conditions at run time and automatically run different branches based on the results of the WQL queries. Refer to Microsoft SQL Server 2008 Books Online to see more examples of the WQL queries that can be used with this task.

WMI Event Watcher Task

Using the WMI Event Watcher task, you can configure your SSIS package to watch for an event and, based on the event occurrence, decide whether to run the package or raise an alert using underlying WMI technology. This is a powerful feature, though it requires your skills to write WQL event queries to specify the events you want to watch for. You can use this task in situations such as the following:

- To check for availability of enough resources (disk space, memory, and so on) on the computer system before running the package
- To watch for files being added to a folder before starting the package
- To wait for an e-mail (from another package) to arrive with particular subject before initiating the processing of the SSIS package
- To wait for memory to be available before starting another process

Here is the step-by-step procedure to configure this task:

1. After you have placed this task on the Designer panel and started editing, you can specify a name and description for the task on the General page.
2. On the WMI Options page, specify a WMI Connection Manager in the WmiConnection field.

3. Specify the source from which the task can read the WQL query in the WqlQuerySourceType field by selecting from the available options, Direct Input, File Connection, or Variable. Depending on your choice, the WqlQuerySource field changes. For example, when you choose Direct Input, you can specify the WQL query in the WqlQuerySource field.
4. In the ActionAtEvent field, you can specify the action the task should take when the event occurs. You can choose only to log the event notification and the status after the event, or you can choose to fire an SSIS event with the event notification and the status logged after the event (see Figure 5-23).

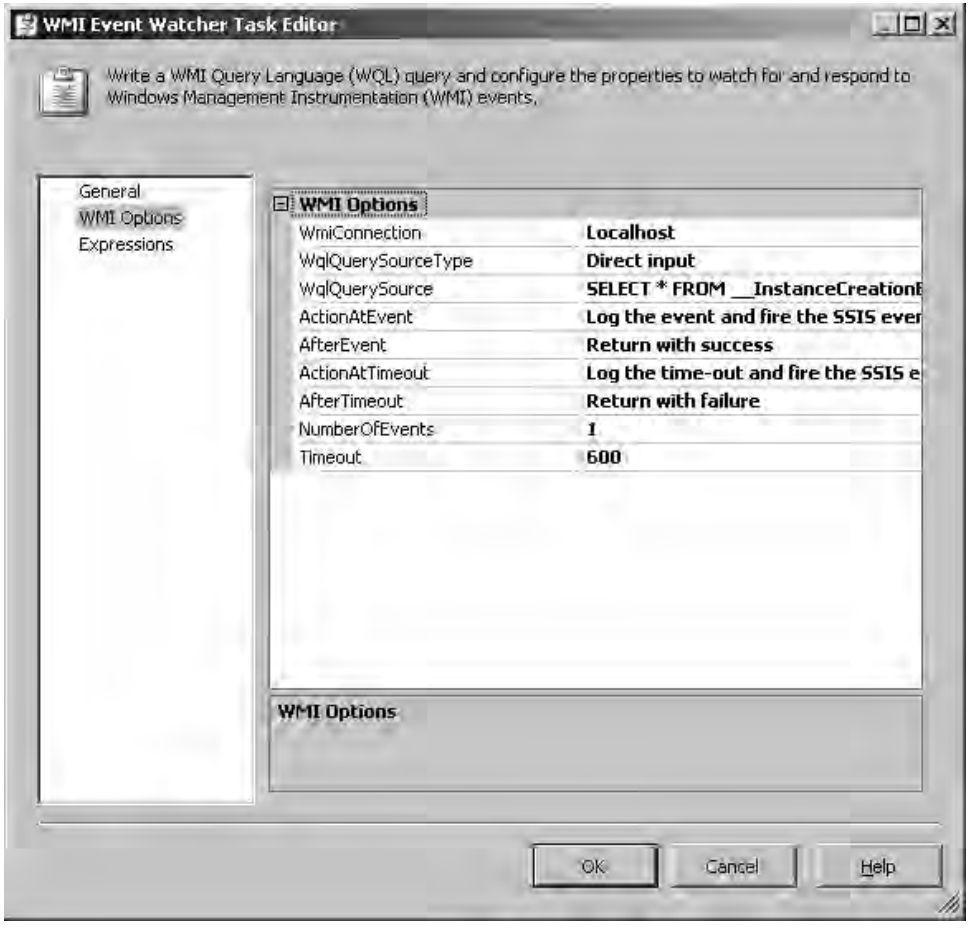


Figure 5-23 *Configurations of the WMI Event Watcher task*

5. In the AfterEvent field, you can specify how the task should respond after the event. The task can be configured to return with success or failure depending on the outcome of the task, or you can configure the task to watch the event again.
6. The next two options, ActionAtTimeout and AfterTimeout, specify how the task should behave and respond when the WQL query times out.
7. In the NumberOfEvents field, you can specify the number of times the task should watch for the event.
8. Lastly, specify the time-out setting for the WQL query in seconds.

Transfer Database Task

The Transfer Database task transfers a database from the source SQL Server to the destination SQL Server; you can copy or move your database. This task can use the DatabaseOnline or DatabaseOffline method of transfer. When you choose online transfer of the database, the source database stays online even during transfer and can respond to user queries; in the offline method, the source database is detached and the database files are copied over to the destination server. When using the DatabaseOffline method, you can specify to reattach the source database or leave it offline. The user who runs this task to transfer a database must be a member of sysadmin server role in either offline or online mode; however, if you are using online mode, the database owner (dbo) role is sufficient to transfer the database.

You can configure the Transfer Database task in the following ways:

- The Transfer Database task has three pages: General, Databases, and Expressions. You specify a name and description in the General page.
- On the Databases page, specify a SourceConnection and a DestinationConnection to the source and destination servers using the SMO Connection Manager in the Connections section.
- Specify the Action from the drop-down list to copy or move the database and select the DatabaseOnline or DatabaseOffline transfer method in the Method field. Then choose the database you want to transfer from the drop-down list provided in the SourceDatabaseName field. If using DatabaseOffline method, you'll need to specify the files for the database in the SourceDatabaseFiles field and a True or False value in the ReattachSourceDatabase field (see Figure 5-24).
- In the Destination Database section, specify the appropriate values in the DestinationDatabaseName and the DestinationDatabaseFiles fields. Finally, you can choose to overwrite the files if they exist at the destination.
- Use the Expressions page to modify any property at run time using property expressions.

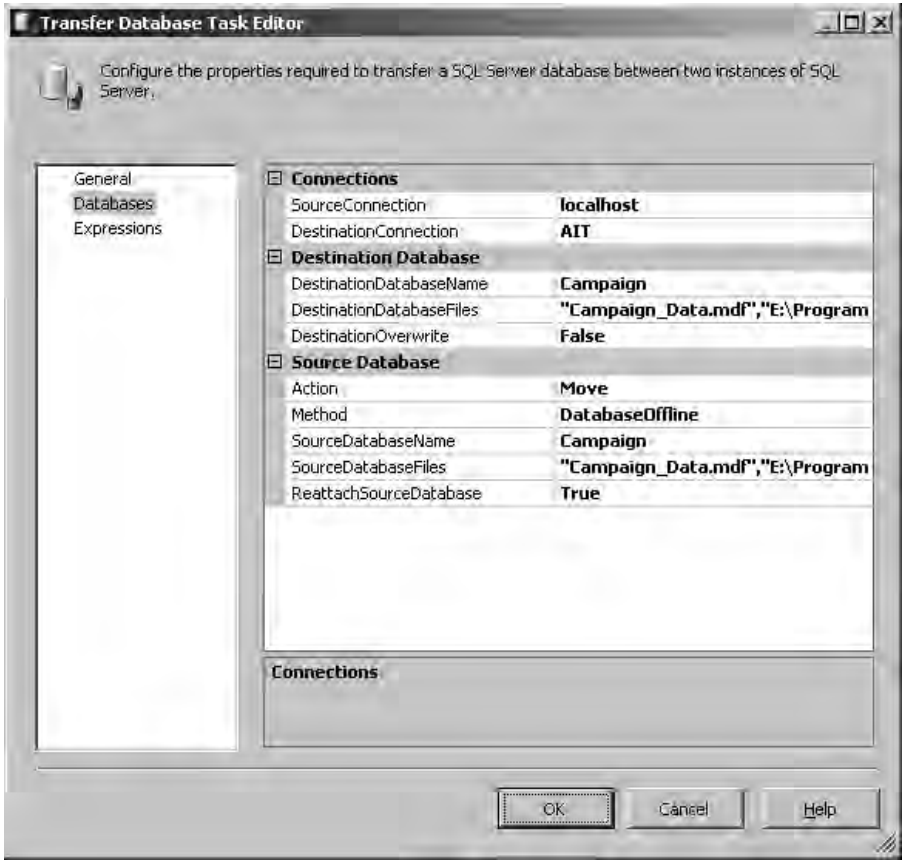


Figure 5-24 *Configurations for the Transfer Databases task*

Transfer Error Messages Task

The Transfer Error Messages task copies one or more SQL Server user-defined error messages from one SQL Server to another. Once you define error messages on one SQL Server for your application, you might like to make those error messages available at all the servers wherever you install your application. This task helps avoid the rework in creating these error messages. You create the error messages on the first server by using the sp_addmessage stored procedure and use this task to copy the defined error messages to the remaining servers. These messages can be viewed using the sys.messages catalog view.

You can configure the task as follows:

1. The Transfer Error Messages task editor has three pages—General, Messages, and Expressions. Specify a name and a description for this task in the General page.
2. In the Messages page, specify a SourceConnection and DestinationConnection to the source and destination servers using the SMO Connection Manager in the Connections section.
3. You can choose FailTask, Overwrite, or Skip in the IfObjectExists field; in case the error message already exists at the destination server (see Figure 5-25).

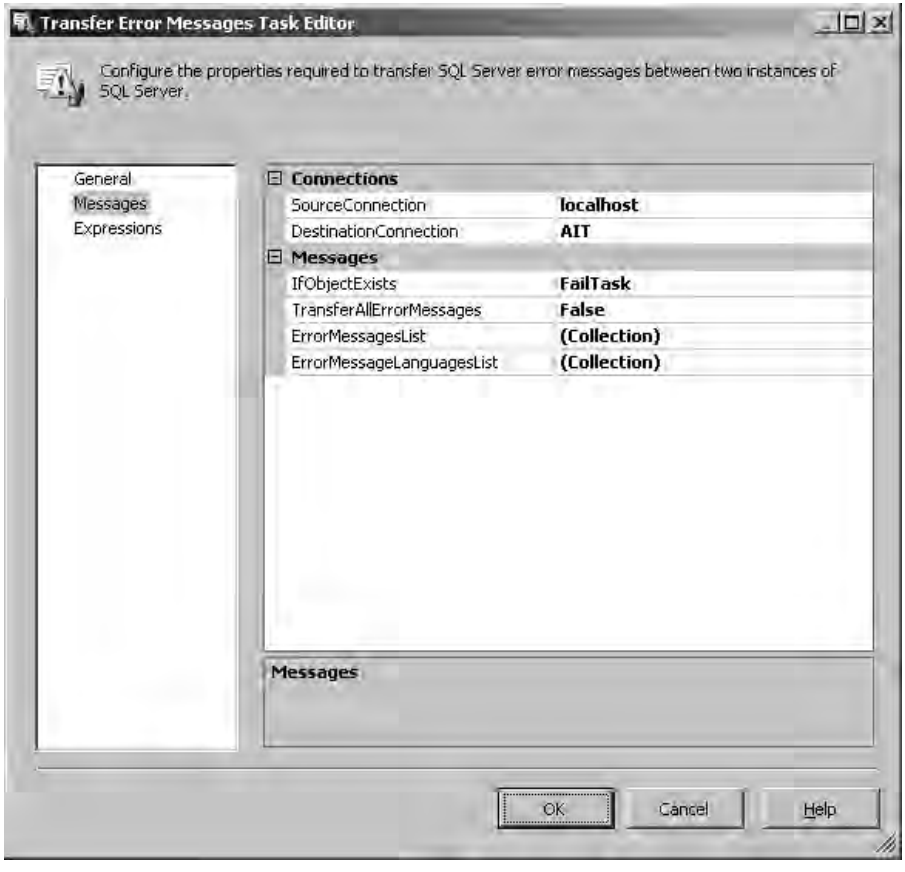


Figure 5-25 *Configuration options for the Transfer Error Messages task*

4. Then you can choose to transfer all error messages by selecting True in the TransferAllErrorMessages field. If you intend to transfer a subset of the messages, you will select False in this field, in which case, the ErrorMessagesList and ErrorMessageLanguagesList fields become available to you. Specify the error messages you want to transfer by selecting from the collection in the ErrorMessagesList field. You can also select the languages for which you want to transfer user-defined error messages in the ErrorMessageLanguagesList field.
5. Use the Expressions page to modify any property at run time using property expressions.

Transfer Jobs Task

The Transfer Jobs task copies SQL Server Agent jobs from the source SQL Server instance to the destination SQL Server instance. This task uses the SMO Connection Manager to connect to the source and destination SQL Server instances. The task is quite straightforward to configure as explained here:

1. Open the editor window for the task and specify the name and description in the General page.
2. Most settings are configured in the Jobs page. Specify a SourceConnection and a DestinationConnection to connect to source and destination servers using the SMO Connection Manager in the Connections section.
3. In the Jobs area, you can specify to copy all the jobs to the destination SQL Server or select from the list provided in the JobsList field. The selected jobs will be displayed as a collection in the field (see Figure 5-26).
4. After selecting the jobs you want to transfer, you can fail the task, overwrite, or skip the jobs if jobs of the same name already exist on the destination SQL Server using the IfObjectExists field.
5. You can also choose to enable the jobs at the destination SQL Server using EnableJobsAtDestination field.
6. As with other tasks, you can use the Expressions page to modify any property at run time using property expressions.

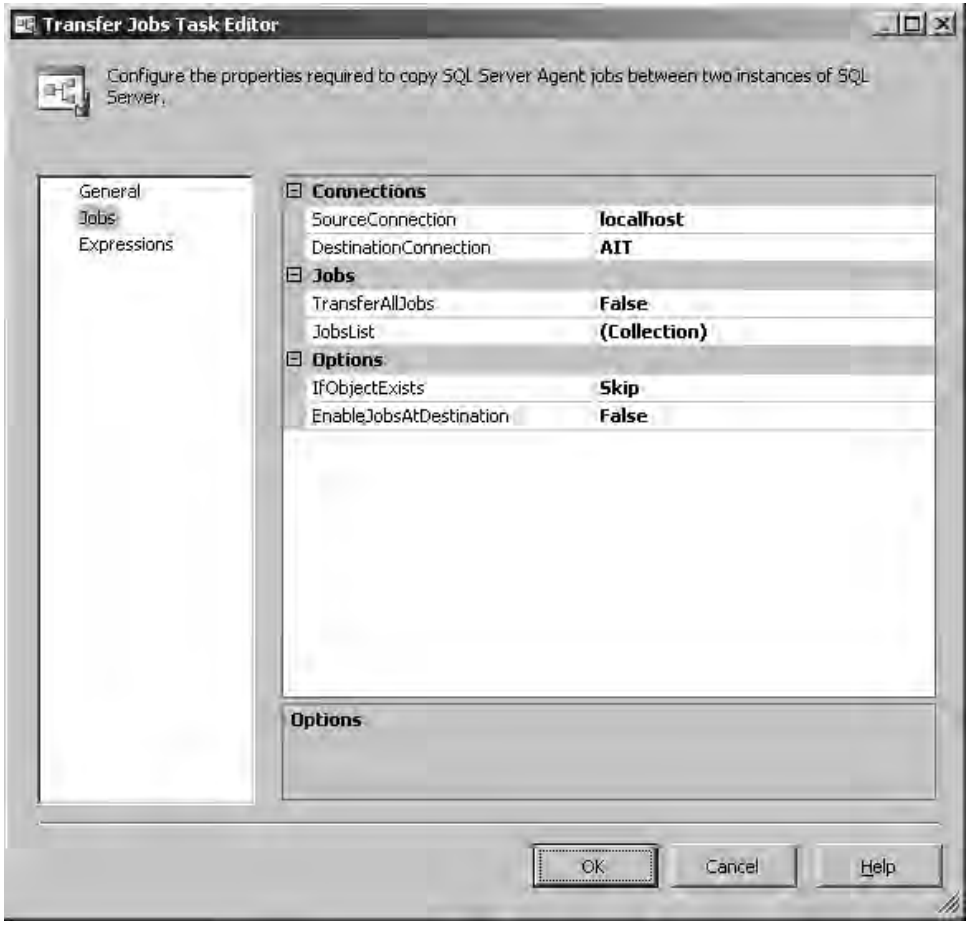


Figure 5-26 *Configuration options for the Transfer Jobs task*

Transfer Logins Task

The Transfer Logins task allows you to copy logins from the source SQL Server instance to the destination SQL Server instance. You can choose to transfer all the logins on the server or the selected logins on the server or all the logins for the selected databases. The following steps explain the configurations of this task:

1. Specify a name and description in the General page.
2. Specify a SourceConnection and a DestinationConnection to connect to source and destination servers using the SMO Connection Manager in the Connections section.

3. In the Logins section, choose from the three options to specify the logins you want to transfer in the LoginsToTransfer field:

AllLogins	Choose to transfer all logins from the source SQL Server instance to the destination SQL Server instance. When you select this option, the subsequent two fields LoginsList and DatabasesList are disabled.
SelectedLogins	Choose to transfer only the selected logins from the source SQL Server instance to the destination SQL Server instance. When you select this option, the LoginsList field becomes available and you can select the logins to transfer from the list provided in this field (see Figure 5-27).
AllLoginsFromSelectedDatabases	Choose this option to transfer all the logins from the selected databases. When you select this option, the DatabasesList field becomes available and you can select the databases from the list provided in this field for which you want to transfer the logins.

4. Finally, in the Options area, you can choose to fail the task, overwrite the logins, or skip the logins if they already exist in the destination server using the IfObjectExists field. You can also choose to copy the SIDs associated with the logins to the destination SQL Server instance using the CopySids field. As you may know, each login is assigned a unique security identifier in SQL Server. The database users are then linked to SQL Server logins using these SIDs. When you transfer a database to a different server, the database users get transferred with the database, but their associated logins are not transferred, as logins do not reside in the database context and creating new logins even with the same name on the destination server would not help as the newly created logins will be getting their own SIDs. To map logins on the destination server with database users, you will need their SIDs. So, when you transfer a database using Transfer Database task and subsequently use this task to transfer logins associated with the transferred database users, you must set the CopySids property to True so that the transferred database users can be mapped to the transferred logins. If you do not set the CopySids value to True, the destination server will assign new SIDs to the transferred logins and the database will not be able to map users to these logins.
5. You can use the Expressions page to modify any property at run time using property expressions.

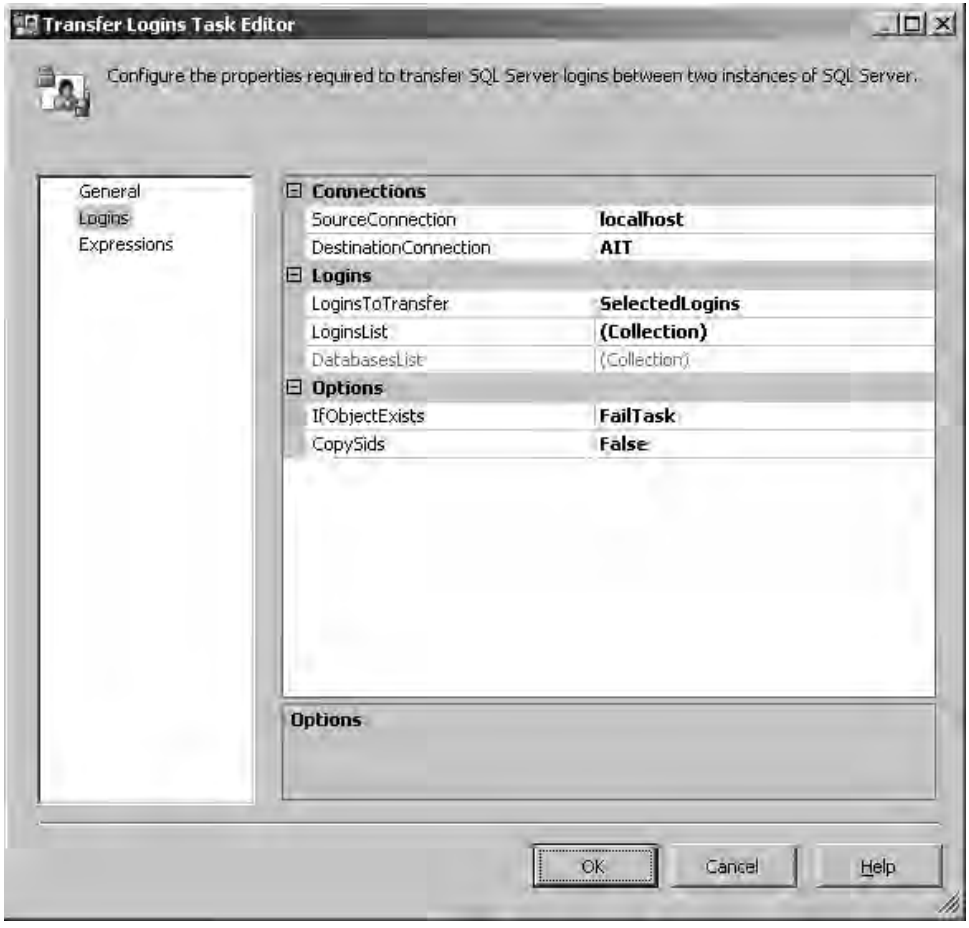


Figure 5-27 *Configuration options for the Transfer Logins task*

Transfer Master Stored Procedures Task

The Transfer Master Stored Procedures task copies user-defined stored procedures that are stored in the master database of the source SQL Server instance to the master database of the destination SQL Server instance. Following are the steps you would take to configure this task:

1. Specify the name and the description for the task in the General page of the editor window.
2. Specify the source and destination connections using the SMO Connection Manager.

3. Choose to copy all or specific stored procedures defined in the master database in the TransferAllStoredProcedures field. You can select the stored procedures from the list in the StoredProceduresList field if you choose to transfer only the specific stored procedures (see Figure 5-28).
4. Finally, you can choose to fail the task or overwrite the stored procedures or skip over the stored procedures that already exist at the destination.
5. The Expressions page can be used to modify any property at run time using property expressions.

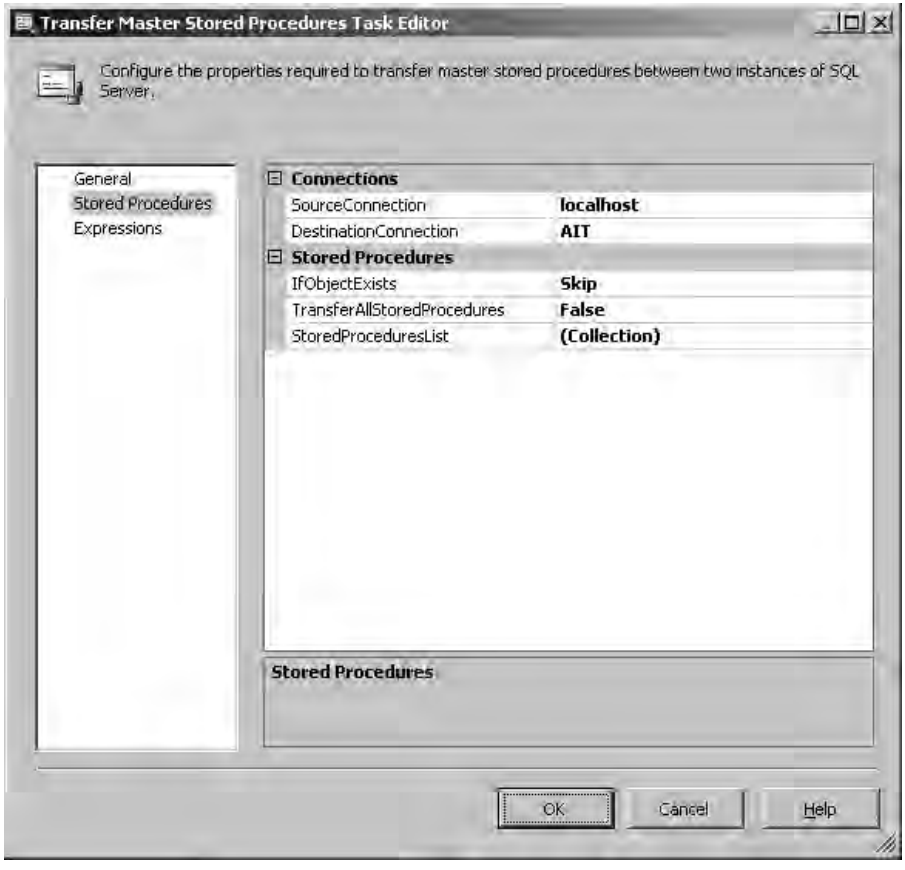


Figure 5-28 *Configuration options for Transfer Master Stored Procedures task*

Transfer SQL Server Objects Task

The Transfer SQL Server Objects task transfers SQL Server objects such as Schema, Tables, Primary Keys, and so on, along with other types of objects from the source SQL Server instance to the destination SQL Server instance. Following are the steps you perform to configure the options for this task:

1. Specify the name and the description for the task in the General page of the editor window.
2. In the Objects page, configure source connection and the destination connection using the SMO Connection Manager. Choose the source database and the destination database from the list provided in the database fields of the relevant connections.
3. All the remaining options are binary options for which you select True or False to indicate the acceptance for the option or otherwise. For the options that you don't want to apply to all the objects, select False as a choice. The default is False in all the options.
4. In the Destination Copy Options section, indicate whether you want the task to copy all objects or specific objects. If you select False, the ObjectsToCopy field becomes available for you to choose the objects you want to copy. When you expand this field, you will see a list of objects categorized based on the type of object. For each of the object types in the category list, you can choose either to copy all the objects of that type or to select from the collection provided in the List field. The object types you can choose here are Tables, Views, Stored Procedures, User Defined Functions, Defaults, User Defined Data Types, Partition Functions, Partition Schemes, SQL Assemblies, User Defined Aggregates, User Defined Types, XML Schema, and Schema (see Figure 5-29).
5. In the Destination section, you can choose to drop objects at the destination server before the transfer, determine whether the extended properties are to be transferred, choose to copy a schema or to include a collation in the transfer, and choose whether to include the dependent objects in the transfer. You can choose to copy data and then can also select to replace the existing data or append to the existing data.
6. In the Security section, you can select whether to transfer the database users, database roles, SQL Server logins, and object-level permissions.
7. In the Table options section, you can select to include Indexes, Full-Text Indexes, Triggers, Primary Keys, Foreign Keys, and all data referential integrity (DRI) objects in the transfer. You can also specify whether the script generated by this task to transfer the object is in Unicode format.
8. Finally, you can use the Expressions page to modify any property at run time using property expressions.

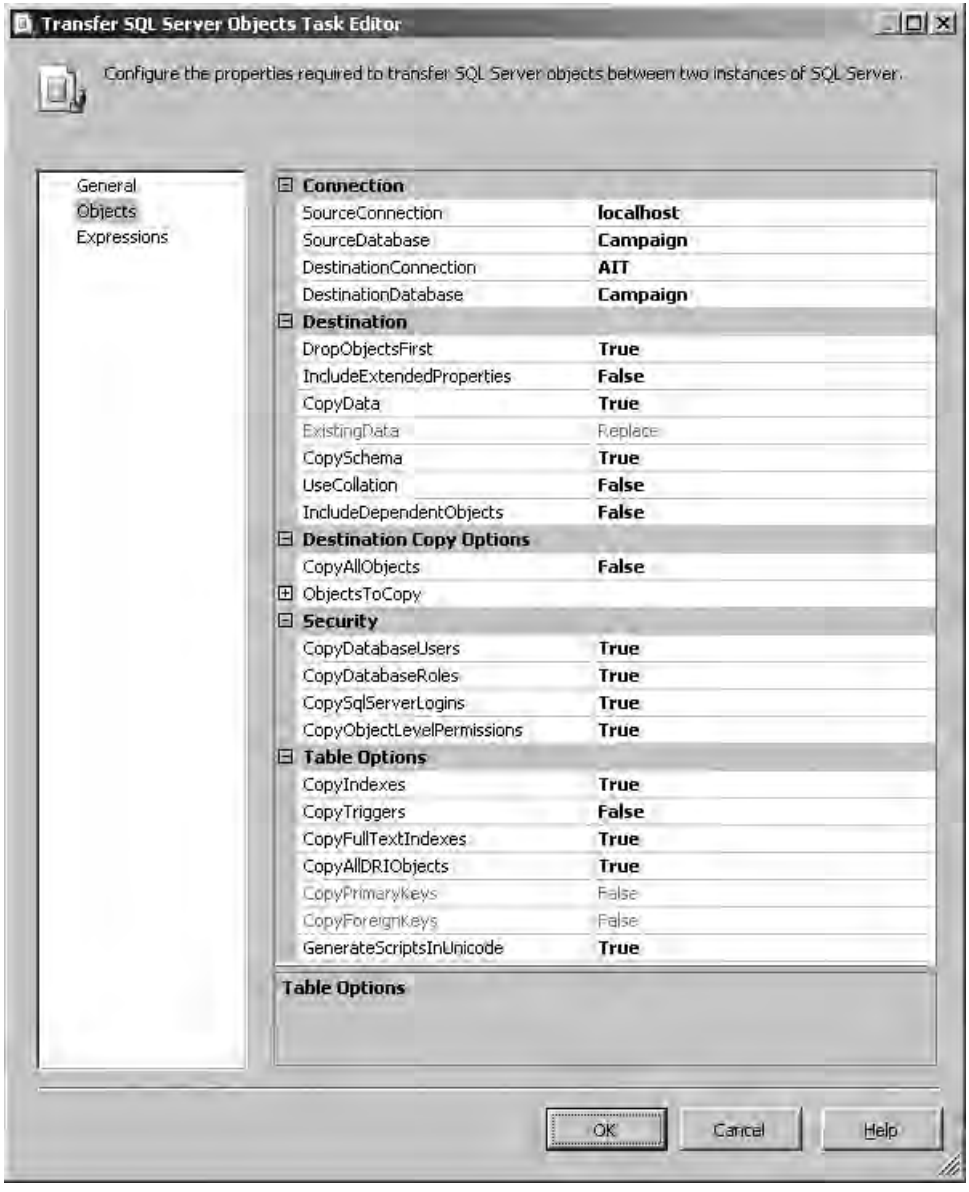


Figure 5-29 *Configuration options for Transfer SQL Server Objects task*

Back Up Database Task

The Back Up Database task uses the BACKUP DATABASE statement to perform different types of SQL Server database backups. Using the Back Up Database task, a package can back up a single database or multiple databases. Following is the step-by-step method to configure this task:

1. Specify a connection to the SQL Server using the ADO.NET Connection Manager in the Connection field.

   Choose the databases you want to back up in the Databases field from the available options: All databases, System databases (master, msdb, model), All user databases (excluding master, msdb, model, and tempdb) or select databases from the available list. You can choose to ignore the databases that are not online at the time this task is run.
2. Specify the Backup type by selecting from the available options of Full, Differential, or Transaction Log.
3. If you've selected a single database for backup, you can choose the backup component by choosing Database or its files and filegroups.
4. Specify the destination where you want to back up the database. The available options are Disk and Tape.
5. Choose to back up databases across one or more files or choose to back up each database to a separate file (see Figure 5-30). Configure available options in either method.
6. Select to verify the integrity of the backup files.
7. You can also choose to compress the backup based on the default server setting or as specified in the task. You can see the T-SQL command that this task will send to the server by clicking View T-SQL.

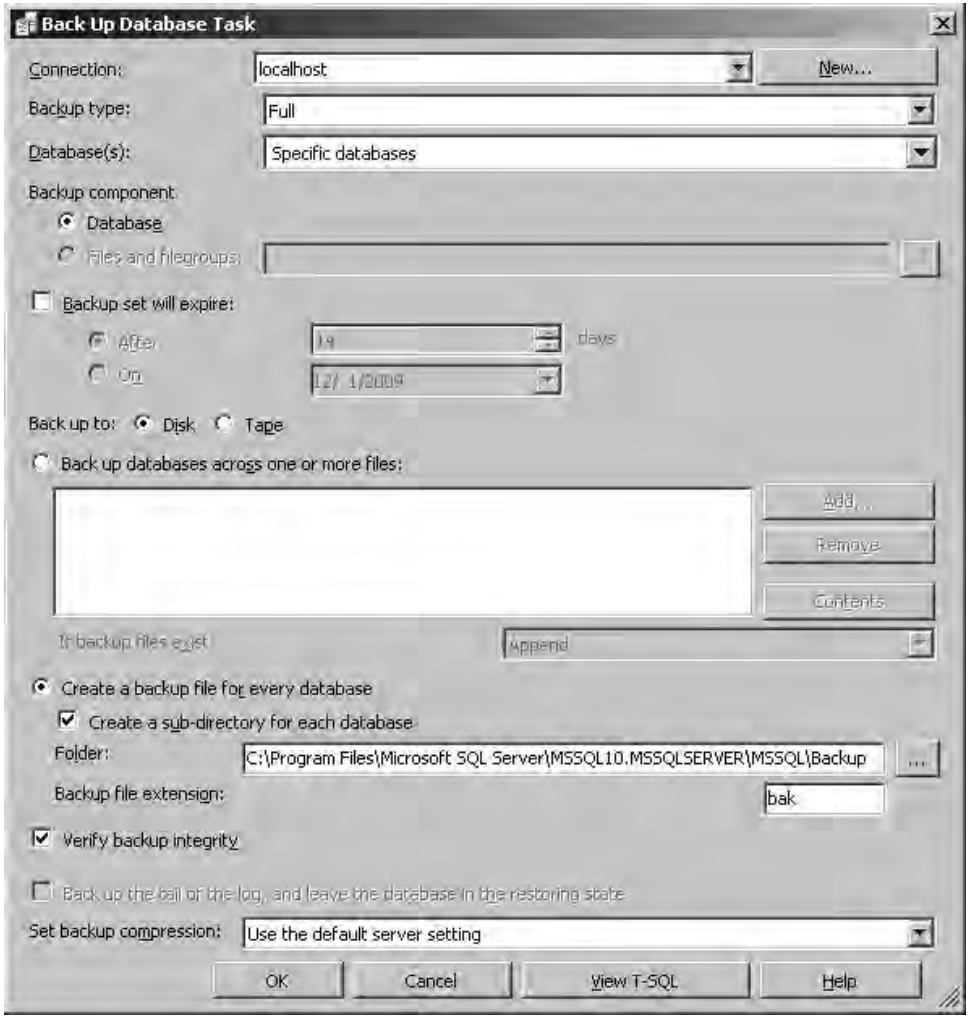


Figure 5-30 *Configuration settings for the Back Up Database task*

Check Database Integrity Task

The Check Database Integrity task uses DBCC CHECKDB statement to check the allocation and structural integrity of all the objects in the specified databases.

The GUI interface of the task is simple to configure (refer Figure 5-31):

1. Specify a connection to the SQL Server using the ADO.NET Connection Manager in the task editor.

Figure 5-31 *Checking the Database Integrity Task interface*

Choose the databases from Databases field. When you click the down arrow in the field, following list of options appear: All Databases, System Databases (master, msdb, model), All User Databases (excluding master, msdb, model, and tempdb) or select databases from the provided list. You can choose either a single database or multiple databases and can also choose to ignore the databases that are not online at the time this task is run.

2. Finally, you can also specify whether to include the database indexes in the integrity check as well.

Execute SQL Server Agent Job Task

The Execute SQL Server Agent Job task runs jobs configured in the SQL Server Agent. You can create a variety of jobs under the SQL Server Agent. For example, the jobs can be ActiveX script jobs or replication jobs, which can be run from within the SSIS package using this task. The SQL Server Agent is a Windows service and must be running for jobs to run automatically.

The task interface is intuitive and easy to configure, as you can make out from Figure 5-32. After you specify a connection to the SQL Server, the task reads the SQL Server Agent jobs from the specified SQL Server and lists them under the Available SQL Server Agent Jobs area. You then select the job you want to run from the list provided.

Figure 5-32 *Interface for the Execute SQL Server Agent Job task*

Execute T-SQL Statement Task

Using the Execute T-SQL Statement task, you can run Transact-SQL (T-SQL) statements against the specified database. This task supports only the T-SQL version of the SQL language and cannot be used to run statements on the servers that use other dialects of the SQL language. The Execute T-SQL Statement task cannot run parameterized queries and cannot save result sets to variables. This is a lightweight task and is mainly designed to run maintenance T-SQL commands against SQL Server. While SSIS provides many other maintenance tasks for SQL Server, having an ability to send T-SQL commands directly to the SQL Server enables you to achieve the desired results quickly and efficiently.

This task is quite simple to configure, as you can see from Figure 5-33. You specify a connection to the SQL Server in the Connection field and type your T-SQL command in the space provided. Optionally, you can specify an execution timeout value to the task.

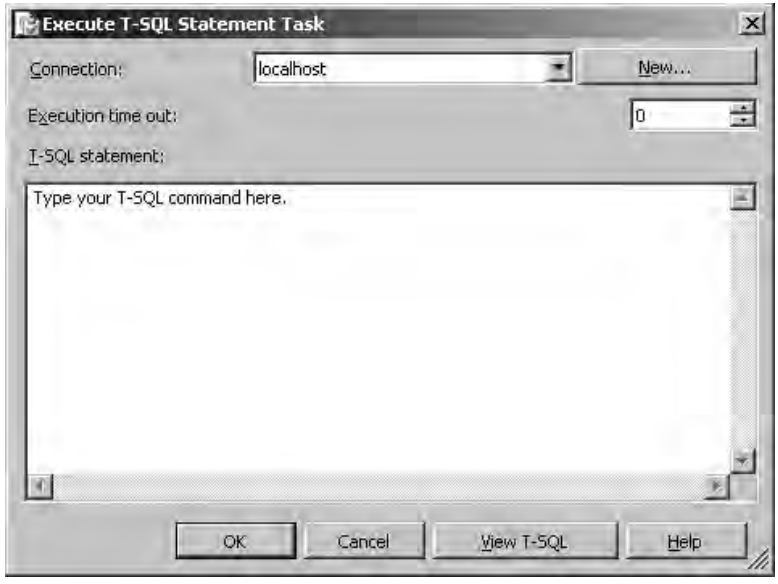


Figure 5-33 *Interface for the Execute T-SQL Statement task*

History Cleanup Task

When the SQL Server Agent runs a job, it keeps the history about each run in the MSDB database. Over a considerable period of time the job history can fill up the MSDB database with lot of unwanted data. The tables of the MSDB database that get affected are backupfile, backupfilegroup, backupmediafamily, backupmediaset, backupset, restorefile, restorefilegroup, and restorehistory. The History Cleanup task enables a package to delete the historical data stored in these tables of MSDB database for backup and restore activities, SQL Server Agent jobs, and database maintenance plans. Here's the step-by-step configuration method for this task:

1. Connect to the SQL Server using the ADO.NET (SqlClient Data Provider) Connection Manager.
2. Choose the historical data that you want to delete. The available options are: Backup and restore history, SQL Server Agent job history, and Maintenance plan history (Figure 5-34).
3. Specify the time period for which you want to retain the data. You can specify the time period by number of days, weeks, months, or years from the date the task is run.

Figure 5-34 *History Cleanup task options*

Maintenance Cleanup Task

While the History Cleanup task removes the stale data such as job history or backup and restore activity from the tables in the MSDB database, the Maintenance Cleanup task deletes database backup files or Maintenance Plan text report files for the time period specified.

Here are the steps you can take to configure this task:

1. In the user interface, specify a connection to the server using the ADO.NET (SqlClient Data Provider) Connection Manager.
2. Choose the type of files you want to delete from the available backup files or Maintenance Plan text reports options (Figure 5-35).
3. Specify a folder to look for the type of files selected in the preceding step, or point the task to the specific file. Optionally, you can choose to include first-level subfolders.
4. Specify a value for the time period for which you want to keep files. You can specify this value by providing a number, and unit of time in day, week, month, or year.

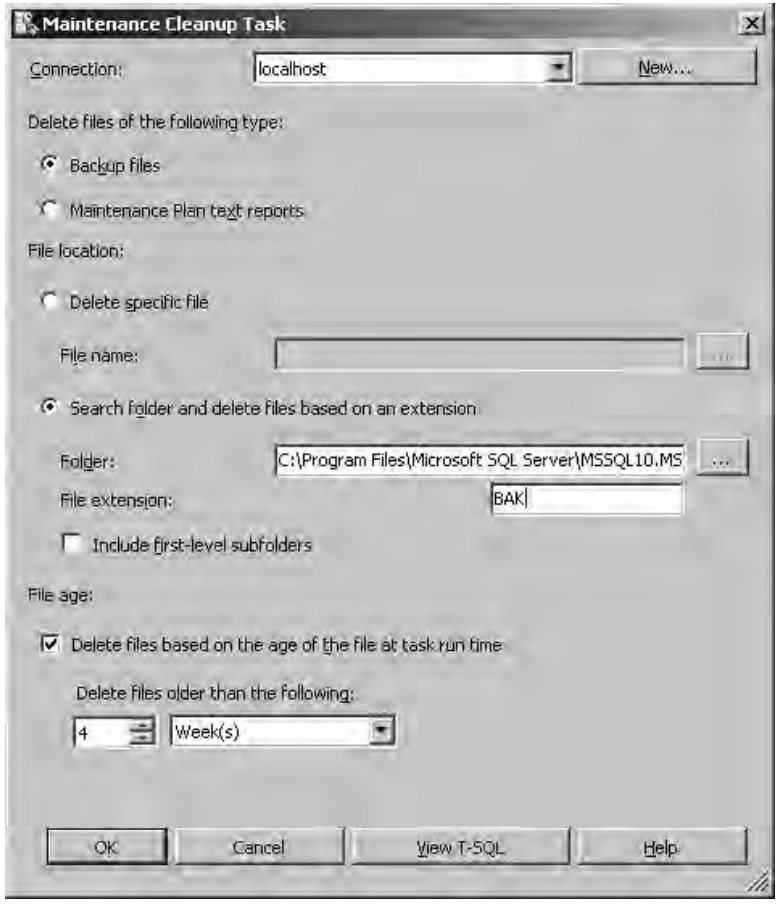


Figure 5-35 *Configuration options for the Maintenance Cleanup task*

Notify Operator Task

The Notify Operator task sends notification messages to SQL Server Agent operators. For this task to work properly, you need to have the SQL Server Agent service working, and for this task to show you the list of operators, you need to have operators defined in the SQL Server Agent and assign them an e-mail address. This task lists only users that have an e-mail address associated with them.

Following is the step-by-step method to configure this task:

1. Specify an ADO.NET (SqlClient Data Provider) Connection Manager in the Connection field. If you do not have any operator configured with an e-mail address to which you are trying to connect, you will receive the error message "There are no operators with e-mail addresses defined on this server." You can configure operators using SQL Server Management Studio.
2. Select the operators you want to notify from the list, as shown in Figure 5-36 under the Operators To Notify area.
3. Type in the subject and the message you want to send in the provided fields and click OK to complete the configuration.

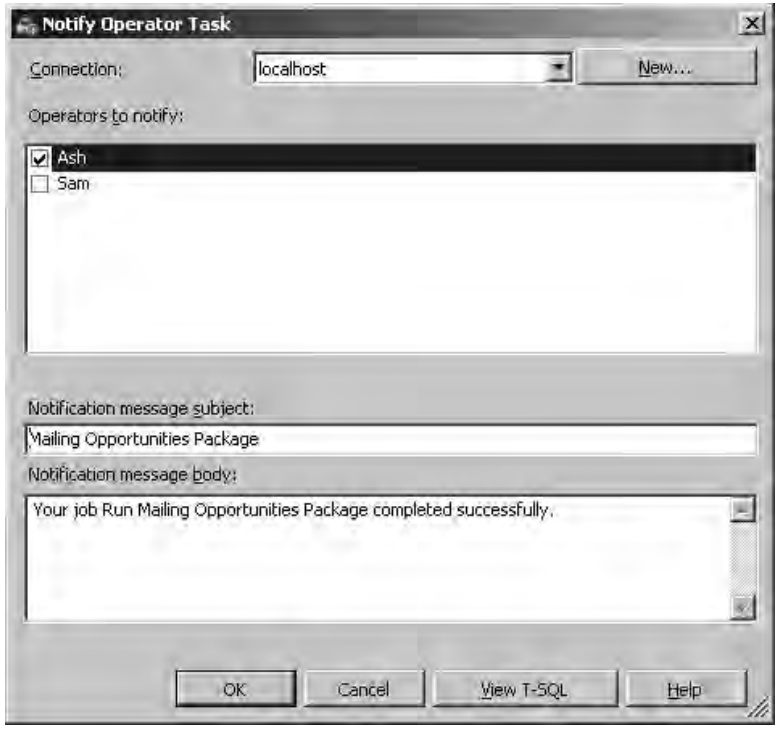


Figure 5-36 *User Interface for the Notify Operator task*

Rebuild Index Task

When you insert, update, or delete records from tables, the indexes become fragmented and the performance suffers. To keep the performance at the optimal level, you may decide to rebuild indexes as part of your SSIS package. The Rebuild Index task drops the index and creates a new one using the specified fill factor for SQL Server database tables and views. By rebuilding the index, this task removes fragmentation of pages, compacts the tables, reclaims disk space, and orders the rows in the contiguous pages, thus improving performance. This task effectively runs the `ALTER INDEX REBUILD` T-SQL command.

Here is the step-by-step method to configure this task:

1. Specify the ADO.NET Connection Manager in the Connection field.
2. Choose the databases from the following options: All Databases, System Databases (master, msdb, model), All User Databases (excluding master, msdb, model, and tempdb) or select databases from the provided list. If you choose a single database from the list, the Object option highlights, enabling you to choose tables or views for which you want to update statistics. However, if you choose multiple databases, you can't select the tables or views and the Object option is dimmed.
3. If you have selected a single database in the Databases option, you can select Table, View, or Tables and Views in the Object option (Figure 5-37). Selecting Table or View enables the Selection option.
4. When the Selection option is available, you can either select all objects or select objects from the list.
5. Choose how much free space you want to leave in a page. This free space is governed by the FILLFACTOR represented as a percentage. You can either use the original FILLFACTOR by selecting Reorganize Pages with the default amount of free space or you can type in a value for the free space as a percentage when you select the "Change free space per page percentage to" option. Based on your value of free space, a FILLFACTOR value is calculated. For example, if you specify 20% for the free space per page, the FILLFACTOR value will be calculated as 80%. The specified values of 0 and 100 are treated similarly, and the pages are filled to the full capacity.
6. When you build the index, the intermediate sort results are generally stored in the same database. However, you can also store these sort results in the tempdb. By doing this, you can reduce the time required to build an index, when the tempdb is on a separate set of disks than the database in question. Note that this may put extra requirements of space on the tempdb. So, when you want to use tempdb to store the intermediate sort results to reduce time, you can select the Sort results in tempdb advanced option.
7. Choose the "Keep index online while reindexing" advanced option to let the queries or updates happen to the underlying table during reindexing.

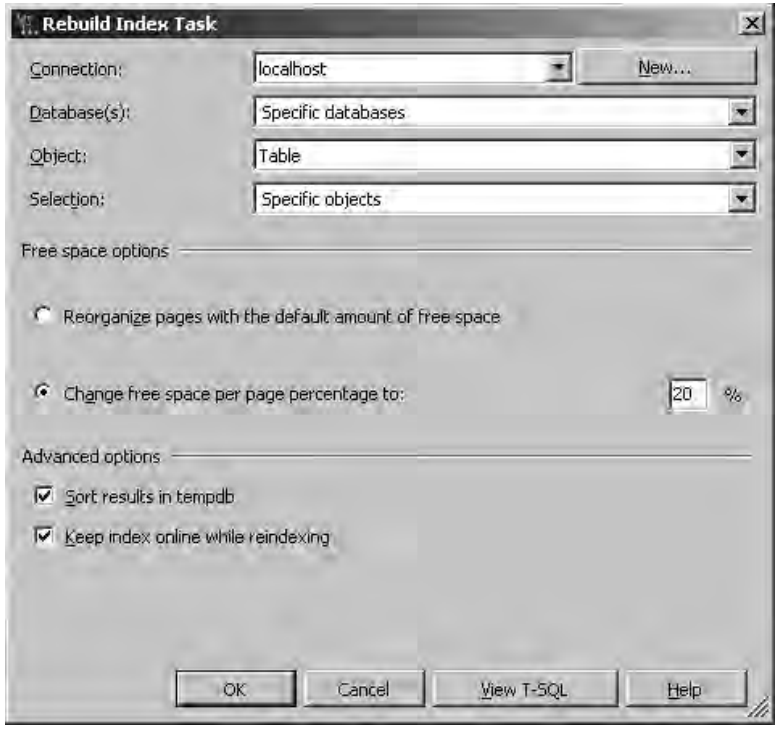


Figure 5-37 *Configuration options for the Rebuild Index task*

Reorganize Index Task

As mentioned earlier, the insert, update, or delete activities on the tables cause indexes to become fragmented and the performance suffers. To fix this, you can use this task in your SSIS packages; for example, after the Bulk Insert task you may use this task. The Reorganize Index task reorganizes indexes in SQL Server database tables and views and does the defragmenting of indexes by reordering leaf-level pages and compaction. This task runs the `ALTER INDEX` T-SQL command and uses the `REORGANIZE WITH (LOB_COMPACTION = ON)` clause in the command.

Here is the step-by-step method to configure this task:

1. Specify an ADO.NET Connection Manager in the Connection field.
2. Choose the databases from the option list (refer to Figure 5-38): All Databases, System Databases (master, msdb, model), All User Databases (excluding master, msdb, model, and tempdb), or select databases from the provided list.

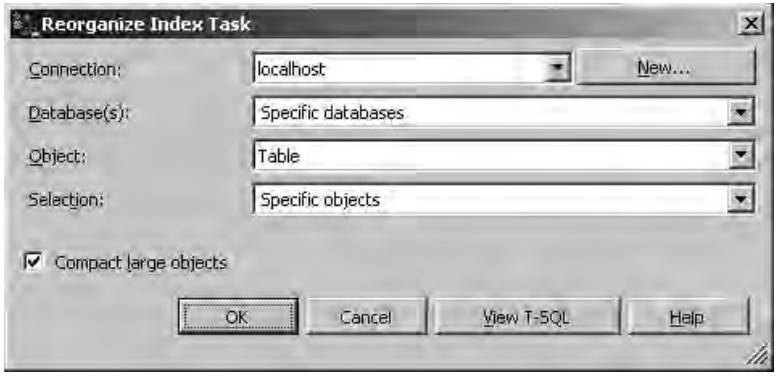


Figure 5-38 *Interface for the Reorganize Index task*

If you choose a single database from the list, the Object option highlights, enabling you to choose Table, View, or Tables and Views for which you want to update statistics. However, if you choose multiple databases in this option, you can't select the tables or views and the Object option is dimmed.

3. Selecting Table or View highlights the Selection option. When the Selection option is available, you can either select all objects or select objects from the list.
4. You can choose to compact the large objects. The data with the image, text, ntext, varchar(max), nvarchar(max), varbinary(max), or xml data type is considered as the Large object data. You can optionally choose to view the T-SQL code.

Shrink Database Task

This task runs the DBCC SHRINKDATABASE T-SQL command to shrink the specified database. The step-by-step configuration process goes like this:

1. Specify an ADO.NET Connection Manager to create a connection to the server.
2. Choose the databases from the option list: All Databases, System Databases (master, msdb, model), All User Databases (excluding master, msdb, model, and tempdb), or select databases from the provided list.
3. After selecting the database, specify the size in megabytes beyond what you want this task to execute to shrink the database.
4. Specify how much free space you want to leave in the database by typing in a value as a percentage (from 0 through 100) in the "Amount of free space to remain after shrink" option (see Figure 5-39). This percentage value is calculated on the space occupied by the data in the database and not the total size of the database. For example, when you have a database of 100MB with 40MB of data

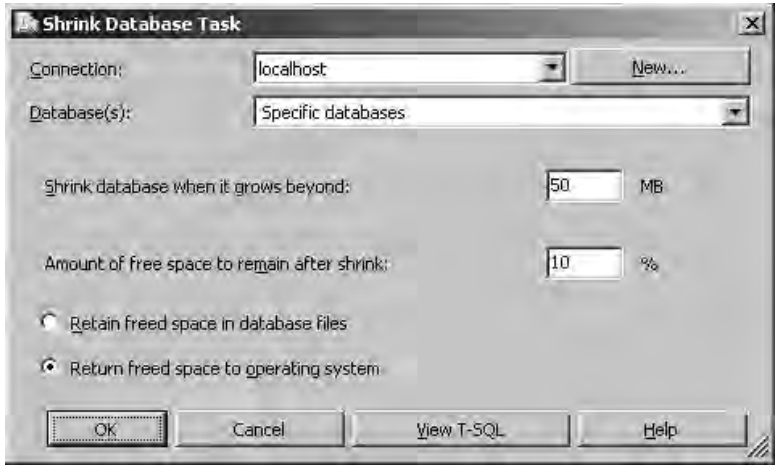


Figure 5-39 *Interface for the Shrink Database task*

and 60MB of free space, specifying a 50 percent value in the "Amount of free space to remain after shrink" option will shrink the database to 60MB, having 40MB of data and 20MB of free space.

5. Finally, choose between the two options either to retain the freed space in the database or to return the freed space to the operating system after shrinking.

Update Statistics Task

This task runs `UPDATE STATISTICS` T-SQL command to update information about the distribution of key values for one or more statistics groups (collections) in the specified databases. Following is the step-by-step procedure for configuring this task:

1. Specify an ADO.NET (SqlClient Data Provider) Connection Manager to connect to the server.
2. Choose one or more databases from the option list: All Databases, System Databases (master, msdb, model), All User Databases (excluding master, msdb, model, and tempdb), or select databases from the provided list.
3. If you choose a single database from the list, the Object option highlights, enabling you to choose Table, View, or Tables and Views for which you want to update statistics (see Figure 5-40). However, if you choose multiple databases in this option, you can't select the tables or views and the Object option is dimmed.
4. Selecting Table or View highlights the Selection option. When the Selection option is available, you can select all objects or select objects from the list.

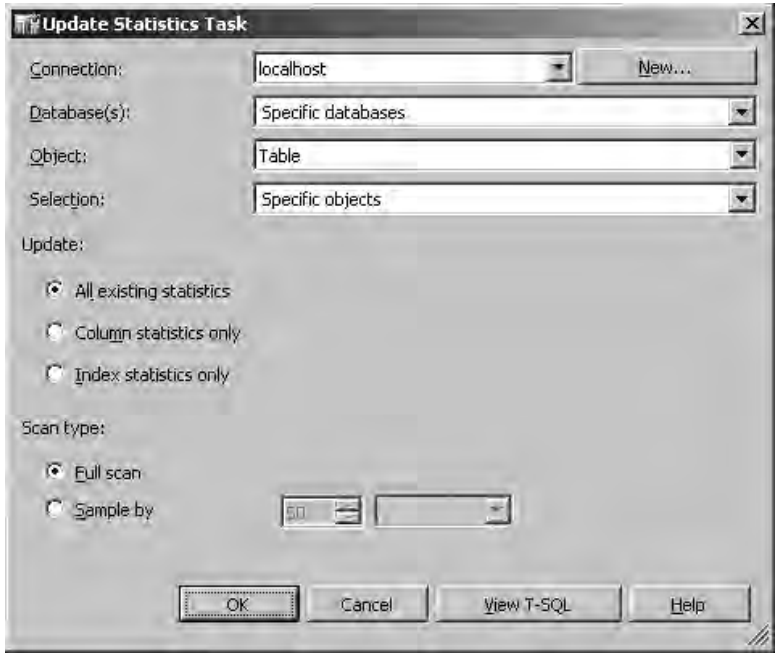


Figure 5-40 *The Update Statistics task interface*

5. Choose between the three options to update All Existing Statistics or Column Statistics Only or Index Statistics Only. If you view the T-SQL code, you will see that the Column Statistics Only option adds the WITH COLUMN clause and the Index Statistics Only option adds the `WITH INDEX` clause to the SQL code.
6. Finally choose a full scan or else specify a percentage or number of rows for the scan type.

Summary

We have covered a lot of ground in this chapter—all the Control Flow tasks, and we've used some of them in Hands-On exercises. We have also used variables and property expressions, along with a little exercise on precedence constraints. By now you must be getting ready to create packages involving complex features and data manipulation. Before we get into advanced features of event handling and logging (Chapter 8) and data flow (Chapters 9 and 10), we will cover administration of SSIS packages in next two chapters so that while you're learning about functional objects within SSIS, you also understand the storage and security architecture of Integration Services.

Chapter 6

Administering Integration Services

In This Chapter

- Connecting to Integration Services Service
- Managing SSIS Packages
- Running SSIS Packages
- Summary

When you're developing a package in Business Intelligence Development Studio (BIDS), the package is stored in the file system. Once the package has been developed, you can choose to save the developed package either on the file system or inside the MSDB database of SQL Server. Later you may use SQL Server Management Studio to run, monitor, or manage the packages from the chosen storage location. SQL Server Management Studio provides these management capabilities using a Windows service—the SQL Server Integration Services 10.0 service. Using this service, you can manage SSIS packages on local or remote servers. Integration Services service lets you stop and start SSIS packages, monitor running packages, import and export packages, and manage and customize the package store. SQL Server Management Studio provides a GUI interface to manage SSIS packages when you connect to Integration Services. Another utility, dtutil, helps you to manage SSIS packages from a command line. It is important to understand that the Integration Services service is a Windows service for managing SSIS packages and behaves in the same way as other Windows services do; for instance, you can use tools to connect to it and get your requests serviced such as to copy or run a package. SQL Server Management Studio is the out-of-the-box application that allows you to connect to the Integration Services service and manage the stored and running SSIS packages, although you can also use the dtutil utility or a custom-built application to do the same.

During installation, when you choose Integration Services in the Feature Selection page, SQL Server Integration Services 10.0 service is installed. If you don't install the Integration Services service, you cannot manage your packages using SQL Server Management Studio; however, you can still save and execute your packages while working in BIDS. Another benefit of having the Integration Services service installed is that it caches the SSIS tasks and components metadata when the package is loaded the first time and speeds up subsequent loading of the package.

Connecting to Integration Services Service

As mentioned earlier, you can connect to Integration Services using SQL Server Management Studio. When you install the Integration Services service, by default, it is started and is ready to be connected to. Generally, you'll be connecting to it with the default settings; however, you may have reasons to connect differently. You will be using Integration Services to manage your packages in the following scenarios:

- Managing packages with default settings
- Managing packages saved on a remote server
- Managing packages on an instance of SQL Server
- Connecting to Integration Services on a remote server

Managing Packages with Default Settings

Integration Services creates a configuration file, MsDtsSrvr.ini.xml, on installation. The default configuration file for Integration Services is located in the folder C:\Program Files\Microsoft SQL Server\100\DTS\Binn, though your location may differ, depending upon your installation. Locate the file, right-click it, and choose Open With | Notepad. You will see the following XML configuration code in the file:

```
<?xml version="1.0" encoding="utf-8"?>
<DtsServiceConfiguration xmlns:xsd="http://www.w3.org/2001/
XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <StopExecutingPackagesOnShutdown>true
  </StopExecutingPackagesOnShutdown>
  <TopLevelFolders>
  <Folder xsi:type="SqlServerFolder">
  <Name>MSDB</Name>
  <ServerName>.</ServerName>
  </Folder>
  <Folder xsi:type="FileSystemFolder">
  <Name>File System</Name>
  <StorePath>..\Packages</StorePath>
  </Folder>
  </TopLevelFolders>
</DtsServiceConfiguration>
```

When you connect to Integration Services using SQL Server Management Studio, you see the File System and MSDB root-level folders, which are explained in detail later in this chapter. For now, let's focus on connectivity; note that the ServerName element in the configuration file contains a dot (.), which is pointing to the default instance of the local SQL Server. This means by default, you connect to the MSDB database on the default instance of local SQL Server. The File System folder is configured by the FileSystemFolder folder type element, which points to C:\Program Files\Microsoft SQL Server\100\DTS\Packages folder by default.

Managing Packages Saved on a Remote Server

You might have saved packages to SQL Server on a remote server and want to manage those packages by connecting from your client machine. In this case, you will have SQL Server Management Studio running on your client machine and will connect to the local instance of Integration Services. However, the local instance is configured to manage the storage areas such as the MSDB database that are on the remote server to show packages saved to SQL Server and a UNC path for packages saved to File System. The configuration file is modified to accomplish this.

For the MSDB database of the remote SQL Server, you'll modify the ServerName element as shown:

```
<ServerName>RemoteSQLServerName</ServerName>
```

And for the File System folder on a remote server, you'll modify the StorePath element as shown:

```
<StorePath>\\RemoteFileServerName\ShareName</StorePath>
```

You will have to restart the SQL Server Integration Services 10.0 service for changes to take effect. After restart, when you connect to the local instance of Integration Services using SQL Server Management Studio, the MSDB folder in Stored Packages will show the packages saved to the remote SQL Server and the File System folder will show the packages saved to the file share on the remote server.

Managing Packages on an Instance of an SQL Server

One important point is to understand that Integration Services is not a multi-instance application like SQL Server. However, it is aware of SQL Server instances and can connect to them. For example, you might prefer to save your packages in the instance of SQL Server to which they relate instead of saving them into default instance. In that case you can modify the ServerName element as follows and connect to the MSDB database of the specified instance of SQL Server.

```
<ServerName>ServerName\InstanceName</ServerName>
```

You will have to restart the SQL Server Integration Services 10.0 service for changes to take effect. After restart, when you connect to Integration Services using SQL Server Management Studio, the MSDB folder in Stored Packages will show the packages saved to the specified instance.

Connecting to Integration Services on a Remote Server

Sometimes, you may also want to connect to Integration Services on a remote server; for instance, you may be using SQL Server Management Studio on your client machine but connecting to remote Integration Services. You can do so when you choose Integration Services in the Server Type field and the remote server name in the Server Name field in the Connect To Server dialog box, which pops up when you start SQL Server Management Studio or when you click the Connect button in the Object Explorer and choose Integration Services from the options list. Remember, Integration Services is a Windows service and uses Windows authentication; that is, it is logged on to the user's

account to connect to a remote server. If your account has permissions to connect to the remote Integration Services server, you will connect without any issues. However, you may receive an "Access Is denied" error if you do not have sufficient rights. To allow remote users connections, users need to have permissions on DCOM objects and the Integration Services service needs to be configured for the DCOM permissions. As you can guess, administrators will have access by default; however, if you're using restrictive security on your server (that is, if you want to give users just enough rights to let them perform their jobs), you need to assign them appropriate permissions. First, you will need to add the required users into the Distributed COM Users group. You also need to assign users Remote Activation rights in the Launch and Activation Permissions section under the Security tab of MsDtsServer100 application properties. You can access MsDtsServer100 application under the DCOM Config node of Component Services management tool. For more details on how to configure rights for users, search for the article "Connecting to a Remote Integration Services Server" on MSDN.

Managing SSIS Packages

SQL Server Management Studio enables you to connect to Integration Services and manage your SSIS packages using a graphical interface. Integration Services allows you to save your packages either on the file system or in the MSDB database and can manage the storage and running of your packages from this single interface. You can import packages from any storage area and export into another. You can also run your packages from here and watch their progress while the packages are executing.

Let's do a quick Hands-On exercise to understand the storage options provided by the Integration Services service.

Hands-On: Working with Integration Services Storage Folders

In this exercise, you will learn about the storage areas where you can save packages and how you can change their storage type and location.

Method

The following will be completed:

- Connect to the Integration Services service and understand the storage areas
- Save a package to SSIS Package Store
- Add a root-level folder to Stored Packages
- Import and export packages

Exercise (Connect to Integration Services Service and Understand the Storage Areas)

In the first part of this Hands-on, you will use SQL Server Management Studio to connect to Integration Services service and understand the folder structure it provides.

1. Run SQL Server Management Studio. Choose Integration Services in the Server Type field of the Connect To Server dialog box and type in the name of your server in the Server Name field. Then click Connect.
2. You should see the two top-level folders—Running Packages and Stored Packages. The Running Packages folder lists the currently running packages. You will be monitoring running packages from this place. Expand the Running Packages folder. You will realize that it doesn't have any subfolder and you will not be able to create a subfolder under it. This is because the Running Packages folder is not extensible.
3. Expand the Stored Packages folder. Note that it has two subfolders—File System and MSDB. The packages saved on to the file system appear under File System folder, and the packages saved inside the SQL Server MSDB database appear under the MSDB folder. You can create subfolders under any of these folders, as this area is configurable and extensible. You will be working primarily with this area to manage the storage of your folders.
4. The File System and MSDB folders are called *root-level folders.* The File System folder lists the packages stored in the file system. The default location of this folder is %Program Files%\Microsoft SQL Server\100\DTS\Packages. Right-click the File System folder and choose New Folder from the context menu. Type **Campaigns** in the Create New Folder window and click OK to close the dialog box. You will see the Campaigns subfolder under the File System folder, you might have to refresh if this is not visible. Now explore to the %Program Files%\ Microsoft SQL Server\100\DTS\Packages folder and you will see Campaigns folder there.

   You can also create subfolders in the MSDB folder. Packages saved to the SQL Server MSDB database appear under the MSDB folder in Integration Services. To sum up, you can create subfolders under the File System folder and the MSDB folder to manage your packages in a hierarchical order. You can then delete or rename these subfolders; however, you cannot delete or rename the root folders.

Exercise (Save a Package to an SSIS Package Store)

In this part, you will explore three package storage areas the Integration Services service provides, and will save a package into SSIS Package Store area.

5. Start BIDS and open the Contacting Opportunities solution. When the solution loads, open the Mailing Opportunities.dtsx package under the SSIS Packages node to open the package in the Designer. When the package appears on screen, choose File | Save Copy Of Mailing Opportunities.dtsx As. This will open the

Save Copy Of Package dialog box, which gives you choices of where you want to save your package. Click the down arrow in the Package location field to see the locations where you can save your packages. You have three options:

- SQL Server
- File System
- SSIS Package Store

Choosing SQL Server lets you save your packages to the sysssispackages table of the MSDB database in SQL Server. The packages saved using this option will appear under the MSDB folder when you connect to Integration Services using SSMS.

The File System option allows you to save your package anywhere on the file system. You can specify a path in the Package Path field for saving the package. You will be using this option if you do not want to save your packages in the Integration Services' defined storage area, the %Program Files%\Microsoft SQL Server\100\DTS\Packages folder. The packages saved using this option will not be available for management under SQL Server Management Studio. There are times or instances when you would like to choose this option; for instance, you may want to save your packages on a central file server instead of the local server or you may not want your packages be made available in Integration Services storage folders.

The third option of SSIS Package Store allows you to save your package in the Integration Services–defined storage area on the file system or in the MSDB database. When you select this option and click the ellipsis button to specify a Package Path, you are taken to the *root-level folders* inside the Integration Services Stored Packages area and the File System and MSDB folders are shown to you. It's worth noting the difference between the File System option provided to you in the Package Location field earlier and the current File System folder shown inside the SSIS Package Store. The earlier option of File System allows you to save your package anywhere on the file system and those packages will not be visible in SQL Server Management Studio, while selecting the File System folder in the SSIS Package Store option lets you save your package only under the Stored Packages root folder defined in Integration Services, which by default is %Program Files%\Microsoft SQL Server\100\DTS\Packages, and the packages saved here will be available for management in SQL Server Management Studio. The defined storage area on the file system is configurable and you can change the location of this folder; however, it is worth noting the difference between these two options so that you don't confuse yourself while saving the packages.

6. Select SSIS Package Store in the Package location field and type in your server name or **localhost** in the Server field. Click the ellipsis button opposite to the Package Path field to open the SSIS Package dialog box. Expand the File System folder, select Campaigns, and then type **Mailing Opportunities** in the Package Name field. Click OK to return (see Figure 6-1). Note the path specified in the Package path field. Click OK again to save the package.

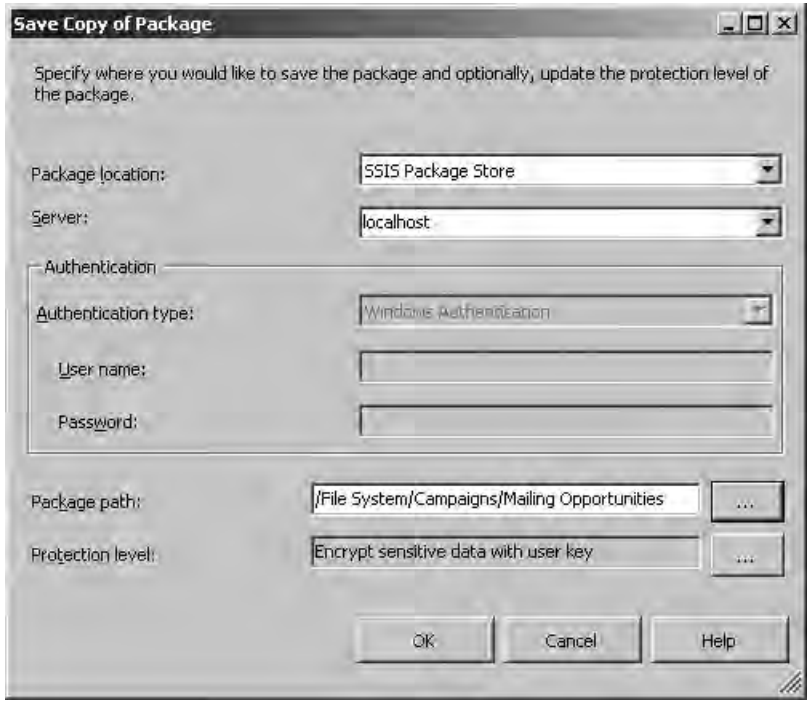


Figure 6-1 *Saving your package to the SSIS Package Store*

7. Switch to SQL Server Management Studio, right-click the Campaigns folder, and choose Refresh to see the Mailing Opportunities package.

   Also, if you explore to the Campaigns folder under the %Program Files%\Microsoft SQL Server\100\DTS\Packages folder, you will see the Mailing Opportunities.dtsx file saved there as well. This is the default location specified in the Integration Services configuration file, which is discussed in the next part.

Exercise (Add a Root-Level Folder to Stored Packages)

Now you know that the root-level folders can contain subfolders. But what if you want to store your packages at different location than the default location? Integration Services provides you with a configuration file to configure the storage locations.

8. Open the MsDtsSrvr.ini.xml file with Notepad to see the XML configuration code. This code has been listed earlier, in the section "Managing Packages with Default Settings." Locate the XML element `<StopExecutingPackagesOnShutdown>` and note that you can control how Integration Services treats the running packages when the service is stopped. By default, it is set to True, which means that the running packages will be stopped when the SSIS service is stopped. By changing

True to False in this line, you can configure Integration Services to continue running packages even if the SSIS service has been stopped.

9. We will now add a root file system folder so that we can save our packages to a different area. Move to the XML element `<Folder xsi:type="FileSystemFolder">` in the configuration file. Note that the storage path ..\Packages actually points to the %Program Files%\Microsoft SQL Server\100\DTS\Packages folder. Copy the `<Folder>` element—i.e., copy four lines starting from `<Folder xsi:type="FileSystemFolder">` to `</Folder>`. Now add these lines between the `</Folder>` and `</TopLevelFolders>` elements. In the new lines, change the values of the `<Name>` element to **My SSIS Packages** and `<StorePath>` element to **C:\SSIS\Packages**. Your new lines should look like this:

```
<Folder xsi:type="FileSystemFolder">
<Name>My SSIS Packages</Name>
<StorePath>C:\SSIS\Packages</StorePath>
</Folder>
```

Save the configuration file. To see the changes, you have to restart the Integration Services service. Start the SQL Server Configuration Manager and click the SQL Server Services node in the left-hand pane. Locate the SQL Server Integration Services 10.0 service from the list, right-click, and choose Restart. Once the service stops and starts again, switch to SQL Server Management Studio and refresh Stored Packages to see the newly added root level folder, My SSIS Packages. This folder points to C:\SSIS\Packages folder on the file system.

Exercise (Import and Export Packages)

You can change the storage location of packages by using the import and export features of the Integration Services. This can also be done from the command prompt using the dtutil utility. In this exercise, you will import the Mailing Opportunities package to the MSDB folder and then export this package to the My SSIS Packages store you created previously.

10. Switch to SQL Server Management Studio, right-click the MSDB folder, and choose Import Package. An Import Package window will open. This window is similar to the Save Copy Of Package dialog box, except it has an extra field for specifying the package name to be imported. In the Package Location field drop-down list, choose SSIS Package Store and type **localhost** in the Server field. Click the ellipsis button opposite to the Package Path field. In the SSIS Package dialog box, expand the File System folder, then the Campaigns folder, and then select the Mailing Opportunities package (see Figure 6-2). Click OK to return.
11. Verify that the Package name field contains the name Mailing Opportunities. Click OK to start importing the package. Refresh the MSDB folder to see that the Mailing Opportunities package has been imported.

Figure 6-2 *Selecting a package to import*

Connect to the local database server using the Connect button in the Object Explorer window. Open a new query pane and run the following query against the MSDB database:

```
SELECT * FROM msdb.dbo.sysssispackages WHERE name like 'Mailing%'
```

The result set will contain all the information about the Mailing Opportunities package.

12. Let's export our package to an area on the file system, which is not defined in the SSIS Package Store. Go to Integration Services in the Object Explorer of SQL Server Management Studio, right-click the package Mailing Opportunities stored under the MSDB folder, and choose Export Package from the list of options.
13. An Export Package\MSDB\Mailing Opportunities dialog box opens. In the Package location field, choose SSIS Package Store and type **localhost** in the Server field. These settings enable Integration Services to read storage locations from the MsDtsSrvr.ini.xml file on the local server.
14. Click the ellipsis button next to the Package Path field to open the SSIS Package dialog box. Here you can see the My SSIS Packages folder you created earlier in Integration Services. Select My SSIS Packages and verify that the Package Name field already has Mailing Opportunities filled in. Click OK to close this dialog box. You will see the /My SSIS Packages/Mailing Opportunities value specified in the Package Path field. This is a user-friendly option that allows you to create a storage location pointer in Integration Services; all users can save packages without bothering about the actual storage location of the package. This is also quite helpful in case you want to keep your packages on a central server for archiving on your

network storage. You also get an opportunity to define a new name to your package here, which is helpful in case you are saving with version numbers attached in the end of the package name. Click OK to export the package. Explore to the C:\SSIS\Packages folder to verify that the package has been saved there. Leave the SQL Server Management Studio running (if you're not ready for a break yet!), as we will return to it in the next exercise.

Review

In this exercise, you worked with root folders of Integration Services and managed to create a subfolder to the root folder, save a package to it, and view the options available for saving a copy of the package. You also learned about where you can monitor the running packages and where you can save them. After that, you used the configuration file for Integration Services to create an additional root folder, My SSIS Packages. The path specified for the My SSIS Packages folder was on the local server hard disk in this exercise, but it is not limited to that. You can actually use a UNC network share (such as \\<computername>\sharename) for the folder path and point your SSIS packages store to a central network share. You also saw how to use import and export features to change the storage type and location of the packages.

While we are discussing the configuration file options, you might find a couple of other options useful. If you want to use an SQL Server named instance on the same server or on the remote server for storing your Integration Services packages, you can do so by specifying the instance name in the <ServerName> element of configuration file. You can specify the named instance in the format *InstanceName\ServerName.* Save the file, restart the Integration Services service, and you are ready to store your packages in the specified SQL Server named instance.

The second option is to use an alternative configuration file. Integration Services does allow you to set up and use an alternative configuration file. By default, Integration Services refers to the MsDtsSrvr.ini.xml configuration file located at C:\Program Files\ Microsoft SQL Server\100\DTS\Binn\ MsDtsSrvr.ini.xml. This location is mentioned in the Windows registry. You can change the location of the default configuration file by modifying the value of the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\100\SSIS\ServiceConfigFile. After making this change, all you need to do is restart the Integration Services service, which will then read the new file from the location you specified in the registry.

dtutil Utility

You have worked with Integration Services from within SQL Server Management Studio and have imported and exported your SSIS packages with the help of GUI tools. Having used these tools, you have realized how easy it is to manage your

Integration Services packages from the tools' GUI. However, sometimes you need a command-line tool to write scripts to do the work in batches, you may want to automate tasks, or perhaps you want to schedule jobs. In this section you will learn to use dtutil, a command prompt utility provided for managing Integration Services packages.

The dtutil utility can be used for the following:

- Verifying the existence of an SSIS package
- Copying SSIS packages
- Moving SSIS packages
- Deleting SSIS packages
- Signing SSIS packages

You can use dtutil to perform any of these operations on any type of storage—i.e., the MSDB database in SQL Server, anywhere on the file system, and the defined SSIS Package Store.

Here is the base syntax:

```
dtutil /option [value] [/option [value]]...
```

The parameters can be in any order and are not case-sensitive. Also, they are defined in terms of *option and value pairs.* The options must begin with a slash (/) or a minus sign (–). The dtutil uses different options for passing user credential details of the source and destination servers. For the source server, dtutil uses these options:

- `SourceS` For the source server
- `SourceU` For the user name at the source server
- `SourceP` For the password of this user

And for the destination server, dtutil uses the following options:

- `DestS` For the destination server
- `DestU` For the user name at the destination server
- `DestP` For the password of this user

You may see a pipe (|) at some places in the syntax, which is used as an OR operator to show the possible values of the arguments. Finally, it is easier to learn the dtutil commands by practicing them than by trying to memorize the syntax. So let's get started with the Hands-On exercises.

Hands-On: Using dtutil

In this Hands-On exercise, we will use dtutil against various scenarios to understand how it works and learn the syntax along the way.

Method

Start SQL Server Management Studio and connect to Integration Services. Also, open Windows Explorer and the command prompt and be ready to switch windows to see the results.

Exercise (Verify the Existence of an SSIS Package)

You can check the existence of a package using the EXISTS option of dtutil:

```
dtutil /EXISTS [/option [value]]...
```

1. Verify whether the Mailing Opportunities package exists in the MSDB database of the local server instance:

   ```
   dtutil /EXISTS /SQL "Mailing Opportunities"
   ```

   As we know that the package exists in the local instance, you will see the "The specified package exists" message. The SQL option checks for packages stored in the MSDB database of the SQL Server. The other options that can be used instead of SQL are FILE and DTS. The FILE option allows the dtutil to check the existence of a package on the specified file system folder, whereas DTS checks through the defined SSIS Package Store. Note that the SQL option cannot be specified with the DTS or FILE option, as they are mutually exclusive. However, you can use the User, Password, or Server options to extend the SQL option usage.
2. Verify that the Mailing Opportunities package exists in the C:\SSIS\Packages folder of the local server:

   ```
   dtutil /EXISTS /FILE "C:\SSIS\Packages\Mailing Opportunities.dtsx"
   ```

   The `FILE` option checks the specified file system folder for the existence of the package. Note that `FILE` option cannot be specified with the `DTS`, `SQL`, `User`, `Password`, or `Server` options. Also, note the way you specify the name of an SSIS package—with the FILE option you have to specify the .dtsx extension with the package name, while this is not to be included for the other two SQL and DTS options.
3. Verify that the Mailing Opportunities package exists in the shared folder called Data, of the remote server named AIT:

   ```
   dtutil /EXISTS /FILE "\\AIT\Data\Mailing Opportunities.dtsx"
   ```

In this example, we used a UNC path to specify the location of the package. *AIT* is the name of the second server I'm using to test Integration Services packages designed for this book.

4. Verify that the Mailing Opportunities package exists in the My SSIS Packages store:

```
dtutil /EXISTS /DTS "\My SSIS Packages\Mailing Opportunities"
```

The DTS option specifies that the verification for existence of the package should be done on the SSIS Package Store. Note that the DTS option cannot be specified with the FILE, SQL, Server, User, or Password options.

Exercise (Copy SSIS Packages)

If you have used the COPY command of good old DOS (this is still available at the command prompt, but hardly anyone uses it now), you will find copying packages quite the same. Using the DOS COPY command, you provide the source path and name of the file to be copied and destination path and name of the file to be copied to. With dtutil, you provide similar parameters but in a slightly more complex manner because multiple storage locations (SQL, DTS, and FILE) are involved here. The syntax for the COPY command is shown here:

```
dtutil /{SourceLocation} [SourcePathandPackageName] /COPY
{DestinationLocation};[DestinationPathandPackageName] ...
```

Note that you do not use a backslash (/) when you provide the destination location after the COPY command.

5. You have saved your package in the file system while designing the package using BIDS. Now you want to copy your package from the file system to the MSDB database of the SQL Server (named W8KR2 in my case).

```
dtutil /FILE "C:\SSIS\Packages\Mailing Opportunities.dtsx" /COPY
SQL;"Mailing Opportunities" /DestS W8KR2
```

If you still have the Mailing Opportunities package saved to the MSDB database of your server, you will see a confirmation prompt, "Package 'Mailing Opportunities' already exists. Are you sure you want to overwrite it?" Typing **Y** will overwrite the package and typing **N** will cancel the command. To stop such confirmation prompts, you can use the /Quiet option in the command while trying to copy, move, or sign the package. The option DestS specifies the destination SQL Server. If this option is missing in the dtutil command, the local SQL Server will be used.

Try the same command by changing the destination package name to **Mailing Opportunities01**. When you see the message "The operation completed successfully," switch to SQL Server Management Studio; refresh the MSDB folder under Stored Packages in Integration Services to see this new package there.

6. You have a package in the MSDB database and want to copy it to the file system on a different server:

```
dtutil /SQL "Mailing Opportunities" /SourceS W8KR2 /COPY
FILE;"\\AIT\Data\Mailing Opportunities.dtsx"
```

By now, you must have noted that the double quotation marks have been used on the package path and name. This is because the package has a white space in its name; you don't need to use double quotes if your package doesn't have white space in its path or name. The `SourceS` option specifies the source SQL Server name. Again to clarify, *W8KR2* and *AIT* are the names of servers I use in my little test lab. You should replace the names with your computer names to perform this test. Once the command has been run successfully, check the destination location to see that the package has been copied over.

Exercise (Move SSIS Packages)

The MOVE option moves an SSIS package from one storage location to another. The syntax for this option is quite similar to the syntax of the COPY option.

```
dtutil /{SourceLocation} [SourcePathandPackageName] /MOVE
{DestinationLocation};[DestinationPathandPackageName] ...
```

7. You have a package in your My SSIS Packages store, which you want to archive—i.e., you want to remove it from the SSIS Package Store and move it to the central archival location.

```
dtutil /DTS "\My SSIS Packages\Mailing Opportunities" /MOVE
FILE;"\\AIT\Data\Mailing Opportunities.dtsx"
```

You will see a prompt to confirm that you want to write the existing package. Type **Y** to overwrite. Once the package is moved successfully, switch over to SQL Server Management Studio and refresh the My SSIS Packages folder to see that the package has been removed from there. Check the destination folder to find the package exists there.

8. Another example of moving a package could be from the default SQL Server instance to the named SQL Server instance. The TRAINING is the named instance that has been used with user name and password in the following command:

```
dtutil /SQL "Mailing Opportunities" /MOVE SQL;"Mailing
Opportunities"
/DestU admin /DestP Tra1n1n9 /DestS SARTH\TRAINING
```

Exercise (Delete SSIS Packages)

You can delete a package using the delete (DEL) option. The generic syntax for using the option is shown here:

```
dtutil /DEL /{SQL | DTS} [DestinationPathandPackageName]...
```

9. You want to delete an existing package from a shared folder on the network:

   ```
   dtutil /DEL /FILE "\\AIT\Data\Mailing Opportunities.dtsx"
   ```

   When you see the message "The operation completed successfully," switch over to Windows Explorer and check that the package has been deleted from the specified folder.

10. You want to delete a package from the MSDB database in SQL Server:

    ```
    dtutil /DEL /SQL "Mailing Opportunities01"
    ```

    On successful deletion, switch over to SQL Server Management Studio and go to the MSDB folder; then refresh it to find that the package is no longer there. This option uses Windows Authentication, which is more secure. It is recommended that you use Windows Authentication whenever possible, though you can also use SQL logins to operate dtutil by specifying an SQL user name and password for the SQL Server.

Exercise (Sign SSIS Packages)

When you are working on developing a complex SSIS project consisting of several packages, you may want to deploy the packages when they have achieved a certain level of functionality, while the development team continues to further enhance the package functionalities. In such scenarios, if you are concerned that your packages can be changed and run accidentally by other developers working on the same project, you can use the SIGN option to sign the packages and prevent these changed packages from being loading and running.

```
dtutil /{SourceLocation} [SourcePathandPackageName] /Si[gn]
{DestinationLocation};[DestinationPathandPackageName];Hash
```

The `SIGN` option uses three arguments separated by semicolons. An SQL destination can include the `DESTU`, `DESTP`, and `DESTS` options. The Path argument specifies the location of the package on which to take action and the Hash argument specifies a certificate identifier expressed as a hexadecimal string of varying length.

11. For protecting your package saved in the file system, use the following command:

    ```
    dtutil /FILE "C:\SSIS\Packages\mailing Opportunities.dtsx" /SIGN
    FILE;C:\SSIS\Packages\SignedPackage.dtsx;
    7B18F301A198B83778B5E546729B0539A0D4E758
    ```

The hash value actually corresponds to the digital certificate I've installed on my computer, which you should change accordingly when you use this command.

Exercise (Encrypt SSIS Packages)

Properties containing information such as a password or connection string can be treated as a sensitive data. Properties containing sensitive data can be protected using encryption. Integration Services packages have a property called *ProtectionLevel* that you can use to set the level of protection for your packages. If you are creating a custom task, you can specify that the properties be treated as sensitive. Following are the ProtectionLevel options available:

- Level 0 strips the sensitive information.
- Level 1 encrypts sensitive data using the local user credentials.
- Level 2 encrypts sensitive data using the required password.
- Level 3 encrypts the package using the required password.
- Level 4 encrypts the package using the local user credentials.
- Level 5 encrypts the package using SQL Server storage encryption.

The generic syntax for encryption option is shown here:

```
dtutil /{SourceLocation} [SourcePathandPackageName] /En[crypt]
{DestinationLocation};[DestinationPathandPackageName];ProtectionLevel;Password
```

12. Let's encrypt a package stored in the file system with a password using this command:

    ```
    dtutil /FILE "C:\SSIS\packages\mailing opportunities.dtsx" /ENCRYPT File;
    "C:\ssis\packages\Encrypted mailing opportunities.dtsx";3;abcd
    ```

    Now, if you open this newly created package Encrypted mailing opportunities.dtsx after adding it to a project in BIDS, you will have to specify the encryption password *abcd* while adding and loading the package. You may be wondering by now how one can decrypt the encrypted packages. Yes, you are right; dtutil has a `Decrypt` option too. Using this option lets you set the decryption password when loading a package with password encryption.

Exercise (List Contents of Folders)

You may want to check the contents of folders and subfolders in My SSIS Packages store and SQL Server (MSDB database) storage area. `FDi` is a short form of asking dtutil to show a folder directory:

```
dtutil /FDi {SQL | DTS}; FolderPath [;S]
```

You can use the S argument with this command if you want to see the contents of subfolders.

13. Here's how you check the contents in SSIS Packages Store area:

    ```
    dtutil /FDi DTS
    ```

    When you use this command, the contents of the root folder will be returned, showing you the File System, MSDB, and My SSIS Packages folders.
14. To see the contents of My SSIS Packages folder, type this:

    ```
    dtutil /FDi DTS;"My SSIS Packages"
    ```

Exercise (Create a Folder)

You can also create subfolders in the Integration Services storage area or the MSDB database for storing packages in a hierarchical manner.

```
dtutil /FC[reate] {SQL | DTS};ParentFolderPath;NewFolderName
```

where *ParentFolderPath* is the location for the new folder and *NewFolderName* is the name of the new folder.

15. Create a folder in the MSDB database:

    ```
    dtutil /FC SQL;\;MyPackages
    ```

    When you see the message "The operation completed successfully," run the following command:

    ```
    dtutil /FDi SQL
    ```

    You will see the MyPackages folder created in the MSDB database. You can also verify the existence of a folder as we did for packages. To verify the MyPackages folder, type this:

    ```
    dtutil /FE SQL;MyPackages
    ```

    The FE command option stands for Folder Exists.

Exercise (Rename a Folder)

The folders created in the Integration Services area and the MSDB database can be renamed using the FR option in dtutil.

```
dtutil /FR[ename] {SQL | DTS};ParentFolderPath;OldFolderName;NewFolderName
```

where *ParentFolderPath* is the location of the folder to rename, *OldFolderName* is the current name of the folder, and the *NewFolderName* is the new name of the folder.

16. Rename the MyPackages folder.

    ```
    dtutil /FR SQL;\;MyPackages;Archives
    ```

Exercise (Delete a Folder)

Using the FDe command option, you can delete a folder existing in Integration Services, the storage area, or the MSDB database of the SQL Server.

```
dtutil /FDe[lete] {SQL | DTS};ParentFolderPath;FolderName
```

where *ParentFolderPath* is the location of the folder to delete and *FolderName* is the name of the folder to delete.

17. Delete the Archives folder from the MSDB database:

    ```
    dtutil /FDe SQL;\;Archives
    ```

    Once the operation is completed, switch over to SQL Server Management Studio to verify the deletion of the folder from the MSDB folder under the Stored Packages folder.
18. While working on big projects, you will create a package with some basic configurations, connection managers, auditing tasks, variables, and such configured in it. You will then use this package as a template package whenever you're starting to develop a new package. This approach will save lot of time and will apply the same development standards across all the packages. However, this approach requires you to recreate the package ID. All the packages get a GUID created automatically at the time of package creation that you can also change any time and create a new ID. This ID is important, as many applications and utilities use them to identify and run packages. If two packages have same ID, obviously, the applications or utilities will get confused and undesirable results may occur. You can generate a new ID at any time during design time by going to the ID property and selecting the <Generate New ID> option. You can also generate a new ID even after the package has been saved and avoid reopening a package; actually you can generate IDs for multiple packages in a folder when you use the dtutil

command in a batch. Use the following command to generate a new GUID, and verify it by opening the package in BIDS before and after the command.

```
dtutil /I /FILE "C:\SSIS\Packages\mailing Opportunities.dtsx"
```

Review

In this exercise, you learned how to use the dtutil utility to manage your packages and the storage areas. You used it to copy, move, delete, sign, and encrypt packages stored in various storage areas. You also used the dtutil utility to see the contents of folders; to create, rename, and delete a folder from various storage areas; and to verify the existence of a folder or a package in a particular storage area.

Running SSIS Packages

You can run an Integration Services package using the following applications or utilities:

- SQL Server Import and Export Wizard
- BIDS
- Execute Package Utility (DTExecUI)
- DTExec Utility
- SQL Server Agent
- Programmatically

Your choice of tool or utility can be guided by various factors as discussed next.

SQL Server Import and Export Wizard

The SQL Server Import and Export Wizard is the easiest utility to work with. Its interactive GUI provides a simple interface to build and run Integration Services packages. You have already used this tool in variety of scenarios to create and run packages. For more details on the SQL Server Import and Export Wizard, refer to Chapter 2.

BIDS

You can run your packages from within BIDS while you are developing, debugging, and testing your packages. You can run your package by clicking the Start Debugging button on the toolbar, by pressing F5, or by right-clicking the package in Solution Explorer and choosing the Execute Package option from the context menu. You have

already used these options while developing packages in the last chapter. BIDS uses an execution host DtsDebugHost that will start whenever you run a package inside BIDS. You can see this process in the Task Manager while running a package. The DtsDebugHost.exe file exists in the Binn folder under DTS and has both 32-bit and 64-bit versions. This is the main reason BIDS can support both 32-bit and 64-bit execution environments despite being itself a 32-bit application. You can control the execution environment by using the Run64BitRuntime debugging configuration in Project properties. The Run64BitRuntime is by default set to True, which means the package will execute using the 64-bit version of DtsDebugHost if the 64-bit SSIS run time is installed; otherwise, this setting is ignored. BIDS attaches to this process at run time to report back the debugging information. This reporting is additional overhead on execution, and hence using BIDS is the slowest method to run a package. Refer to Allan Mitchell's blog for "Comparing Overhead on the Execution Methods" on SQLIS.com to compare performances of different methods.

Also, when you run a package in BIDS, a Progress tab appears to show how the package is doing while the package is running, which finally converts into an Execution Results tab when you switch to design mode or stop debugging on package completion. You do not need Integration Services to design and run packages inside BIDS; however, the packages run in this fashion in SSIS Designer run immediately and cannot be scheduled. If you want to run your package outside BIDS, you must have Integration Services installed on the local computer. If your solution contains multiple projects and packages and you want to run a particular package before any other package each time you execute the project, you will have to mark that package as a startup package. To mark a package as a startup package, right-click the project in the Solution Explorer window and choose Properties. Go to the Debugging page and select the package name in the StartObjectID field in the Start Action section.

Execute Package Utility (DTExecUI)

The Execute Package Utility (DTExecUI) is a GUI-based version of the DTExec command prompt utility covered in the following section. To run a package using DTExecUI, connect to Integration Services in SQL Server Management Studio, right-click the package, and select the Run Package option from the context menu or simply double-click the package file to invoke the DTExecUI. Alternatively, you can first invoke the utility by choosing Start | Run | DTExecUI and then select the package to run from the File System, MSDB, or SSIS Package Store root folder of the Integration Services storage area.

Hands-On: Running an SSIS Package Using the Execute Package Utility

In this exercise, you will learn how to use the Package Execution utility to run SSIS packages.

Method

You can start this utility from inside SQL Server Management Studio or from the command prompt by typing **DTExecUI**.

Exercise (Working with the Execute Package Utility)

In this exercise, you will use SQL Server Management Studio to start this utility and run the Mailing Opportunities package.

1. While connected to Integration Services in SQL Server Management Studio, expand the MSDB folder below the Stored Packages area.
2. Right-click the Mailing Opportunities package, and choose Run Package. The Execute Package Utility dialog box shown in Figure 6-3 will appear.

   You can run your package by clicking the Execute button without any additional run-time options. However, you may want to go through some of the useful configurations while deciding how to run your package.
3. Click Configurations on the left pane to see the configuration files options. The Package Execution utility can read the configuration files to modify the properties of your package at run-time. You can specify the order in which these files are to be loaded. This is important, as the configurations load in order starting from the top of the list, and if multiple configurations modify the same property, the configuration that loads last prevails.
4. Similar to configuration files, the Package Execution utility can read the command files to include execution command lines to your package at run time. Click Command Files to see the interface. You can specify the order of your command files using the arrow buttons provided on the right side of this window.
5. Go to the Connection Managers page. You can edit the connection string of the connection manager that the package uses by selecting the check box provided in front of the connection manager. This option can be very helpful if you want to test your package against different data sources or so.
6. Open the Execution Options page. You can configure the run-time properties such as validation behavior and maximum number of concurrent executables, and specify checkpoints for the package here.

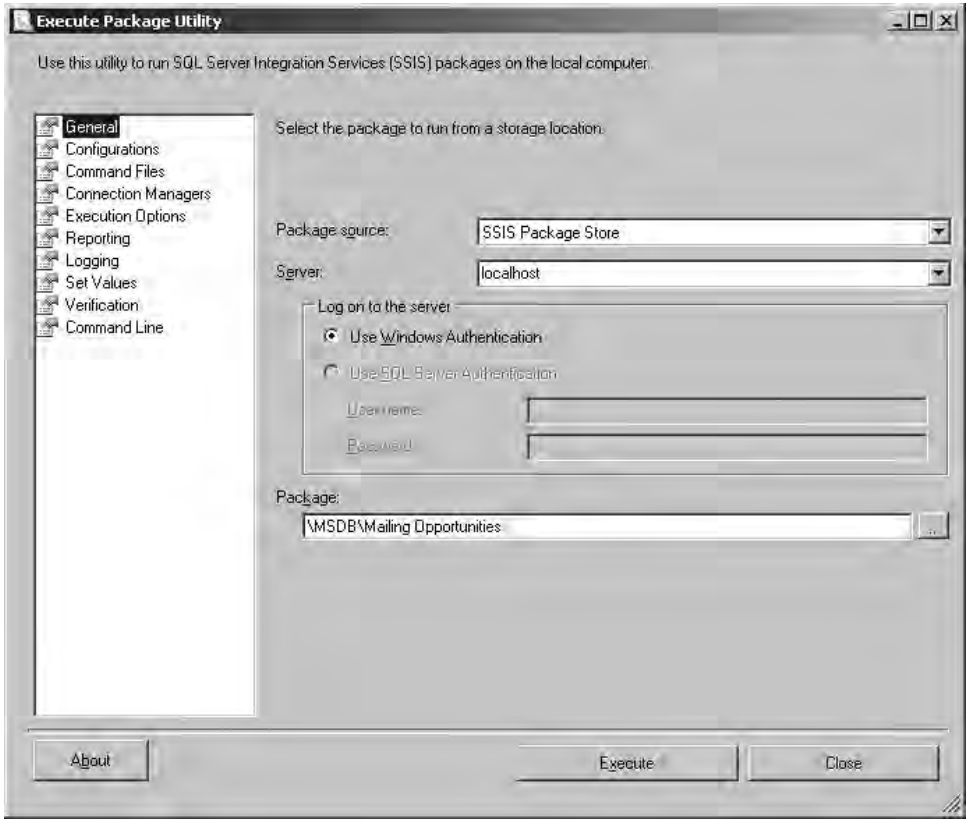


Figure 6-3 *Running the Execute Package utility*

Select Fail the package on validation warnings if you want your package to fail in case a validation warning occurs, and specify that the package be validated only by selecting the Validate Package Without Executing Option.

Notice the *–1* in the Maximum Concurrent Executables option. The value of *–1* means that the maximum number of concurrently running executables allowed is equal to the total number of processors on the machine executing the package, plus two. Alternatively, you can specify the maximum number of concurrent executables. The valid values are positive numbers starting from *1*. Specifying a 0 or any other negative value for this property will fail the package.

Checking the Enable Package Checkpoints option lets you specify a checkpoint file if you want to use one and also lets you override restart options already specified in the package with those specified here.

7. Move on to the Reporting page. This page has Console events and Console logging boxes, which can be configured with the numerous given options. You can use the Reporting page to set the options for the package by selecting the events and the information about the running package to report to the console. Console events specify the events and types of messages to report, whereas Console logging specifies the information that you want written to the log when the selected event occurs.
8. In the Logging page, you can tell the Package Execution utility which log providers to use at run time by specifying the log providers and the connection string for connecting to a log. The options for log providers are:
   - SSIS Log Provider For Windows Event Log
   - SSIS Log Provider For XML Files
   - SSIS Log Provider For Text Files
   - SSIS Log Provider For SQL Server
   - SSIS Log Provider For SQL Server Profiler

   Click in the field under the Log Provider column to choose from one of the available log providers. Then you can specify the connection string in the Connection String field.
9. The Set Values page is probably the one you will be using most often when you want to update property values of packages, executables, connections, variables, and log providers at run time. This useful feature makes running and testing packages easier. You can modify the property values by typing in the modified value and the path to the property. The path can be specified using a backslash (\) before the container, the period (.) before the property, and including a collection member within the brackets.
10. In the Verification page, you can set the attributes that must be met before allowing a package to run. For example, you can set the attributes to run only signed packages. Other options are to verify the package for the build number, package ID number, or version ID number.
11. Finally, the Command Line page allows you to review and optionally edit the command (see Figure 6-4) that has been generated by the options you selected in the various pages.

    Select the Edit The Command Line Manually radio button to make changes in the command line. If you accidentally make mistakes here and want to undo your changes, you can do so by selecting the Restore The Original Options radio button. Please note that when you select options in the other pages of the Package Execution utility, the command line gets updated; however, this is not the case if you make changes to the command line—i.e., these changes don't get reflected in the options of relevant pages. Also, these changes are not validated for the same reason.
12. Now click Execute to run the package, and you will see the progress of the running package in Package Execution Progress window.

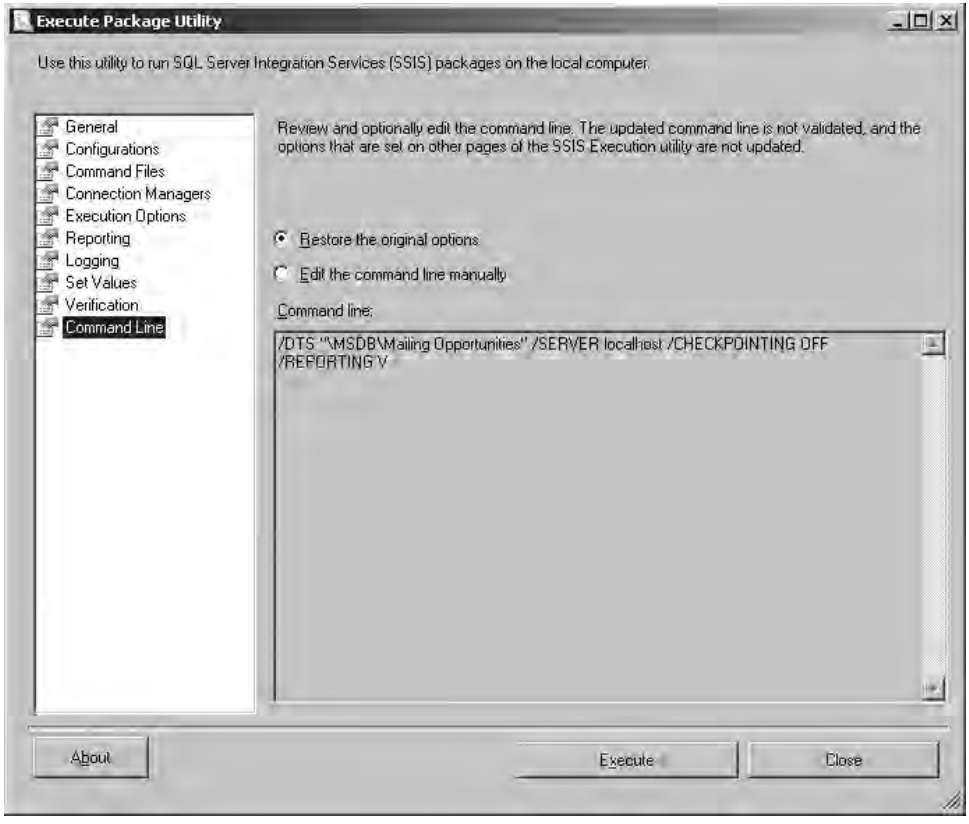


Figure 6-4 *Command generated by the Package Execution utility*

Review

In this exercise, you learned about the Package Execution utility that you can run from within the SQL Server Management Studio or from the command prompt. You also learned various options and how to set values for the properties at run time. You saw that this utility generates a command line that can be copied to a file and used with the CommandFile option in DTExec, or pasted directly to the Command Prompt window when running a package using DTExec. The DTExecUI user interface provides the facility to select several options that you find difficult to write syntactically correct and create command lines that can be passed on to the DTExec command prompt tool.

The DTExecUI and DTExec utilities have a couple of differences worth noting. First, DTExecUI is a GUI-based tool that helps you create command lines easily, but lacks the ability to automate the execution of packages. On the other hand, DTExec is a handy and powerful command prompt tool that is a bit difficult to program, especially

if you are not deploying packages very often, but once the command line is created, it can automate running of packages with the help of the SQL Server Agent service. So you will use DTExecUI when you want to run a package one off or manually and also when you want to write a complex command line for running a package automatically using DTExec and the SQL Server Agent. The second difference that can cause much grief on a 64-bit server is that the DTExecUI is a 32-bit only utility, whereas DTExec comes in both 64-bit and 32-bit flavors. You might get conflicting results if you're running your packages using different versions of the utilities.

DTExec Utility

SSIS provides the command prompt utility DTExec to configure and execute SSIS packages; it replaces the DTSRun utility provided in Data Transformation Services of SQL Server 2000. DTExec is a powerful utility that lets you access packages saved in the SQL Server, on the file system or in the SSIS Package Store. It allows you to configure connections, checkpoints, variables, logging, and reporting for the package you want to run.

With DTExec, you can run packages on the local server unless you use the SQL Server Agent or custom-build application to run DTExec commands from a remote computer. The SQL Server Agent allows you to automate the execution of your SSIS packages using the DTExec utility. You can select the Operating System (CmdExec) Agent subsystem, while adding a step to the job and use the DTExec command directly in the command area. As mentioned earlier, you can use the DTExecUI utility to create a command line with all the options you require and use it directly with DTExec.

By default, the DTExec utility is located in the <drive>:\Program Files\Microsoft SQL Server\100\DTS\Binn folder. DTExec comes in both 32-bit and 64-bit versions, so on a 64-bit server if you install 32-bit utilities, a 32-bit version of DTExec may also exist in the <drive>:\Program Files(x86)\Microsoft SQL Server\100\DTS\Binn folder. In such a case, the environment variable PATH contains the 32-bit path before the 64-bit path, making 32-bit DTExec run by default from a command prompt. However, the SQL Server Agent doesn't use this path statement; rather, it uses a registry setting to identify the DTExec location and will run the 64-bit version of DTExec. Make sure you test your packages against the correct version of DTExec before creating SQL Server Agent jobs with this utility. You can, though, avoid this situation by modifying the order of paths in the PATH environment variable.

The basic syntax for the command is shown next; the parameters are specified as option-value pairs. The options are specified beginning either with a minus sign (–) or a slash (/) and are not case-sensitive, barring passwords.

```
DTExec /option [value] [/option [value]]...
```

When the DTExec utility is executed, it passes through different phases, depending upon the parameters specified in the command line, and in the end, it can return an exit code as well. Let's study these phases and the parameters that decide the behavior of DTExec.

Command Sourcing Phase

This is the initialization phase in which the DTExec utility reads the parameters—i.e., the list of options and arguments specified. If you want to see only the help on the commands using a `/?` or `/H` option, for example, DTExec will run for this phase only and all the subsequent phases will be skipped. If you specify an argument along with help options, DTExec will display help for that argument.

```
DTExec /H DTS
```

`/Rem` is another option that is parsed in this phase. Using `/Rem`, you can include remarks on the command prompt or more appropriately in command files. These remarks are discarded during the command sourcing phase.

In the command sourcing phase, if the `/CommandFile (/com)` option is used, DTExec opens up the file specified with the `/CommandFile` option and reads all the options from the file until EOF is reached. You can either use the complete option name or use the short form of the option shown in parentheses with each option. The file specified with `/CommandFile` option is a text file that contains additional DTExec command options.

Here's an example: Open a blank text file in Notepad and type **/DTS "\msdb\ mailing opportunities"** at the beginning of the file. Save this as **C:\dtscmds.txt**. Then go to a command prompt and type this command:

```
DTExec /com "C:\dtscmds.txt"
```

You will see that the Mailing Opportunities package has executed successfully. DTExec doesn't put any restriction on how you write command options—i.e., on the same line or on different lines in the command file.

Package Load Phase

After passing through the first phase, DTExec goes on to load the package from the storage specified by the `/SQL`, `/FILE`, or `/DTS` option.

To load a package stored in the MSDB database of the SQL Server, type the following command using the `/SQL (/SQ)` option:

```
DTExec /SQL "Mailing Opportunities"
```

The `/File` (`/F`) option loads a package saved in the file system. The file path can be specified in either Universal Naming Convention (UNC) format or as a local path.

```
DTExec /F "C:\SSIS\Packages\Mailing Opportunities.dtsx"
```

Don't forget to include the file extension when using the `/File` option.

Finally, to load a package saved in the SSIS Package Store, you will use the `/DTS` (`/DT`) option.

```
DTExec /DTS "\MSDB\Mailing Opportunities"
```

You can use only one of these three options, as they are mutually exclusive. Using these options together will cause the package to fail. You can also use the following supportive options:

- `/User` can be used with the `/SQL` option when the package is protected by SQL Server Authentication. If you omit the `/User` option, Windows Authentication is used. It is recommended that you use Windows Authentication whenever possible.
- `/Password` (`/P`) option is used with the `/User` option. If you omit the `/Password` option, a blank password will be used.
- `/Decrypt` (`/De`) is used to load the package that has been encrypted with a password encryption; this option is used to set the decryption password.
- `/Server` (`/Ser`) is used to specify the server from which to retrieve the package when the `/SQL` or `/DTS` option is specified. If you omit the `/Server` option when using `/SQL` or `/DTS`, the local server default instance will be used to retrieve the package.

The following command will run the package stored at the RemoteServer:

```
DTExec /SQL "Mailing Opportunities" /Ser RemoteServer
```

Configuration Phase

After the package is loaded, DTExec goes on to process rest of the options that set flags, variables, or properties; verify the package versioning; and specify the logging and reporting. These options are as follows:

- The `/CheckPointing` (`/CheckP`) option is used when you want to include checkpoints during package execution. The valid values are *on* or *off*. The default is *on* when the option is specified without a value. The checkpoints enable a failed

package to rerun from the point of failure. For example, if you are loading multiple tables in a data warehouse and your package fails halfway through, then using checkpoints will enable this failed package to rerun from the failed task position and save the processing for already imported tables. To achieve this, the DTExec utility records the current position in a file called CheckFile.

- The `/CheckFile` (`/CheckF`) option is used by DTExec to specify the checkpoint file so that a failed package can be restarted from the point of failure. The filename specified with the `/CheckFile` option is used to update the value of the CheckPointFileName property of the package.
- The `/ConfigFile` (`/Conf`) option is used at run time when you want to set different configuration values than what were specified at design time. The run-time values can be stored in an XML file and loaded before executing the package using the `/ConfigFile` option.
- The `/Connection` (`/Conn`) option can be used to change the connection managers at run time. You have to specify the connection manager name or ID used in the package along with the connection string to which you want to change.
- The `/ConsoleLog` (`/Cons`) option can be used to display the specified log entries to the console (see Figure 6-5). You can specify the columns you want to see by using the *displayoptions* parameter, for which the values include *N* for Name, *C* for Computer, *O* for Operator, *S* for Source Name, *G* for Source GUID, *X* for Execution GUID, *M* for Message, and *T* for time. If you do not specify

Figure 6-5 *Use of the `/ConsoleLog` option displays the execution log on the screen*

the option parameters, no column is displayed, and if you specify the option parameters, all the values for the selected columns will be displayed that you may want to limit. For limiting the display of log entries on the console, you use extended options that are called *list_options*. The list_options allow you to log only the sources that are specified with parameter (`I`) for inclusion or only the sources that are not explicitly excluded with parameter (`E`) using a list.

- The `/Logger` (`/L`) option lets you choose one or more log providers to be used at run time by specifying an ID and a connection string to establish a connection to the destination store for log entries. The options of the log providers are as follows:
  - SSIS Log Provider for Text Files
  - SSIS Log Provider for SQL Profiler
  - SSIS Log Provider for SQL Server
  - SSIS Log Provider for Windows Event Log
  - SSIS Log Provider for XML Files
- The `/Reporting` (`/Rep`) option works as follows: By default, DTExec will report errors, warnings, and progress of the events when the package is executed. However, you can specify the types of messages to report using the `/Reporting` option. You can choose from the options listed in the following table:

Option Symbol and Name	Description
N – No Reporting	A mutually exclusive option with all other options and should be specified alone.
E – Errors	Errors are reported.
W – Warnings	Warnings are reported.
I – Information	Informational messages are reported.
C – Custom Events	Custom events are reported.
D – Data Flow Task Events	Events related to Data Flow task are reported.
P – Progress	Progress of the execution of package is reported.
V – Verbose Reporting	A mutually exclusive option with all other options and should be specified alone.

- The `/Restart` (`/Res`) option specifies a new value for the CheckpointUsage property on the package. The possible parameters are shown next:

`Deny`	Sets CheckpointUsage property to DTSCU_NEVER.
`Force`	(Default) Sets CheckpointUsage property to DTSCU_ALWAYS.
`ifPossible`	Sets CheckpointUsage property to DTSCU_IFEXISTS.

- The `/Set` option is used to modify the configuration values of variables, properties, containers, log providers, Foreach enumerators, or connections at run time. Both the property path and the value are to be specified according to specific syntax rules. The syntax uses a backslash (`\`) as a container separator, a period (`.`) to delimit properties, square brackets (`[]`) to specify collection members, and a semicolon (`;`) to separate the property path and its value.
- The `/MaxConcurrent` (`/M`) option limits the number of executables running in parallel on your machine. You can specify a positive non-zero integer value or *–1*. A value of *–1* means the maximum concurrently running executables (total number of processors on the computer plus two) are allowed.
- The `/Sum` (`/Su`) option is an optional switch showing a counter that contains the number of rows that will be passed on to the next component.
- The `/VerifyBuild` (`/VerifyB`) option works like this: During the development stage when you will be making lots of changes to your package, you may want to make sure that the package is executed with a stable version of the build while you can carry on the development work without jeopardizing the normal running of the package. You can specify the build numbers you want to run and verify your package against those numbers. In case of a mismatch, the package will fail to run. The `/VerifyBuild` option has three arguments, major, minor, and build, which can be specified in three forms: *major* or *major; minor* or *major; minor; build*.
- The `/VerifyPackageID` (`/VerifyP`) option is similar to the `/VerifyBuild` option; you can specify the package ID with this option to let it verify against the package GUID.
- The `/VerifySigned` (`/VerifyS`) option is a security option to check that the package is signed. If the package is not signed or the signature is not valid, the execution of the package will fail.
- The `/VerifyVersionID` (`VerifyV`) option verifies the version ID of a package to be executed against the version ID you specify with this option.

Validation and Execution Phase

In this phase, the package is validated and then finally run. However, you can also choose only to validate the package without running it.

- The `/Validate` (`/Va`) option lets you validate only without running the package.
- The `/WarnAsError` (`/W`) option, specified along with the `/Validate` option, can go one step further by letting the package treat the warnings as errors and will fail if a warning occurs during validation phase.
- The `/Dump errorcode` option can create the debug dump files when one or more specified events—error, warning, or information—occur while the package is running. You can specify the events by specifying the error codes, separated by a semicolon (;), that you want to use to invoke the system to create the debug dump files. Integration Services creates a binary debug dump file with an extension .mdmp and a text-formatted debug dump file with an extension .tmp; by default, it stores these files in the <drive>:\Program Files\Microsoft SQL Server\100\Shared\ErrorDumps folder.
- The `/DumpOnError` option works like this: If your package has been failing and you want to debug the failures, you can create debug dump files. You can use this option if you want to create the debug dump files, .mdmp and .tmp, only when an error occurs while the package is running.

Finally, the DTExec utility can return an exit code depending upon the state of the package loading and execution. The exit codes are values from 0 through 6 that can be interpreted as follows:

Exit Code	Description
0	The package was executed successfully.
1	The package failed to execute.
3	The user canceled execution of the package.
4	The package could not be found at the specified location.
5	The package could not be loaded.
6	The command line could not be interpreted due to syntax or semantic errors.

SQL Server Agent

The SQL Server Agent can automate the running of SSIS packages. SQL Server Agent jobs can have one or more steps, each running a different package and a bit of work that the agent can do for you. The jobs can also be scheduled to run at specific times using extensive SQL Server Agent scheduling capabilities.

One thing worth mentioning here is that Integration Services is a server component and not a redistributable component, as was its predecessor DTS 2000. This means that you can't install SSIS (as redistributable software) on a client machine and execute SSIS packages as you could have done in the past with DTS 2000. Also, SSIS packages will always run locally even when they load a package from a remote server; hence, you can't run a package (outside the design environment BIDS) on a computer where Integration Services is not installed. However, there will be instances where users or applications need to run packages while Integration Services is not installed on the local computer. In those instances the SQL Server Agent (or any other scheduling agent) comes in handy, as it allows you to schedule and automate the execution of SSIS packages not only on local servers but also on the remote servers. You can start the jobs on remote servers using system stored procedure sp_start_job or SMO. The only drawback of this method is that after you've created jobs on the SQL Server Agent, you can't configure the jobs at run time; that is, you can't provide parameters or other options at run time that you can do with DTExec utility.

Hands-On: Automating Running an SSIS Package with SQL Server Agent

In this exercise, you will learn how to use the SQL Server Agent service to create and automate jobs for running SSIS packages.

Method

We will use the SQL Server Agent in SQL Server Management Studio to

- Create a new job and add a package (step) to the job
- Create a schedule to execute the package
- View the job history after running

For this exercise, we will again use the Mailing Opportunities package as a test package.

Exercise (Create a New Job and add a Package to the Job)

You will be creating an SQL Server Agent job and then will add an SSIS package to a step in the job here.

1. Start SQL Server Management Studio and connect to the database engine. Once connected, verify that the SQL Server Agent service is started. If not, start the SQL Server Agent service.

2. Expand the SQL Server Agent node in Object Explorer and right-click Jobs; then choose New Job from the context menu to open the New Job dialog box.
3. Type **Running Mailing Opportunities** in the Name field. Ensure that the Enabled check box is ticked before moving on to the Steps page.
4. Go to the Steps page and click New to create a new step for this job. In the Step Name field of the New Job Step window, type **Executing Mailing Opportunities Step**.
5. In the Type field, click the down arrow to display the list of job types and choose SQL Server Integration Services Package. As you choose this option, the window changes to include the options for running an Integration Services package.
6. In the Run As field, leave SQL Agent Service Account selected.
7. In the General tab, set the Package Source field to SSIS Package Store, type **localhost** in the Server field, and leave Use Windows Authentication selected in the Log On To The Server area. Click the ellipsis button next to the Package field to choose the Mailing Opportunities package under the MSDB storage folder. Click OK. Note the package path shown in Figure 6-6.
8. Click the Advanced page in the New Job Step window to see the available options. You can specify what should happen on the success or failure of a step. The default for success of a step is to go to the next step, and the default for the failure of a step is to quit the job and report failure. You can also specify the number of retry attempts and the time to wait between retries here, as well as record output in an output file or table. Click OK to close this dialog box. You should see the package listed in the Job Step List area. Notice the arrow buttons on the lower-left side of this page; you can use these to move your job steps up and down and hence change the order in which they are executed.

Exercise (Create a Schedule to Execute the Package)

In this section, you will be creating and adding a schedule to the job.

9. Move to the Schedules page. You can now create a new schedule for this job by clicking the New button or choose a schedule from the list of already defined schedules by clicking the Pick button. Once you create a schedule for a job, it is listed in the Available Schedules list, where you can pick a schedule for other jobs you define, saving you the time required to redefine similar schedules again. As you have not created any schedule until now, you will create a new schedule here. Click New to open a New Job Schedule window. In the Name field, type **Schedule for Mailing Opportunities package**. Leave the Recurring setting in the Schedule type field and leave the Enabled option checked.
10. In the Frequency area's Occurs field, choose Daily and leave the Recurs Every field set to 1 day.

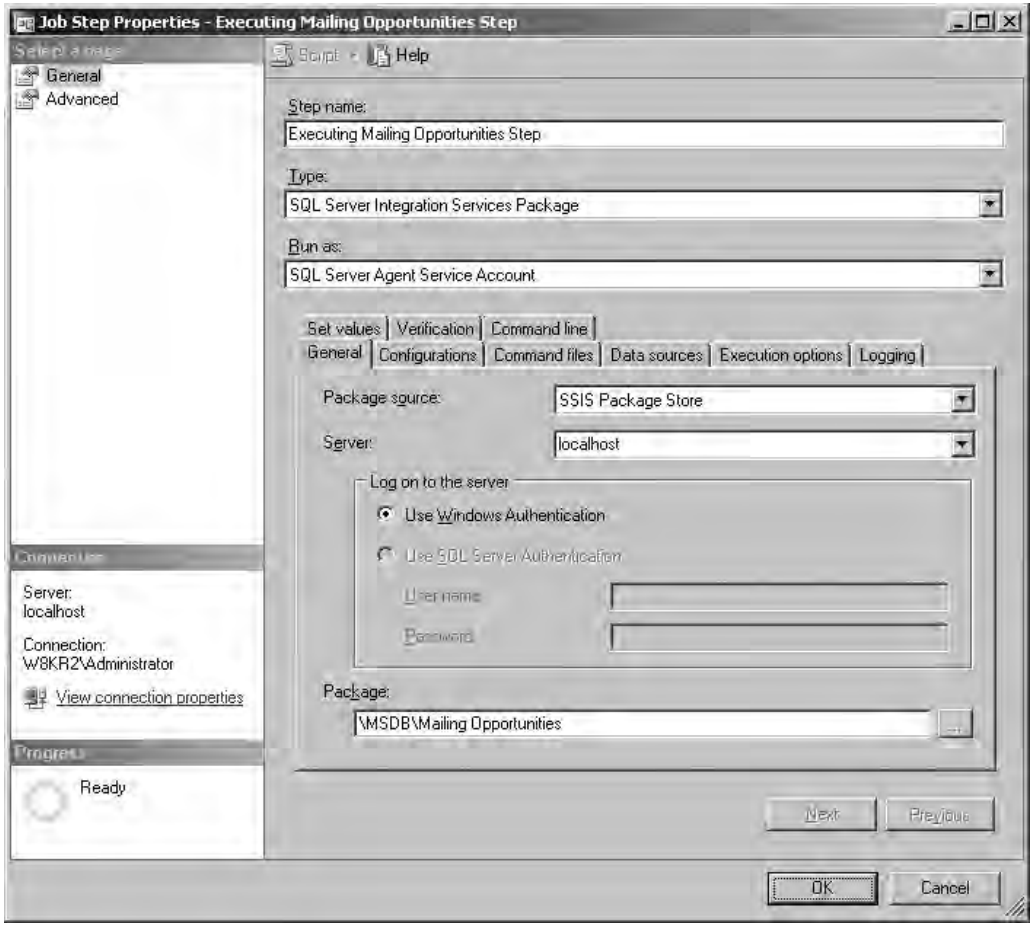


Figure 6-6 *Configuring a step for an SQL Server Agent job*

11. In the Daily Frequency area, select a time equal to the time when you want to execute the package. For the sake of quick testing, set the time about 10 minutes from now.
12. In the Duration area, you can specify when to start and when to stop executing the scheduled packages. Leave default values selected.
13. A summary of the schedule is shown in the Description box in plain English for your review (see Figure 6-7). Click OK.
14. In the next two pages of Alerts and Notifications, you can create alerts, define responses to those alerts, and send the notifications via e-mail, page, or Net Send or write to a Windows Application event log. Refer to Microsoft SQL Server 2008 Books Online for more details on these topics.

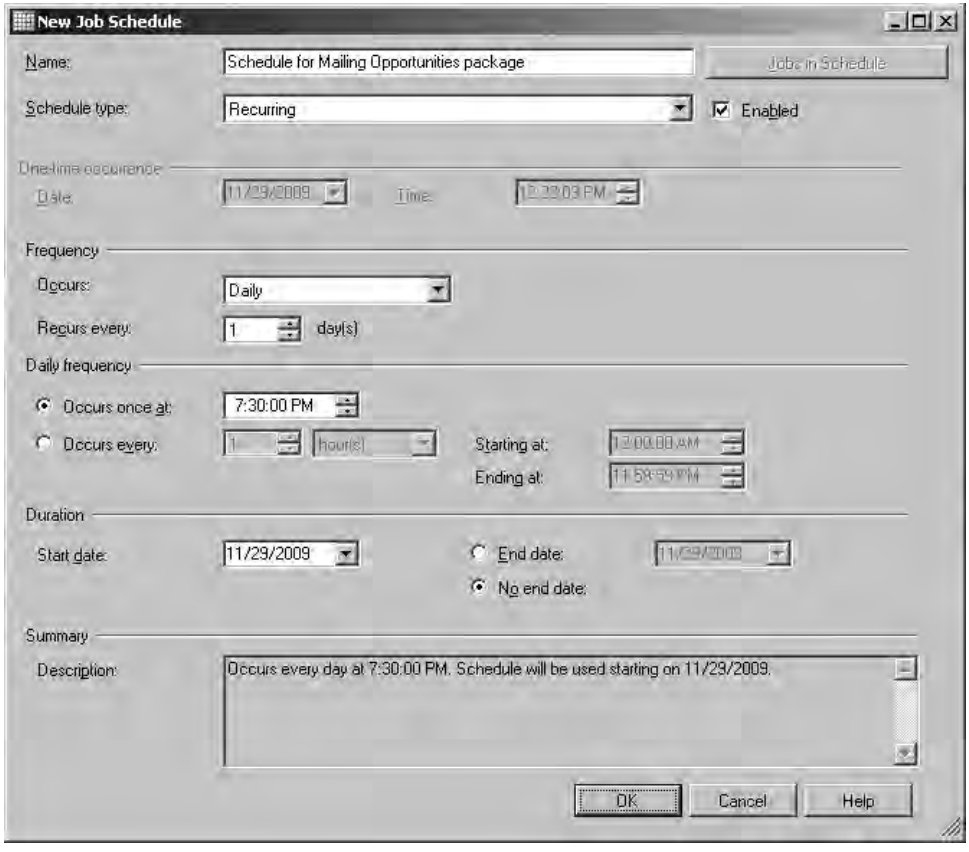


Figure 6-7 *Creating a schedule for a job*

15. The Targets page lets you define which server to target with this job. SQL Server 2008 includes a feature called *multiserver administration* to manage multiple servers by designating them a master server and target servers. The master server distributes jobs to the target servers and receives events from them to keep the status information. Target servers periodically connect to master servers to get the new schedules for the jobs allocated, download new jobs, or update the master server with the latest status about the jobs being run. You can create a master server or target server from multiserver administration by right-clicking SQL Server Agent in the SQL Server Management Studio. If no target servers are listed, the only option is to run the job from the local server. For more details on multiserver administration and how to make a master server or a target server, refer to Microsoft SQL Server 2008 Books Online.

16. Click OK to create the job. You will see this job listed in the Jobs node under SQL Server Agent. Have a cup of coffee, and by the time you return to your computer, you will see the test mails in your inbox.

Exercise (Using an SQL Server Agent Proxy Account to Run the Job)

In the previous part of the exercise, you've added the job step using an SQL Server Agent Service Account. In real life not many DBAs would like to run the jobs under such a privileged account. SQL Server 2008 does provide an alternate way to run the jobs under a less privileged account. To understand how the SQL Server Agent lets you do this, you should know a couple of things.

Firstly, note that several security enhancements have been made in SQL Server 2005 onward that affect the way jobs run in the SQL Server Agent. In the versions before SQL Server 2005, the SQL Server Agent service account had to be a member of the local Administrators group when executing the xp_cmdshell extended stored procedure, ActiveX scripting, or CmdExec jobs owned by users who were not members of the SysAdmin fixed server roles. This is not required in SQL Server 2005 and above, as it has a hierarchy of fixed database roles (in the MSDB database) to control access to the SQL Server Agent. A user must be a member of any of the three fixed database roles in order to use the SQL Server Agent. These roles, SQLAgentUserRole, SQLAgentReaderRole, and SQLAgentOperatorRole, are stored in the MSDB database.

Second, when you define a job step for an SQL Server Agent job, you choose a *Type* for the job as you did in Step 5. This job step is actually called an SQL Server Agent subsystem and is a security object that represents a set of functionality available to special accounts called the SQL Server Agent Proxy. The Sysadmin fixed server role can create multiple proxy accounts and assign each account to a separate step of an SQL Server Agent job. This means that you can create security contexts using SQL Server Agent proxy accounts limited to their own subsystems and hence to a job step only. You can have a proxy for multiple subsystems to keep things simple, but if you have a requirement to secure each and every job step, you can do so. Ideally, you will not run the SQL Server Agent service under a Microsoft Windows account that has administrative privileges but will create dedicated proxy accounts with necessary permissions required to run the job steps for the required subsystems and will use these proxy accounts to configure the job steps. Enough theory, it's time to do the hands dirty.

17. Create two windows accounts as follows that we will use in Chapter 7 as well. Choose Start, right-click Computer, and choose Manage from the context menu. In the Server Manager window, expand Configuration and then the Local Users and Groups folder, right-click the Users folder, and choose New User from the

context menu. This will open a New User window where you can create a user. Fill in the following details to create a user account:

User name	ISUser01
Password	ISUp@ss01

Clear the User Must Change Password At Next Logon option check box before clicking the Create button.

Create another user with the following details:

User name	ISUser02
Password	ISUp@ss02

Close the New User and Server Manager windows.

18. Next create logins for both users in SQL Server Management Studio.

    Assign db_datareader role to ISUser01on the Campaign database. To do this, go to User Mapping page, select the Campaign database in the Users Mapped To This Login section, and select db_datareader in the Database Role Membership section. Click OK to create this login.

    Assign SQLAgentOperatorRole to ISUser02on the MSDB database. To do this, go to User Mapping page, select msdb as the database in the Users Mapped To This Login section, and select SQLAgentOperatorRole in the Database Role Membership section as shown in Figure 6-8. Click OK to create this login.

19. In this step we will create an SQL Server Agent proxy account and will link this account to ISUser01. To link these accounts, we will need to use a credential. So, let's first create a credential for ISUser01. In Management Studio, expand the Security node, right-click on the Credentials node and select New Credential from the context menu. Type **ISUser01** in the Credential Name field. Select the Windows user account ISUser01 in the Identity field by clicking the ellipsis button. Finally fill in the password fields for the ISUser01 account and click OK.

    Now expand SQL Server Agent and right-click the Proxies node. Select New Proxy from the context menu. Type **ISUser01** in the Proxy Name field and select ISUser01 in the Credential Name field by clicking the ellipsis button. Now select SQL Server Integration Services Package in the Active To The Following Subsystems section (Figure 6-9). Click OK to create the proxy account.

20. Until now you've created accounts—ISUser01 with sufficient permissions to execute the package and ISUser02 with rights to start the job. Let's now modify the job to use the proxy created earlier. Open the Running Mailing Opportunities Job properties from the Jobs node. Go to the Steps page and click Edit to edit the step. In the Run As field, select ISUser01 proxy from the drop-down list. Click OK twice to close the Job Properties window.

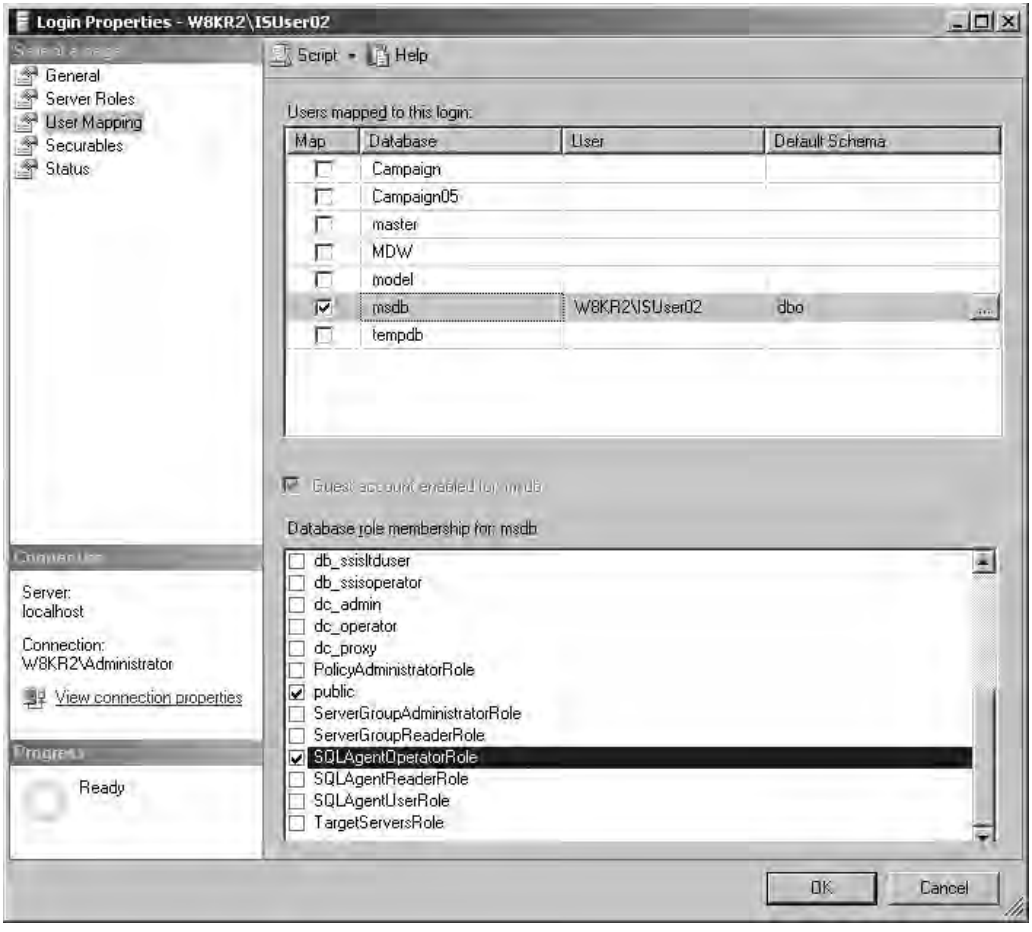


Figure 6-8 *Assigning an SQL Server Agent operator role*

21. Now, Logon with ISUser02 account. Open SQL Server Management Studio and expand the Jobs node under SQL Server Agent. Right-click the Running Mailing Opportunities job and choose Start Job At Step from the context menu. You will see the job executed successfully. So, in the preceding step you've run a job using an account ISUser02 that had no permissions on Campaign database and with the proxy account ISUser01 that had just sufficient permissions to execute the job successfully.

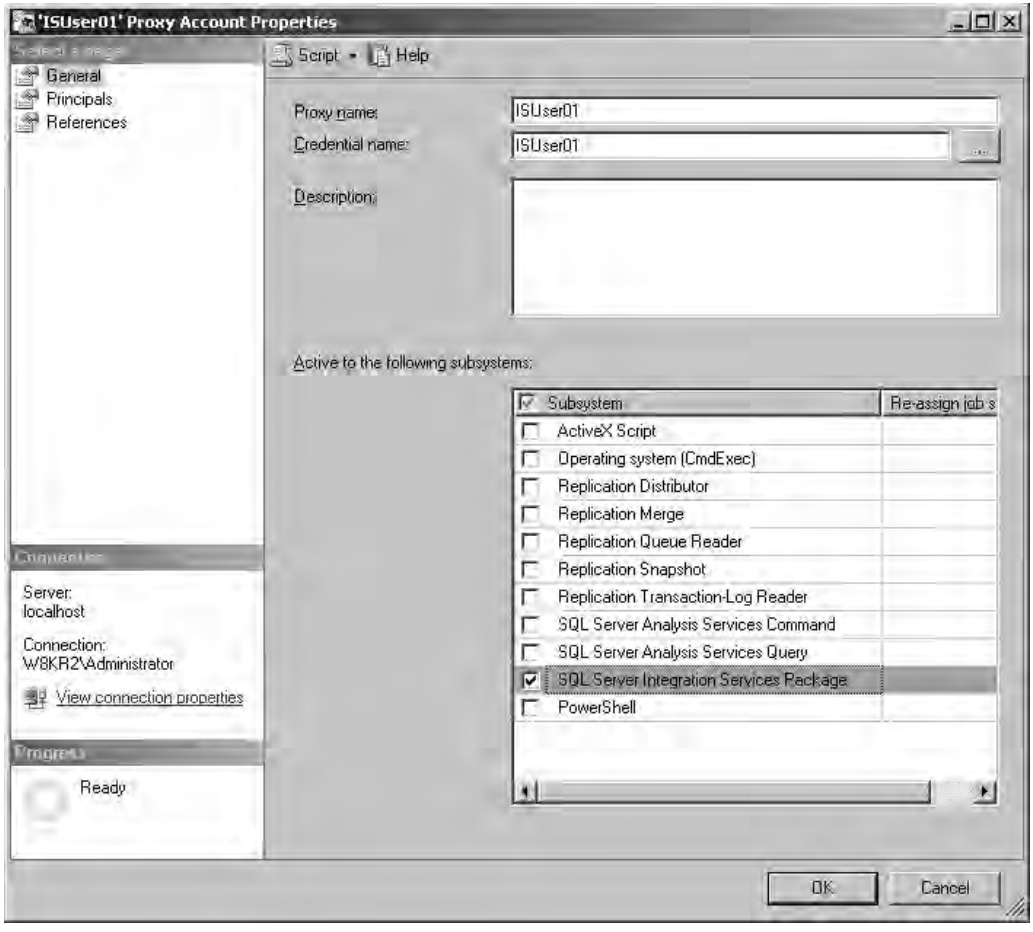


Figure 6-9 *Creating a Proxy for SQL Server Integration Services Package subsystem*

Exercise (View the Job History)

Let's now study how you can obtain the status of a job:

- You can configure a notification to be sent to you via e-mail while configuring the job in the Notifications tab.
- You can choose to write to the Windows Application event log in the Notifications tab while configuring the job.
- You can select to see the Job History from the Report drop-down list box in the Summary sheet of Jobs in SQL Server Management Studio.

- To see the detailed job history, you can right-click the job and choose View History. When you choose the View History option, you will see the history as shown in the Log File Viewer window (see Figure 6-10).

Review

In this exercise, you learned how to automate execution of an SSIS package using the SQL Server Agent. You learned how to create a new job, add steps to the job, specify a schedule for the job, and view job history. You have also seen some of the key features that make creation of jobs in the SQL Server Agent much safer and easier to run, such as:

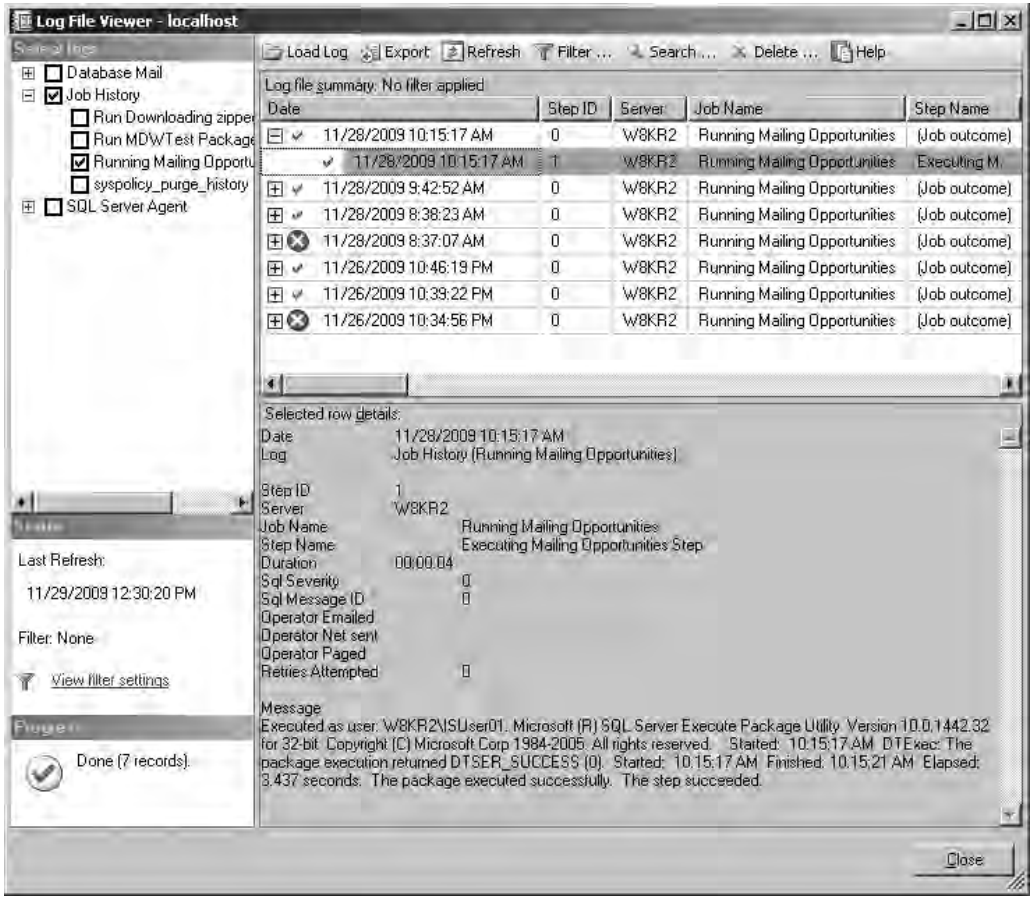


Figure 6-10 *Job History in Log File Viewer window*

- The user account under which the job executes doesn't need to be sysadmin anymore; you can use a proxy account to run the packages.
- The steps of a job have a type that is, the agent subsystem—SQL Server Integration Services Package—which allows you to configure all the options required to run a SSIS package in this category.

Executing SSIS Packages Programmatically

SSIS packages can also be run programmatically using the SSIS Object Model. You can build a custom application such as a Windows Forms application, an ASP.NET Web form, or a Web service that allows users to run SSIS packages on local or remote servers. SSIS exposes a rich object model that allows you to not only manage and run SSIS packages but to build custom objects and packages to extend SSIS functionality. More details on programming facilities in SSIS are covered in Chapter 11. Taking you through the custom application development for this exercise is out of scope of this book; however, the steps that you will take while building a custom application to run SSIS packages are listed here.

1. First of all, you will add a reference to the Microsoft.SqlServer.ManagedDTS assembly in your project, as it provides access to the managed run-time engine.
2. Import the Microsoft.SqlServer.Dts.Runtime namespace in your application, as it will allow you to use classes and interfaces to load and run packages. The Microsoft.SqlServer.Dts.Runtime namespace is commonly used in custom applications, as it contains the classes and interfaces to create packages, custom tasks, and other package control flow elements.
3. The Application class provides a mechanism to discover and access package objects. You will use the Application class to discover and instantiate the package object.
4. If you want to run the package on a local server, you will load the package first before executing it. You can load a package using different methods based on the storage type.
   - **LoadPackage** Loads a package stored in the file system
   - **LoadFromSqlServer** Loads a package stored in the SQL Server
   - **LoadFromDtsServer** Loads a package stored in the SSIS Package Store

For running a package on a remote server, you can use the LaunchPackage web service method that loads the package using one of these methods based on the source or storage type. You can also choose to run a package on the remote server using the sp_start_job system stored procedure inside your application, in which case, you don't have to follow any of the preceding steps, but simply add a reference to the System.Data assembly and build your application as you would to execute a stored procedure on a remote server.

5. Irrespective of the fact that you can build an application to run SSIS packages locally or on a remote server, SSIS packages execute on the server where the SSIS service is installed. So, if you are executing packages locally, the SSIS service has to be installed on the local server and if you are executing packages on the remote server, the SSIS service has to be installed on the remote server.
6. Finally, you use the `Execute()` method to execute the package.

Summary

SQL Server 2008 comes with several tools that enable you to manage Integration Services and Integration Services packages effectively and easily. You started this chapter learning to manage storage locations for storing Integration Services packages. You've used SQL Server Management Studio and dtutil to manage Integration Services packages and storage areas. You also learned how to import and export an Integration Services package from one storage location to the other. In the process, you created a new storage location on the file system for these packages. Last, but most important, you learned about various tools to run an Integration Services package that include the use of the Execute Package Utility and the DTExec command-line utility. In the end you automated running of a package by creating a job and its schedule using the SQL Server Agent. In the next chapter we will discuss securing Integration Services packages and use of fixed database-level roles to control access to SSIS packages.

This page intentionally left blank

Chapter 7

Securing Integration Services Packages

In This Chapter

- Digitally Signing the Package
- Excluding Sensitive Information from the Package
- Encrypting Sensitive Information in the Package
- Encrypting All the Information in the Package
- Using Integration Services Fixed Database-Level Roles
- Considerations for Different Storage Areas
- Summary

Security in SQL Server 2008 Integration Services has been enhanced a great deal compared to DTS 2000. DTS uses package password protection, SQL Server Security, and SQL Server Agent service security, while Integration Services provides the features used by DTS and a lot more to enhance data security. SSIS provides the ability to secure data and connections from various perspectives, depending upon the situation. By design, Integration Services will communicate with SQL Server only over an encrypted channel to protect sensitive data. In Integration Services the *sensitive information* means the passwords used in connection strings, any property of the custom-built components that has the sensitive attribute set, or any variable tagged with the sensitive attribute.

Integration Services secures your packages and data by providing the facilities to do the following:

- Digitally sign the package.
- Exclude sensitive information from the package.
- Encrypt sensitive information in the package.
- Encrypt all the contents of the package.
- Control access to the package by using database-level roles.
- Secure storage areas.

Let's take a detailed look at these options and what they offer in terms of securing Integration Services packages and the metadata used in them.

Digitally Signing the Package

Development of a complex Integration Services solution involves several developers who create many smaller packages to join as modules and form a complex solution for the business problem. During development phase, a package that has been tested successfully to perform a part of the function can be deployed while it is still under development for additional functionality. In such a scenario, you need to avoid the deployment of modified packages while they are still under testing. For example, you may be working to solve a complex scenario for which you have proposed a solution that can be developed and deployed in multiple stages. While development is still underway and many developers have access to SSIS packages, the last thing you would want to do is to run an untested package in the production environment.

You also want to make sure that you run packages only from trusted sources. To identify the source of a package and guarantee the integrity of packages, you can digitally sign a package with a certificate and configure Integration Services to check for the presence and validity of the digital signatures. So, each time the package is loaded, it is verified for digital signatures and hence altered packages wouldn't be loaded. You need to have a digital certificate installed on the server to digitally sign your packages. Once you have that in place, all you need to do is follow these instructions:

1. Using Business Intelligence Development Studio (BIDS), open the package you want to digitally sign.
2. On the menu bar, click the SSIS menu and choose Digital Signing. This will open a Digital Signing dialog box displaying a message "This package is not signed."
3. Click the Sign button. Select a certificate to sign the package and click OK.
4. After signing the package, right-click anywhere on the blank surface of the designer and choose Properties from the context menu. Locate the CheckSignatureOnLoad property and set it to True. This will require that the digital signature on the package be checked every time the package is loaded.

Excluding Sensitive Information from the Package

Integration Services provides a facility to developers to mark certain information as sensitive data. This sensitive data is handled in a more secure way than the other metadata of the package. The examples of sensitive data are passwords, connection strings, or any other information marked as sensitive by a developer in a custom-built component. Once the components have been deployed, Integration Services identifies the sensitive properties automatically and doesn't let users change any of the sensitive attributes. This applies to the standard built-in components as well.

Integration Services provides a set of options to secure the information in a package using the ProtectionLevel package property shown in Figure 7-1. You can opt not to save sensitive data in the package. When you select the DontSaveSensitive option, the sensitive information is removed from the package while saving and is unavailable for future executions of the package. So each time you want to execute the package, you have to provide the required information in order for the package to run successfully. If you change this option to any other option later on, the sensitive information is populated with blank data and you will have to provide the sensitive information—i.e., passwords and so on—in the relevant place to make this information available in the package.

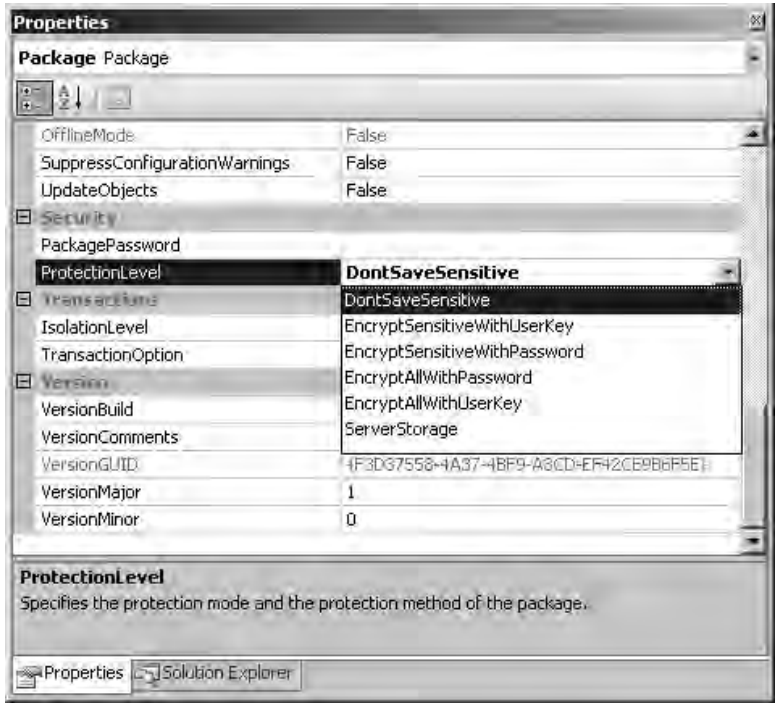


Figure 7-1 *ProtectionLevel property options of a package*

Encrypting Sensitive Information in the Package

The next scenario could be that you want to save sensitive information in the package and also want to protect this information. For this, Integration Services provides two options to encrypt this information in the ProtectionLevel package property—EncryptSensitiveWithUserKey and EncryptSensitiveWithPassword. These options are used to encrypt the sensitive information in the package using a user key or using a password. The Microsoft Data Protection API (DPAPI), which is a cryptography API, is used to fulfill the encryption needs of ProtectionLevel options that use a user key for encryption, while a Triple DES cipher algorithm with a 192-bit key length is used to fulfill the encryption needs of ProtectionLevel options that use a password for encryption.

EncryptSensitiveWithUserKey is the default encryption level for a package. This means that the sensitive information in a package is, by default, encrypted using the current user key, which has been created based on the user profile. Only the current user

using the same profile can load this package. If another user tries to load the package, the sensitive information fields are populated with the blank data and the package will fail to execute, unless the user trying to run the package provides the sensitive information.

The EncryptSensitiveWithPassword package protection level allows you to save the sensitive information in the package and encrypt it using a password, supplied in the PackagePassword property. By using a password as an encryption key for the sensitive information, you can let other developers open the package by supplying a password and hence make the package accessible to all members of the development team. Each time the package is loaded or the ProtectionLevel option is changed, the user must provide the package password. If the package password is not provided, the package is opened without the sensitive information. So to sum up, you will use the EncryptSensitiveWithUserKey option to encrypt the packages that you probably will not share with anybody else and the EncryptSensitiveWithPassword option when you want to share the package with others.

Encrypting All the Information in the Package

Two options are available for encrypting the whole package: EncryptAllWithUserKey and EncryptAllWithPassword. These options use a user key or a package password, respectively, to encrypt all the information in a package.

Select the EncryptAllWithUserKey option to encrypt all the information in a package using a user key. As the user key is generated based on the user profile, only the user who created or exported the package using the same profile can open or load the package.

Select the EncryptAllWithPassword option to encrypt all the information in a package using a password specified in the PackagePassword property. You can use this option to secure the contents of the package yet allow the development team to work on it; a custom-developed package for your application that includes an intellectual property is a good example for this. A package encrypted in such a way can be opened only by providing the password. You cannot load the package if you fail to provide the password.

Hands-On: Working with Package Protection Levels

This Hands-On exercise is designed to enhance the understanding of package protection levels.

Method

In this exercise, we will use each package protection level in turn to see how it works and the effects it has on the security of the package. We will use the Downloading zipped files package, as it requires a password to connect to an FTP server, to see the effects of using it with various protection levels.

Note that if you want to use the Downloading zipped files package that has been provided with this book, you will receive an error when opening the package. When you click OK on the pop-up error message, the package will load properly but without the connection string in the FTP task. This is because, by default, the sensitive information (passwords, connection strings, and so on) in the package get encrypted using the user key, and when another user tries to open the package, an error will occur and the sensitive information will be removed from the package. However, if you open the Downloading zipped files package that you developed yourself in Chapter 5, you will not get any such error.

In addition, this package requires a connection to an FTP server. If you've skipped building this package in Chapter 5, you should find an FTP server and build the package to complete this Hands-On exercise. The provided package may not be of much help as it is pointing to a computer used in the lab setup for this book, which is obviously not accessible to you. Better to use the package that you have created yourself.

Exercise (Excluding Sensitive Information from the Package)

After this exercise, you will be able to exclude sensitive information from the package using the DontSaveSensitive option of the ProtectionLevel property.

1. Open BIDS and create a new Integration Services project with the name **Downloading zipped files** in the location C:\SSIS\Projects. In the Solution Explorer window, delete the Package.dtsx package file. Right-click the SSIS Packages node and choose Add Existing Package from the context menu. In the Add Copy Of Existing Package window, choose File System in the Package location field and type **C:\SSIS\Projects\Control Flow Tasks\Downloading zipped files.dtsx** in the Package path field. Click OK to add this package in your project. Double-click the Downloading zipped files.dtsx package to open it.
2. Right-click anywhere on the blank surface of the Designer and choose Properties from the context menu. In the Properties window, you can view the properties in two ways—Categorized view or Alphabetical view. These views can be set using the two buttons provided in the command bar on the top of the Properties window. In the Categorized view, the properties are grouped together on the category basis, while the Alphabetical view simply lists the properties using alphabetical sort order. Use Categorized view.
3. Scroll down in the Properties window and locate the Security section. Note that the ProtectionLevel field shows EncryptSensitiveWithUserKey selected.
4. Press CTRL-R to open the Solution Explorer. Right-click the Downloading zipped files.dtsx package under SSIS packages folder and choose View Code from the context menu. The package code in XML will be shown in a new tab in BIDS.

5. Press CTRL-F and find *Password* in the XML document. You will be taken to the `ServerPassword` property that is immediately after ServerUserName in the XML document and is listed here:

```
<DTS:Property DTS:Name="ServerUserName">administrator </DTS:Property>
<DTS:Property DTS:Name="ServerPassword" Sensitive="1" Encrypted="1">AQAAANC
Mnd8BFdERjHoAwE/Cl+sBAAAAgp969y9CpkO6k07L3IdJGwAAAAAIAAAARABUAFMAAAADZgAAqA
AAABAAAABhZumzf3dqV1SXY5667BryAAAAAASAAACgAAAAEAAAAMW+xn039fmW+00yN32EHG4YA
AAAAE5rsrl9TvzImKtVSb+UWoZbYuJXBwtLFAAAAMTOWe+5xETOTECqeJbMTSIq/c9e
</DTS:Property>
```

   In this node, note that the `ServerPassword` property is attributed as sensitive data and is set for encryption. Also note that data in this node is all encrypted. This encryption is due to the default EncryptSensitiveWithUserKey setting.

6. Switch to the Designer tab of the package and choose the DontSaveSensitive option in the ProtectionLevel field in the Properties window.

7. Switch to Code view and search for *Password*. This time you will see the same XML node with no encryption attribute and no data in it:

```
<DTS:Property DTS:Name="ServerPassword" Sensitive="1"> </DTS:Property>
```

   This is because the password has been removed from the package.

8. Press F5 to run the package. The package will fail. Stop debugging and click the Execution Results tab. You will see the following error declaring that the password was not allowed:

```
[Connection manager "FTP Connection Manager"] Error: An error
occurred in the requested FTP operation. Detailed error description:
The password was not allowed.
```

9. Each time you start debugging a package, the package is saved using ProtectionLevel option; in this case, it won't save the password and hence is not executing. To execute this package, we have to provide a value to the ServerPassword property. You can do this by setting this value at run time either using Package Configurations or using a script task. We will cover both these methods in the later chapters when we cover scripting and package configurations in Chapter 11 and Chapter 13. For now, just keep in mind that a package that has been saved without the sensitive information can be run by supplying the sensitive (password) information.

Exercise (Encrypting Sensitive Information Using a User Key)

When you use a user key to encrypt the package, the package encryption gets associated with the user profile. We will use a test user account, ISUser01, to log on and open a package that has already been encrypted using a user key by another user, and we will

establish that the sensitive information is replaced when a different user tries to load the package. This package can be executed successfully only by providing the sensitive information in the package. You have already created this user account in Chapter 6.

10. Double-click the FTP Connection Manager in the Connection Managers area in the Designer and provide a password to connect to the FTP server in the Credentials section of the FTP Connection Manager Editor window. Click OK to close it.
11. Open the Properties windows and change the ProtectionLevel property value to EncryptSensitiveWithUserKey. Switch to the XML code for the package and search for *Password* to see that it has been encrypted, like the one shown in the preceding exercise.
12. Press F5 to make sure that the package executes successfully.
13. Save all the files, and then close all the applications and log off and log back on (or switch the user) as ISUser01 with the assigned password.
14. Start Business Intelligence Development Studio and open the Downloading zipped files.sln from the C:\SSIS\projects\downloading zipped files folder.
15. Open the Downloading zipped files.dtsx package. When BIDS tries to load the package, you will see an error on the screen informing you that the package could not be loaded due to errors and prompts you to see the Error List for details.
16. Click OK to close the error and the package will be loaded despite the errors. If you don't see the Error List window open in the lower left-hand corner of the BIDS, you can open it from View menu. In the Error List window, you will see the detailed error message explaining that the encryption key is not valid:

    ```
    "Error loading Downloading zipped files.dtsx: Failed to decrypt
    protected XML node "DTS:Property" with error 0x8009000B "Key not
    valid for use in specified state.". You may not be authorized
    to access this information. This error occurs when there is a
    cryptographic error. Verify that the correct key is available."
    ```

17. Press F5 to run the package. The package will fail. Press SHIFT-F5 to stop debugging. Go to the Execution Results page and read the error message, which states that the FTP password was not allowed. This establishes that the FTP password was removed when we tried to load the package as a different user.
18. Double-click the FTP Connection Manager in the Connection Managers area in the Designer and provide the password to connect to the FTP server in the Credentials section of the FTP Connection Manager Editor window. Click OK.
19. Press F5 to run the package; this time the package will succeed. This certifies that when the package is encrypted with another user key you can still load the package and use it if you know the sensitive information and can supply the correct password.

Exercise (Encrypting Sensitive Information Using the Package Password)

When you opt to encrypt a package using EncryptSensitiveWithPassword option, you then provide an encryption password using the PackagePassword property in the Security section of the Properties window. Here you will learn that if you encrypt the sensitive information in a package using a password, other users can access the sensitive information by specifying the PackagePassword. However, if other users try to load the package without specifying the PackagePassword, the sensitive information is replaced with blanks. You will be performing these steps while still logged on as ISUser01. In the following steps, you will use a package password to encrypt the sensitive information in the package.

20. Open the properties for the package and change the ProtectionLevel property to EncryptSensitiveWithPassword and specify a password **bB12345cC** in the PackagePassword field.
21. Open the XML code for the package. In XML code, if you try to find the word *Password* in the document, you will not get any result, because this word doesn't exist in the document. Instead, find the ServerUserName property, as you know that the ServerPassword property existed immediately after it. You will see something like this in the XML code view:


    ```
    <DTS:Property DTS:Name="ServerUserName">administrator </DTS:Property>
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element" Salt="oOBw/
    g9GpA==" IV="5YsCDRU2aMM=" xmlns="http://www.w3.org/2001/04/xmlenc#"><Enc
    ryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/
    ><CipherData><CipherValue>5YsCDRU2aMM9jrGvOlsQSXNFzBG13LDuBBBI/tK07k/Z1BX
    BYNSQEOWFYD3WgRhEDQ56TKlATw2Tvi7UU7OAJfDXDSnnoYPAwtmgTj3d/Qk72HJwlzNjqJ/
    FiGjC+2sfN4VNzpLSVGQCkV27tDchXriytPz/2pTI1EY58wui1LPAkulpSbunbg==</
    CipherValue></CipherData></EncryptedData>
    ```


    The data in the package has been encrypted using TripleDES with CBC algorithm.
22. Press CTRL-SHIFT-S to save all the items in the package. Close all the applications and log off. Log back on using the administrator user account.
23. Run BIDS and load the Downloading zipped files solution. You may have to double-click the Downloading zipped files.dtsx package in the Solution Explorer to load the package on the Designer. When BIDS loads the package, you will see the Package Password prompt to provide the password (Figure 7-2).
24. If you provide the correct package password, the package will load and you can run the package successfully. However, we will observe the behavior in case someone tries to load the package without the password. Click Cancel to load the package without the password.

Figure 7-2 *A password is required to open packages saved with the EncryptSensitiveWithPassword option.*

25. You will see an error message saying that there were errors while loading the package, which prompts you to see the Error List for more details. Click OK to proceed further.
26. The package will be loaded. Open the Error List window, and you will see an error explaining that the package will be loaded without the encrypted information:

    ```
    Error loading Downloading zipped files.dtsx: Failed to decrypt
    an encrypted XML node because the password was not specified or
    not correct. Package load will attempt to continue without the
    encrypted information.
    ```

    Press F5 to run the package, which will fail, specifying that the password was not allowed in the Execution Results pane. Click Stop Debugging.
27. In the Connection Managers area, double-click the FTP Connection Manager and specify the password in the credentials area. Click OK to close this window. Press F5 to run the package. The package won't run; instead it returns an error similar to the one shown in Figure 7-3.

    Close the error notifications. You get this error because you are still using the EncryptSensitiveWithPassword protection level and haven't provided a package

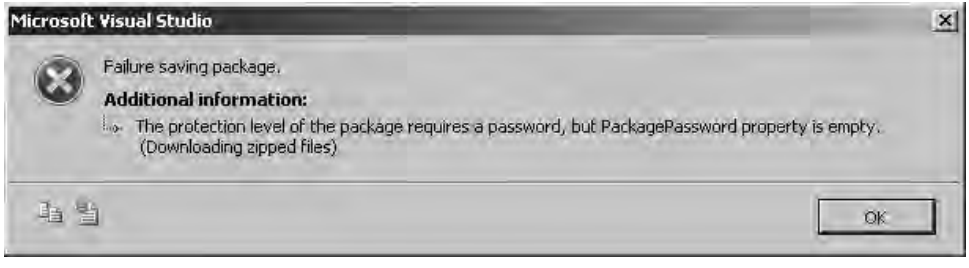


Figure 7-3 *Error returned on saving a package without specifying a package password*

password to encrypt the sensitive information. Until you supply the package password, you can't save the package and subsequently can't execute it. Click Stop Debugging.

28. At this time, you can supply a new package password to save the package, and as you've already provided the sensitive information, that is, the FTP server password, you should be able to save and execute the package. Type **yY56789zZ** in the PackagePassword field and press F5 to run the package. The package executes successfully this time.

Exercise (Encrypting All Information in the Package Using a User Key)

Using the EncryptAllWithUserKey option in the ProtectionLevel property encrypts all the metadata of a package with the user key so that no one else can load or run the package.

29. Switch to the package Properties window and change the ProtectionLevel property to EncryptAllWithUserKey.
30. Switch to the XML View Code tab to see that all the code of the package has been encrypted.
31. Save the package, close all the applications, and log off.
32. Log back on with the ISUser01 account and open the Downloading zipped files solution using BIDS.
33. Double-click the Downloading zipped files.dtsx package under the SSIS Packages folder to load the package.
34. The package still doesn't load. Instead, it returns an error saying that the package could not be loaded and the package might be corrupted. This is because BIDS could not remove the package encryption. Click OK, and then go on to read the detailed error message from the Error List window (Figure 7-4).

    As you can see, the Error List window displays the much-detailed error messages. This is unlike the EncryptSensitiveWithUserKey option, in which the sensitive information is replaced with blank information when other users try to load the package and then the other users can use the package by providing the sensitive information. Using the EncryptAllWithUserKey option, the other users can't load the package at all. This is the most restrictive protection level possible in SSIS. However, use of this protection level should be done with great care, as it ties down the package to the creator's profile; if the creator loses access to his or her profile, the package is rendered useless. The most eligible uses of this protection level may be ad hoc or short-term deployment of highly security sensitive packages.

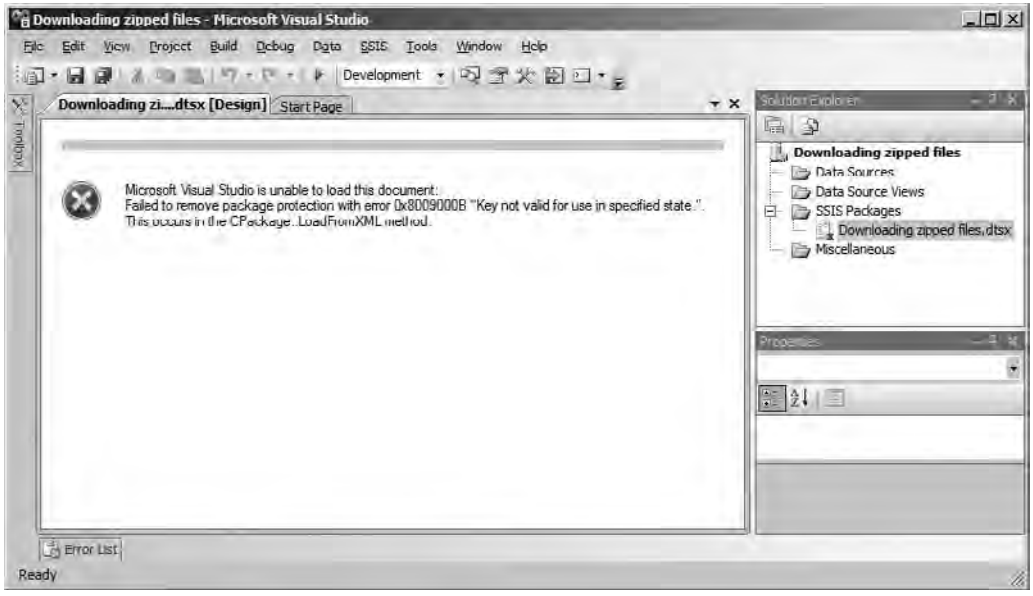


Figure 7-4 *Error loading package with the EncryptAllWithUserKey protection level*

Exercise (Encrypting All Information in the Package Using a Package Password)

In this exercise, you will learn that the EncryptAllWithPassword option of the ProtectionLevel property is the most secure option that allows sharing your package with other users.

35. Continuing from the previous steps, close all the applications and log off, as we cannot use the encrypted package with the previous encryption method. Log on using the administrator account and open the Downloading zipped files solution and the package using BIDS.
36. Open the package properties and scroll down to the Security category and change the ProtectionLevel property to EncryptAllWithPassword.
37. Click in the PackagePassword property and then click the ellipsis button, which appears when you click in the field. Provide the password **bB12345cC** in the Property Editor window, confirm it, and then click OK.
38. View the package code and notice that this time the encryption method used is TripleDES with the CBC algorithm. Press CTRL-SHIFT-S to save all the files. Close all the applications and log off.

39. Log on as the ISUser01 user and open the Downloading zipped files solution in BIDS.
40. The package may not load by itself; double-click the package in the Solution Explorer to load it. You will be asked to provide a password because the package is encrypted. If you provide the password, the package will load and you can successfully execute the package. But, for the sake of testing what happens otherwise, click Cancel here.
41. An error message will tell you that the package could not be loaded, as it might be corrupted, and asks you to see Error List for details. Click OK.
42. The package will not load and you will receive an error message on the Designer saying that Microsoft Visual Studio is unable to load the document, as the password was not specified or not correct. Almost similar errors appeared in the Error List:

    ```
    Error loading 'Downloading zipped files.dtsx' : Failed to remove
    package protection with error 0xC0014037 "The package is encrypted
    with a password. The password was not specified, or is not
    correct.". This occurs in the CPackage::LoadFromXML method.
    ```

    This exercise shows that using the EncryptAllWithPassword protection level option doesn't let other users use the package until they specify the correct package password. If the other users use a wrong password or do not specify a password, the package is simply not loaded and hence not available. This is quite useful feature, as it allows you to share your packages within a team yet keep it encrypted to avoid any unauthorized access.
43. Close the error message tab and open the package again by double-clicking the package file. Specify the correct password this time and the package will be loaded. You can further test this package by clicking Start Debugging to see that it executes successfully.

Review

You have used five out of six available ProtectionLevel property options in this exercise. You first learned that you could choose to exclude the sensitive information from a package using the DontSaveSensitive option. Then you learned the two options EncryptSensitiveWithUserKey and EncryptAllWithUserKey that use a user key to encrypt the sensitive or complete information in a package respectively. Later, you studied two more options to encrypt sensitive data or the complete package using an encryption password that has to be specified in the PackagePassword property: EncryptSensitiveWithPassword for encrypting sensitive information and EncryptAllWithPassword for encrypting the complete package. These options let you share packages with other users while keeping them secure from unauthorized access. The ServerStorage option that you have not used in this exercise is covered next.

Note that when you create and save a package, your package may use other miscellaneous files that may reside outside the package. Using the preceding protection options may

lead to a false impression that all the data is secured. So you should carefully consider all the files you refer to in a package and where these files are saved. These files may include checkpoint files, configuration files, or log files. You need to control access to these files using access control list permissions depending upon where they have been stored.

Using Integration Services Fixed Database-Level Roles

Integration Services enables you to control access to SSIS packages by providing three fixed database-level roles: db_ssisadmin, db_ssisltduser, and db_ssisoperator. These are available in the MSDB database and can be applied only to the packages that are saved to the MSDB database in SQL Server.

Fixed Database-Level Roles and Their Permissions

When you use the ServerStorage option for the ProtectionLevel property, the access to the packages saved to MSDB is controlled by fixed database-level roles. This is an additional security option you get when saving packages into SQL Server (as opposed to saving in File System) and gives you much more control for access permissions on the packages. Any user who needs access to a package must be a member of at least one of the three fixed database-level roles—db_ssisadmin, db_ssisltduser, or db_ssisoperator. By default, these roles have been assigned read or write privileges on the packages as described next:

- **The db_ssisoperator role** This is the weakest role among the three available roles. You generally assign this role to users who need to execute the packages. This role cannot perform any write operation—i.e., it cannot import or create packages and has only read permissions. It can list, view, execute, and export all the packages stored within a storage area. It can also schedule package execution using the SQL Server Agent service.
- **The db_ssisltduser role** This role is designed to give user access to developers who need to create and manage their own packages. As the name suggests, this role can perform functions limited to owned packages and can delete only owned packages and manage owned package roles, although they can import any package. And for the read functions, this role can view, execute, or export only the owned packages but can list all packages.
- **The db_ssisadmin role** This role is designed for administrators to manage packages and package roles. They have all the read and write permissions to

perform any operation on the SSIS packages. The role can list, view, execute, and export owned or other users' packages. This role can perform the write operations for importing, deleting, and managing package roles for owned or other users' packages. Last, to highlight that the server-level role sysadmin can also perform all the functions that of a db_ssisadmin role.

Other than these fixed database-level roles, Windows administrators can view execution details of all running packages in Management Studio and can also stop the running packages.

Let's save the Downloading zipped files package to MSDB and see how these roles affect the access to this package.

Hands-On: Control Access to a Package with User-Defined Roles

A user-defined role is another access control method that can be used in SQL Server 2008 Integration Services. The three fixed database-level roles can be used to create access control for user-defined roles to Integration Services packages. In this exercise, we will study how these roles work together and how they can be configured to control access.

Method

This is a three-step method, depending upon how much control you want.

- *Configure the ServerStorage ProtectionLevel.* To use database-level roles to protect SSIS packages, you first have to enable this by telling Integration Services to use the ServerStorage ProtectionLevel.
- *Add a user-defined role into the database-level roles.* After you've selected to store your package to SQL Server, you can add your user-defined role into the fixed database-level role to assign it preconfigured access permissions.
- *Assign package roles.* For finer control, you can further assign *package roles* to your user-defined roles. The packages stored in the MSDB storage area in Integration Services are saved in sysssispackages table of the MSDB database. The sysssispackages table has three columns—readrolesid, writerolesid, and ownersid—that define the access permissions for the packages saved in the table. If you've been assigned the Reader role on a package, your user SID will appear in the readrolesid column and you can do read operations on the package such as open, execute, and export the package. Of course, with the Writer role, you have write access to the package such as delete, rename, edit, and save a package. These roles are assigned while connected to Integration Services in SQL Server Management Studio.

Exercise (Applying ServerStorage ProtectionLevel to the Package)

You will start this Hands-on with server storage protection level. In this part, you will save a package to SQL Server, and will choose to rely on the server storage and the roles for access control.

1. Make sure you are logged on as administrator. Open the Downloading zipped files.sln solution using BIDS, and provide the package password when asked.
2. Choose File | Save Copy Of Downloading zipped files.dtsx As. This will open the Save Copy Of Package dialog box. Provide the following details in the dialog box, as we will be storing our package to SQL Server:

Package Location	SQL Server
Server	Localhost or provide your SQL Server name
Authentication Type	Windows Authentication
Package Path	/Downloading zipped files

   Click the ellipsis button in the Protection Level field and choose "Rely on server storage and roles for access control" for the Package Protection Level, as shown in Figure 7-5. Then click OK. Click OK again to close the Save Copy Of Package dialog box.
3. Choose File | Close Project and exit from BIDS.
4. Open SQL Server Management Studio and connect to Integration Services.
5. In the Object Explorer, expand the Stored Packages folder and then expand the MSDB folder. You will see a list of packages stored in the MSDB storage area. Right-click the Downloading zipped files package and choose Package Roles from the context menu.

Figure 7-5 *Selecting the ServerStorage option while saving a package*

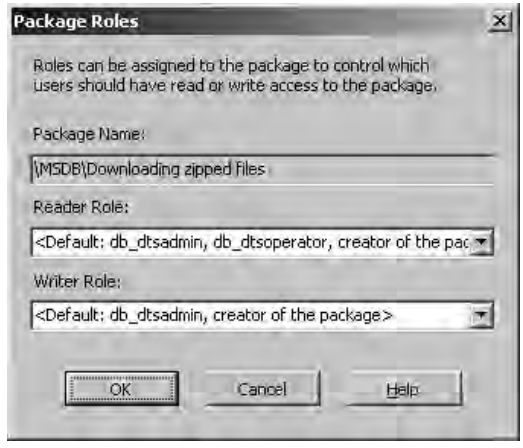


Figure 7-6 *Reader and Writer role assignments to default package roles*

6. This will open Package Roles window, showing the package name and the default settings for the Reader role and the Writer role (see Figure 7-6).

   First of all, if your screen is showing the same dialog box as is shown in Figure 7-6, you have hit a bug in SQL Server 2008 Integration Services. This bug has been reported: The default roles shown in Figure 7-6 are actually the names of SQL Server 2005 Integration Services roles. However, this is just a display issue and has not caused any functional issues to date. Also, note that by default, the Reader role is assigned to the db_ssisadmin and db_ssisoperator fixed database-level roles and the creator of the package, while the Writer role is assigned to db_ssisadmin and the creator of the package.

Exercise (Using Fixed Database-Level Role to Assign Execute Permissions to the Package)

The detailed permissions for the fixed database-level roles define that the db_ssisoperator is able to execute any package. Also, db_ssisoperator role is assigned a Reader role on the package by default. So, we can assign our user-defined role permissions to execute the package by adding it to the db_ssisoperator role. In this exercise, we will assign execute package permission to a user-defined role team and will execute our test package Downloading zipped files. The steps of the process are as follows:

- Create a login for the ISUser02 user account and a user-defined role team.
- Add the user to the team.
- Assign the team group to db_ssisoperator role.

In real life, to assign permissions to a group of users, you will create a user-defined role, make users members of this role, and assign permissions to this role to access the package and the database objects the package needs to access to run successfully. Finally, you will add the user-defined role to the pertinent fixed database-level role.

7. In SQL Server Management Studio, connect to the database engine, open a new query tab, and type the following query:

```
USE msdb
CREATE ROLE Team
EXEC sp_addrolemember 'Team' , 'W8KR2\ISUser02'
```

   This script creates a user-defined role Team and adds ISUser02 as a member to the role Team. We have already created a login and a user account for ISUser02 in the MSDB database in Chapter 6. Refer back if you've missed those steps. (Note that W8KR2 is the computer name used in my setup environment. You need to replace this name with the name of your computer or Domain name if working on Active Directory.)

8. Run the command prompt as user ISUser02 using Runas command on your operating system. I've used the following method on my Windows 2008 server; type the following command in the Run dialog box:

```
Runas.exe /user:W8KR2\ISUser02 cmd
```

   Enter the password when prompted and a cmd window opens up that is running as W8KR2\ISUser02.

9. Now run our test package as ISUser02 to check if this user has permissions to execute the package. Type the following at the command prompt and press ENTER:

```
DTEXEC /SQL "Downloading zipped files"
```

   The package fails with the following error:

```
Could not load package "downloading zipped files" because of
error 0xC0014062. Description: The LoadFromSQLServer method has
encountered OLE DB error code 0x80040E09 (The EXECUTE permission
was denied on the object 'sp_ssis_getpackage', database 'msdb', schema
'dbo'.). The SQL statement that was issued has failed.
```

   This is okay, as the ISUser02 user account doesn't have rights to execute the package yet.

10. Now, add the Team role to database-level db_ssisoperator role; this should be sufficient to assign the ISUser02 execute permissions, as the db_ssisoperator role has Reader Role privileges by default. Switch to SQL Server Management Studio and run the following T-SQL in the query pane:

```
EXEC sp_addrolemember 'db_ssisoperator', 'Team'
```

   This will add Team as a member to the fixed database-level db_ssisoperator role.

11. Switch to the command prompt opened earlier for user ISUser02. Try to execute the Downloading zipped files package again. You will get a success message this time:

```
DTExec: The package execution returned DTSER_SUCCESS (0).
```

Exercise (Using Package Roles for Granular Access Control to the Package)

By now you understand that you can quite easily assign permissions to user-defined roles and hence users or groups in your database using three fixed database-level roles. This is due to the fact that the fixed database-level roles have permissions on each package by default. If you need more control on packages and want to change Reader Role or Writer Role as per your needs, you can do so by removing default permissions from the package and assign the user-defined role directly.

12. Check the sysssispackages table in the MSDB database for the Reader role and the Writer role by running the following T-SQL command:

```
USE msdb
SELECT name, readrolesid, writerolesid from sysssispackages
WHERE name = 'Downloading zipped files'
```

    You will get NULL for both the readrolesid and writerolesid fields.

    If the Reader role and the Writer role have default fixed database-level roles assigned, you will see Nulls in these fields.

13. Go to the Integration Services Connection in SSMS, right-click the "Downloading zipped files" package under the MSDB folder and select Package Roles from the context menu. Select the Team in the Reader Role from the drop-down list as shown in Figure 7-7 and click OK.
14. Now, execute the same T-SQL used in Step 1 again to see the roles in the sysssispackages table. You will see the SID for role Team added in the readrolesid field.
15. Switch to the command prompt opened for user ISUser02 and execute the Downloading zipped files package. You will see a success message, as you've added the user-defined role directly in the Reader role after removing the default permissions for database-level roles.

Review

In this Hands-On exercise, you learned how to assign a group a reader role. You first created a user-defined role, then added the required database user to the role, and then linked that role to the fixed database-level role db_ssisoperator. Similarly, you can

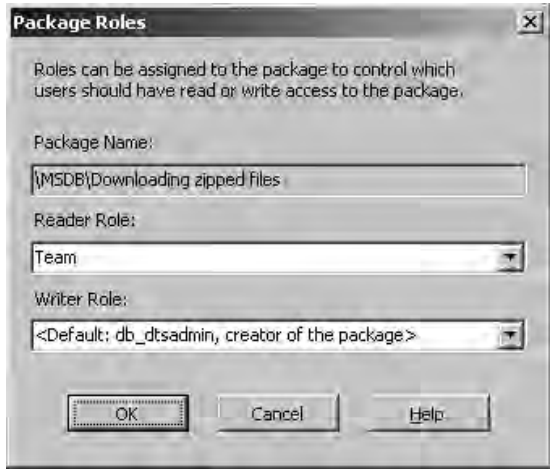


Figure 7-7 *Assigning a Reader role to the Team role*

assign a writer role to user accounts or groups. In the last part, you assigned the Reader role directly to the user-defined role and then linked that role to a fixed database-level role. Alternatively, you could have added users directly to the fixed database-level role, though that would not be a recommended approach. Whichever way you choose to go, make sure you assign a reader or writer role to the user or user-defined role and link them to the fixed database-level roles.

Another point worth mentioning here is that any metadata or configuration files that are saved outside the package cannot be protected solely by the database-level roles. Carefully consider where you want to store your configuration files or other miscellaneous files so that they are properly protected. Examples of such files that can be saved outside the package are configuration files, checkpoint files, cache files, or log files. If you are saving these files to SQL Server (but outside the package) or to the file system, you need to secure them with appropriate security measures. For more details on this, refer to the next section. Last but not least, you also need to assign permissions to user-defined roles on the objects (such as tables, views, and so on) being accessed by your package; otherwise, the package will run but will fail for not being able to access the objects.

Considerations for Different Storage Areas

You need to protect the Integration Services packages and the package metadata from unauthorized access or hardware failures. Integration Services packages may have associated configuration files that have been saved outside the package.

These configuration files are used to update values of properties at run time, making it easier to deploy packages from a development environment to a production environment. While both the package and the package configurations can be saved in the SQL Server or file system, they need to be protected irrespective to where they have been saved.

Considerations for Saving to SQL Server

When you save SSIS packages to SQL Server, you actually save them to the sysssispackages table in the MSDB database. If you are using a legacy package—i.e., the DTS 2000 package—Integration Services saves it to the sysdtspackages table in the MSDB database. This means when you save your packages in the MSDB database, they are protected by server, database, and table-level permissions. Packages developed from the ground up using BIDS (and not using the DTS 2000 package)—i.e., the packages saved in the sysssispackages table—are also protected by fixed database-level roles as you have studied earlier. Also, you can associate a reader role or a writer role to a package and can link these roles to user-defined roles.

You need to consider the protection of miscellaneous files attached to a package, e.g., package configurations that can also be saved to any SQL Server database table, not necessarily the MSDB database. When saved in an SQL Server database table, package configurations can be protected with server-, database-, and table-level permissions.

To protect your packages from hardware failures or miscellaneous types of data losses, you need to develop a backup plan that includes backup of the MSDB database, as this will back up sysssispackages and sysdtspackages tables stored in the MSDB database. For package configurations, you need to make sure that the table containing package configurations is also included in the backup plans to provide complete protection from hardware failures.

On the operational side, you know that the packages used and saved in BIDS, by default, go to the file system. When you want to work on a package in BIDS that has been saved to SQL Server, you'll need to import package into an Integration Services project that will create a local copy on the file system before you can edit it. After editing, when you want to save the package back to SQL Server, you will have to use the Save Copy Of packageName.dtsx As option to save it.

You can argue that if you save your packages to the file system, you wouldn't really need to have SQL Server database engine installed on the server. Your storage decision may be affected by the fact that most of the SSIS packages end up being scheduled using the SQL Server Agent, which in turn requires the SQL Server database engine. While thinking of having only Integration Services on a machine and avoiding SQL Server engine installation, you may think to use the SQL Server Express Edition as a database engine. However, it is worth mentioning that the SQL Server Express Edition doesn't support the SQL Server Agent.

Considerations for Saving to the File System

When saved to the file system, an Integration Services package is stored as an XML file (.dtsx), and package configurations can also be saved to XML file types. If you decide to save your packages and package configurations to the file system, you need to secure them using file system permissions. It's better to create a folder hierarchy for a solution to keep all the projects, their packages, configuration files, and other miscellaneous files if possible and protect this folder hierarchy using file system permissions. This also helps in making it easy to back up all the code and the files for related projects in the same backup job.

One consideration must be given while deciding the storage that how and where the packages will be deployed to production finally. As a developer, you can't decide about the production storage, as this is generally in the control of DBAs. It's not that you can't deploy to an alternate storage in production than what you've got in the development environment, but it's about making things easier for yourself, as you may find testing certain package configurations and access permissions difficult if the environments differ.

Summary

Security is the most often discussed topic these days in the IT industry. As a DBA or information analyst, you need to make sure that the contents of SSIS packages are secure. You have seen that the Integration Services provides a facility to sign a package to check for its integrity and provides six package protection levels, out of which four are for encrypting packages, one is for excluding sensitive information, and one is for using database-level roles. You have learned how to exclude sensitive information from your packages and encrypt sensitive information in packages using a user key or password. You have also used database-level roles to protect the package with read or write access controls. You should understand the various issues involved in deciding the storage location of packages. Remember to consider carefully where to store each file used in an SSIS solution and make sure these files are also protected.

Chapter 8

Advanced Features of Integration Services

In This Chapter

- Logging and Log Providers in SSIS
- Transactions in Integration Services Packages
- Restarting Packages with Checkpoints
- Expressions and Variables
- Handling Events at Package Run Time
- As a Data Source for Reporting Services Report
- Summary

By now, you've worked with most of the control flow tasks and have used their features and properties to create a work flow within your package to perform a job. In this chapter you will learn about some of the advanced functionalities built into Integration Services that you can leverage to extend functionality of your packages.

First, you will learn about logging and log providers in Integration Services, which help you to log the run-time events and provide auditing abilities. Then you will work with transactions and will learn how they can be used to maintain data consistency in the data stores. You will also do a Hands-On exercise on transactions and will learn about the three main scenarios in which you can deploy transactions. Use of checkpoints in packages provides the facility to restart a package from the point of failure and reduce the work that needs to be done to complete the regular data processing jobs. This is where you will do your next exercise on checkpoints and their usability and will restart a failed package. Though you've already been introduced to variables and property expressions and have used them to a basic level, you will be using them extensively in this chapter. Then you will use event handlers to learn how you can make your packages respond to the events occurring during run time and take appropriate actions.

SSIS is quite extensible and can interface with several applications; for instance, it can act as a data source for Reporting Services reports or for an ASP.NET application. SSIS can also work with source control applications such as Visual SourceSafe or Team Foundation Server to allow you to adopt standard software development lifecycle practices. We do not cover these topics here, as these are features of the main software products rather than of SSIS; however, this chapter does cover how SSIS acts as a data source for the Reporting Services Reports feature to demonstrate that SSIS can be configured as an interface to external applications quite easily.

Let's start with the easiest and the most basic requirement of logging events and activities at run time.

Logging and Log Providers in SSIS

You can use logging in Integration Services to log information about events happening at run time for auditing and troubleshooting purposes. You can add the logs at the package level for which you want to log information.

Integration Services provides detailed levels of logging within packages with five preconfigured log providers; plus, you can create your own custom log providers. The log providers specify the format and the storage type of the log. You can log information to the text files, XML files, SQL Server Profiler, or Windows event log; or else to the sysssislog system table in an SQL Server database. Other than the Windows

event log provider, which writes log entries to the Application log of the Windows Event log on the local computer, you will need to connect to the log stores to write log entries into them. Depending on the type of log provider you choose, you may need to add a connection manager to the package. To add a log to a package, first select the log provider and then, if applicable, specify a connection manager to specify the location of the log.

SSIS containers expose their properties as log entries to enable you to log information from them. From an architecture standpoint, an Integration Services package is a container that can enclose other containers such as the Sequence container, the For Loop container, and the Foreach Loop container. These containers can further enclose tasks. And a task is enclosed inside a container called the task host container enabling the task to exhibit the container behavior. This allows you to capture log information at the package level, at the container level, or even at the task level. Once you have enabled logging at the package level, you can then configure logging at the package, container, and task levels.

Let's explore the types of log providers available in Integration Services:

SSIS Log Provider for SQL Server When you want to write the logging information into an SQL Server database so that you can query for certain events, you will select this log provider. It writes the log entries into the sysssislog system table that it creates when the package is run the first time with this log provider selected and configured. An OLE DB Connection Manager is used to specify the database to which you want it to create a sysssislog system table.

SSIS Log Provider for SQL Server Profiler In SQL Server 2008, you have the ability to link Windows performance logs with the Profiler traces so that you can match the events happening inside SQL Server against the overall performance behavior of the server. For such scenarios, when you want to analyze the execution of the package step-by-step against the performance logs, you will select this log provider to log information in a format that can be read by the SQL Server Profiler. The log file name extension must be *.trc* for it to be used with the SQL Server Profiler. This provider uses a file connection manager to write log entries to a file with .trc extension.

SSIS Log Provider for Text files This log provider adds the simplest and most used type of logs in the form of text files that can be transferred anywhere and can be read by several other applications. This log provider writes the logging information into a CSV (comma-separated value) formatted text file that you specify using a file connection manager.

SSIS Log Provider for Windows Event Log When you use this log provider, your package will log entries into Windows Application Event Log on the local computer. This is the simplest form of log provider to work with, as you do not even need to configure a connection manager.

SSIS Log Provider for XML Files When you want to write the logging information in XML format, you will use this log provider. You will then use a file connection manager to specify a file with an .xml extension into which it writes the log entries.

A few words about log providers in general: You can add more than one log of each type of log providers in your package. For example, you can add the SSIS log provider for text files twice in your package to contain different set of information in the different files. Once you've added the logs and have specified their connection managers, you can then enable logging for the tasks and containers in your package and select the events to log to one or more types of logs that you've added into your package.

By this time, you can well imagine that you can write different events to different logs with the ability to write events happening within a container to multiple logs. Also, with different levels of containers and tasks within the package, you can select relevant events to be logged based on the role or level of the container within the package. For example, you may want to log package start time and end time at the package level but may want to log much more information at the task level. By default, Integration Services provides several types of information for logging against each event or log entry. This information, called the *Integration Services log schema,* consists of the following elements:

- **Event** Name of the event or log entry.
- **Computer** Name of the computer where the event occurred.
- **Operator** Name of the user under whose context the package is being executed.
- **SourceName** Name of the container or task where the event occurred.
- **SourceID** Each task or container in the Integration Services package is assigned a unique ID, which is displayed in the ID field in the Properties window. This field refers to that unique ID of the task or container where the event occurred.
- **ExecutionID** Each time a package is executed, it is assigned a globally unique execution ID that is recorded in the logs by this element.
- **MessageText** Some events generate messages such as "Beginning of package execution" during package execution, which are logged in this element.
- **DataBytes** Specifies a byte array (BLOB) that is specific to the log entry.

You will find some additional elements available in the logs depending on the type of log, such as ID, StartTime, EndTime, and DataCode. Other features are associated

with logging functionality, such as inheritance of logging options and a facility to apply templates of logging configurations to keep consistency across the organization. These features are best understood visually while working with them, so let's do a Hands-On exercise to configure logging in a package.

Hands-On: Configuring Logging in a Package

The objective of this Hands-On exercise is to enable and configure logging in a package and study the log results after executing the package.

Method

To implement logging, we will be using the Contacting Opportunities package that you created in Chapter 4. However, to avoid version conflict and confusion among these two packages, we will create a new Integration Services project and add the Mailing Opportunities.dtsx package in to it.

Exercise (Create a New Integration Services Project)

Here you will create a new Integration Services project and add the Mailing Opportunities.dtsx package in to it.

1. Create an Integration Services project in BIDS with the following details:

Name	Contacting Opportunities with Logging
Project	C:\SSIS\Projects

2. After the project has been created, go to the Solution Explorer window and delete the Package.dtsx package.
3. Right-click the SSIS Packages node and select Add Existing Package. In the Add Copy Of Existing Package dialog box, choose SQL Server in the Package Location field if it is not already selected. In the Server field, type **localhost** and leave Windows Authentication selected in the Authentication Type field. Click the ellipsis button next to the Package Path field to open the SSIS Package window, which should display the Mailing Opportunities package under SSIS Packages folder, which you imported to MSDB in Chapter 6. However, if you skipped Chapter 6 and this package is not there, add this package from the file system instead. Select Mailing Opportunities from the list and click OK. Close the Add Copy Of Existing Package dialog box by clicking OK. You will see the Mailing Opportunities.dtsx package added under SSIS Packages node in the Solution Explorer.
4. Double-click the Mailing Opportunities.dtsx package in the Solution Explorer to load it in the Designer.

Exercise (Enable and Configure Logging)

This is the part in which you will enable logging and configure log entries at various levels in the package.

5. Right-click anywhere on the blank surface in the Control Flow tab and select Logging from the context menu. In the Containers pane of the Configure SSIS Logs dialog box, click the check box to the left of Mailing Opportunities to enable logging at the package level.
6. On the right pane of the Providers And Logs tab, click in the Provider Type field and select "SSIS log provider for SQL Server" from the list. Click Add to add this log provider. Similarly, add another log of SSIS log provider for Text Files type at the package level. You have added two different types of log providers to your package, as you will be logging the package level information to SQL Server and the Send Mail task level logging information to the text file.
7. Click in the Name field of the SSIS log provider for SQL Server log and rename it **Package level log to SQL Server**. Similarly rename the SSIS log provider for Text files log name as **SMTP task level log to Text file**.
8. Click in the Configuration field of Package level log to SQL Server and select localhost.Campaign Connection Manager.
9. Click in the Configuration field of SMTP task level log to Text file and select <New Connection…>. This will open a File Connection Manager Editor. Click in the Usage Type field and select Create File from the drop-down list. Type **C:\SSIS\Projects\Contacting Opportunities with Logging\ SMTPTaskLevel.log** in the File field and click OK. At this stage, your Configure SSIS Logs dialog box should look like the one shown in Figure 8-1.
10. Now that you have enabled logging for your package, you can select the log entries against which you want to log information. As you will be logging package-level information to the SQL Server table, select the check box on left of the Package level log to SQL Server log.
11. Go to the Details tab and select the check box provided to the left of Events column to select all the events listed below it, as shown in Figure 8-2.

    You can select the desired events from this list provided for the container or the task. Some objects in SSIS also display custom log entries here as well that are used to log special information related to that particular object. Spend some time here reading the description provided for each event.
12. After enabling logging and the log entries at package level, you can move on to configure subcontainers and tasks. Click Iterating October Opportunities. The Details tab will gray out and an alert will ask you to enable logging if you want to implement unique logging options for this container. Also note that the check in the check box on the left of this container is grayed out. This means that the logging options will be inherited from the immediate parent container—i.e.,

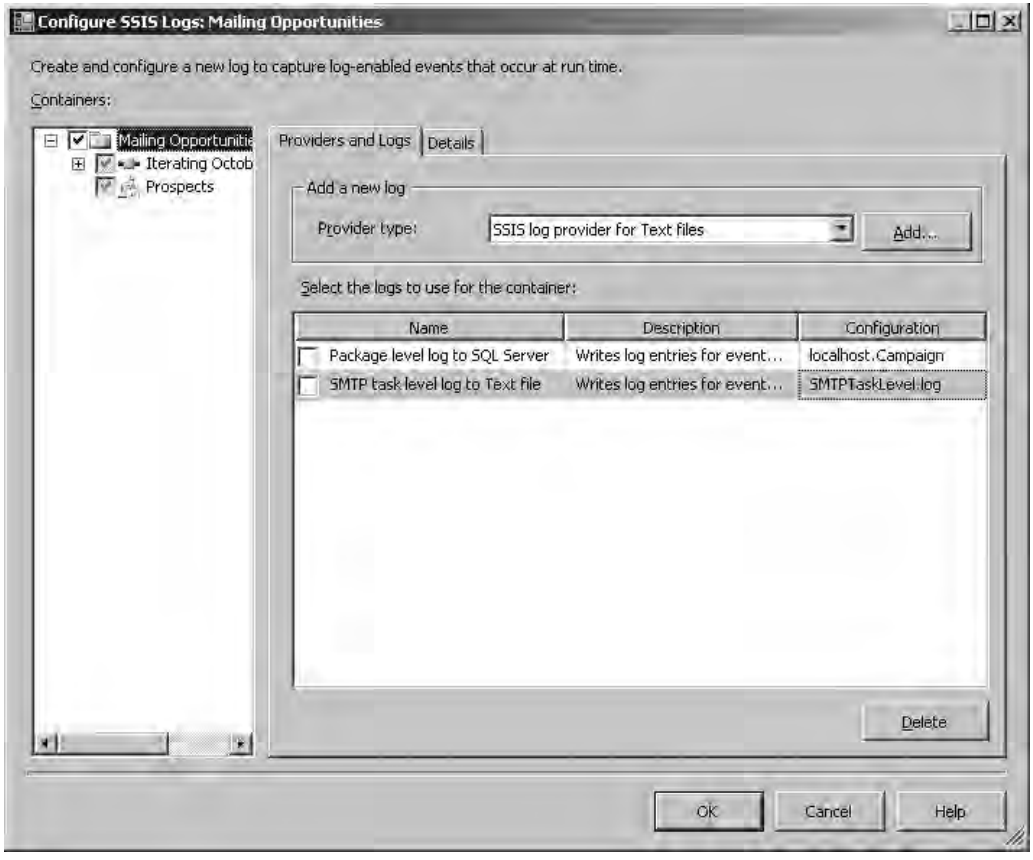


Figure 8-1 *Enabling logging in an Integration Services package*

you do not need to define logging options specifically for child containers or tasks. This makes logging easy and quick to implement. However, if you want to control the logging level for each container or task, the option is available.

Each container or task in the package can have three levels of logging modes that are defined by a property called LoggingMode: UseParentSetting, Enabled, or Disabled. You can set the logging mode on an object either from the Properties window of the selected object or by using the check boxes provided for the object in the Configure SSIS Logs dialog box. If you click the Iterating October Opportunities Container check box to clear the check mark, this disables the LoggingMode, and if you click again to apply a check mark in the check box, it will enable the LoggingMode property, which means that you can define unique logging options for this container. As we do not want to set separate logging options for this container, click again to gray out the check box to imply that the parent logging options will be used.

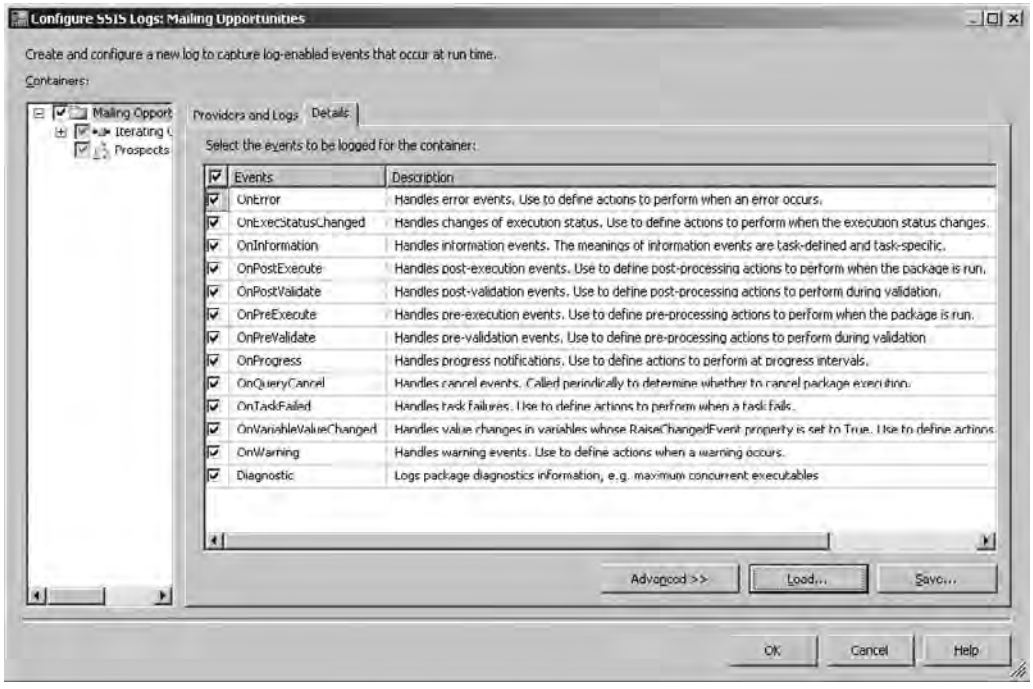


Figure 8-2 *Enabling events to log*

13. Expand the Iterating October Opportunities container and select the Mailing Opportunities task. Note that the events on the Details tab remain the same, as the logging options have been inherited from the parent. Click twice on the check box to the left of Mailing Opportunities to place a highlighted tick in the check box and change the LoggingMode to Enabled for this task. As you do that, you will notice that three new events—SendMailTaskBegin, SendMailTaskEnd, and SendMailTaskInfo—have been added to the list in the Details pane. These are the custom log entries provided by the SMTP task, as you know many of the Integration Services objects provide custom log entries for logging specific information related to their functionality only.
14. Click Advanced and you will see the information fields that will be logged when the selected events occur. These log schema fields have been described earlier in this chapter. Select the OnProgress, SendMailTaskBegin, SendMailTaskEnd, and SendMailTaskInfo events only, as shown in Figure 8-3. Note that all the fields of log schema are selected by default.
15. Go to Providers and Logs tab. Select the check box for SMTP task level log to Text file, as you want to log events to a text file for Mailing Opportunities Task. Click OK to close the Configure SSIS Logs dialog box. You've successfully configured logging for this package.

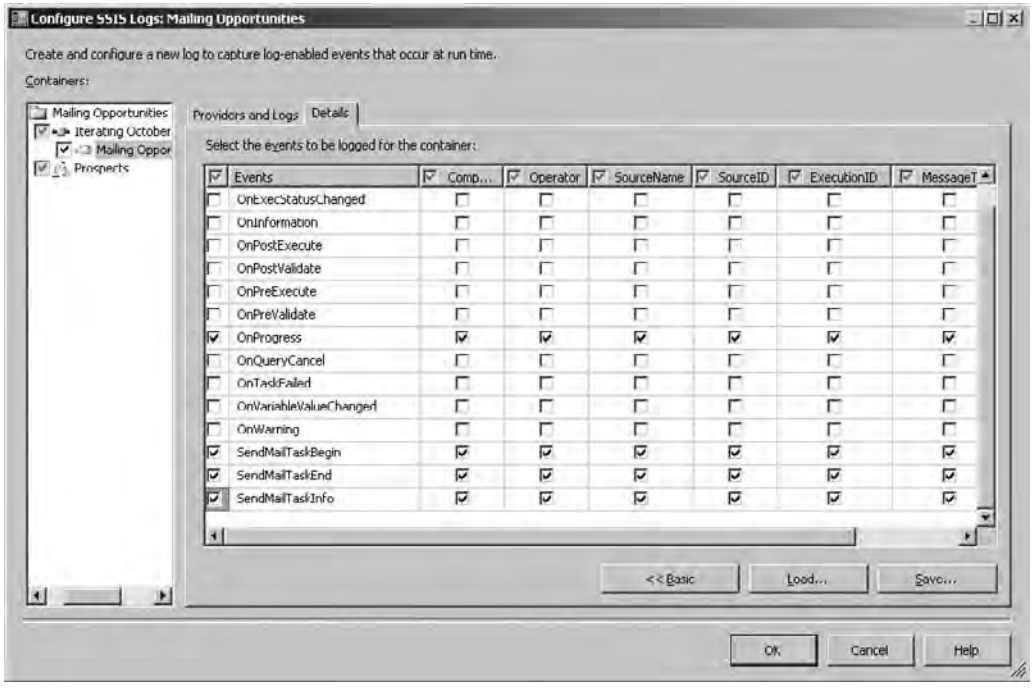


Figure 8-3 *Configuring unique log entries for the Mailing Opportunities task*

Exercise (Analyze Integration Services Logs)

Finally, let us execute the package to record logs entries and check them using SQL Server Management Studio.

16. Press the F5 key to execute the package. You will see the components changing color from yellow to green. When the package execution completes, press SHIFT-F5 to return to design mode.
17. Run SQL Server Management Studio and connect to the database engine. Run the following query in the Query pane:

    ```
    Select * from [Campaign].[dbo].[sysssislog]
    ```

    Look through the results and particularly note the OnPreExecute event and the OnPostExecute event of Iterating October Opportunities. Note the start time and end time of these events.
18. Explore to the C:\SSIS\Projects\Contacting Opportunities with Logging folder and open the SMTPTaskLevel.log file using Notepad. Scroll right to the end of file and note that the Send Mail task has been initiated and completed seven times, corresponding to the seven messages it sent out. These events occurred in between the start time and end time of the Iterating October Opportunities container.

Review

You've enabled and configured logging options in this exercise with unique logging options on the Mailing Opportunities task. You've seen how the inherited logging options work and that you do not need to set any configurations. However, if you need to save disk space and tighten up logging only for the required events, you can do so by configuring unique log entries as you have done with the Mailing Opportunities task. One interesting thing to note is that the events such as OnError and OnWarning, not captured at the task level, do not get lost; instead, they travel up and can be captured at container level. This means that you do not need to log everything at every component level. You can actually choose discrete levels in the package where you want to capture errors and warnings. This will maintain high levels of performance and keep the disks free of unwanted logs. From the best practice point of view, you should be logging heavily during development and debugging phases while reducing the logging to the minimum required level at normal production runs. Some developers log to the Windows Event log in the development environment, as this is very simple to set up and they won't have to manage log files or clear up any files.

Transactions in Integration Services Packages

Transactions enable you to implement data integrity and consistency within your packages. In DBMS world, a transaction is considered an atomic piece of work that must be completed entirely or rolled back altogether. Sometimes it becomes imperative to use transactions when working with database systems. For example, in banking systems, when somebody transfers money from his account to someone else's account, the money has to be debited and credited in a single transaction—either both operations of debiting and crediting will happen or both will be rolled back if the system is unable to complete the transaction at any stage while processing this transaction. In this way, you can be sure that the data integrity will be maintained. Transactions are required to possess the ACID properties (Atomicity, Consistency, Isolation, and Durability), without which the integrity of the transacted data cannot be guaranteed. Integration Services also allows you to use transactions within a package or among one or more packages.

Containers within an Integration Services package can be configured to use transactions by setting the TransactionOption property to Required or Supported in the Transactions section of the container's Properties window. As the tasks are enclosed within the task host containers, the tasks can also be configured to use transactions using the TransactionOption property. This property can have one of three possible values:

- **Required** The container must participate in a transaction when the TransactionOption is set to Required. That is, if the parent of this container has already started a transaction, this container will join that transaction; if no transaction exists, this container will start a new transaction.

- **Supported** The container will participate in a transaction if it already exists when the TransactionOption is set to Supported. However, if a transaction doesn't exist, it will not start a new transaction.
- **NotSupported** The container will not participate in a transaction even if a transaction exists. That is, the containers having TransactionOption set as NotSupported will neither start a transaction nor join an existing transaction.

You can use transactions with various types of possibilities and scenarios. Because Integration Services tasks support programming SQL Server, you can effectively use native transactions supported by SQL Server. But for most of the work, you will be using the Microsoft Distributed Transaction Coordinator (MSDTC) to support transactions in Integration Services. MSDTC is a Windows service that provides a transaction infrastructure across multiple computer systems or distributed computing environments.

You can configure transactions involving tasks in a container, you can involve multiple containers in a transaction, or you can have a transaction spanning over multiple packages. These packages can be running on different machines, effectively resulting in a transaction running in a distributed environment. Multiple transactions can be included in a package, and multiple packages can be run under one transaction.

How package transactions are run depend on you how you configure transactions on the containers and subcontainers and how you configure transactions when the package is run as a child package under the context of an Execute Package task. When a package is run under the context of an Execute Package task, it can inherit the transaction started by the parent package if the Execute Package task and the package are configured to join this transaction. In addition, while configuring multiple transactions within a package, you may have a transaction running a transaction. These *nested transactions* go hand in hand with the container hierarchy within a package. However, issues arise when you use nested transactions that are not related—i.e., they do not fall particularly within a single parent transaction, a condition that can cause tasks not to roll back completely as desired. While configuring transactions within your package, try to keep child transactions within the scope of a single parent transaction and, as a best practice, test thoroughly before deploying to production.

Hands-On: Maintaining Data Integrity with Transactions

You are tasked with making sure that the data is always kept in consistent state, irrespective of the data sources from which the data is coming. You are trying to identify how the data could become inconsistent; you determined that one of the reasons could be the data import process, when part of a record could not be imported. Various packages and processes are importing data and you want to determine exactly how transactions can protect your data.

Method

In this exercise, you will be simulating various scenarios of data import and will work with transaction options to maintain data integrity. You can configure transactions within a package in various ways, but to help understand how they work, you will deal with three main cases in which transactions can be used. The main steps involved in this exercise are as follows:

- Create a package and understand how data consistency can be affected with loading operations.
- Case I covers use of transaction involving multiple tasks but in a single container.
- Case II covers transactions spanning multiple containers.
- Case III covers transactions spanning multiple packages.

Exercise (Create a Simulation Package for Data Consistency Issues)

You begin this exercise by creating NewCustomer, EmailAddress, and Vehicle tables using SQL Server Management Studio. Then you move on to BIDS and create a new Integration Services project to simulate data consistency issues.

1. Open SQL Server Management Studio, connect to the database engine, and run the following queries to create three tables after opening a new query pane:

```
USE [Campaign]
GO
CREATE TABLE [dbo].[NewCustomer](
   [CustomerID] [varchar](10) NOT NULL,
   [FirstName] [varchar](50) NULL,
   [SurName] [varchar](50) NULL
) ON [PRIMARY]
Go
CREATE TABLE [dbo].[EmailAddress](
   [CustomerID] [varchar](10) NOT NULL,
   [Email] [varchar](100) NOT NULL,
   [Type] [varchar](50) NULL
) ON [PRIMARY]
Go
CREATE TABLE [dbo].[Vehicle](
   [CustomerID] [varchar](10) NULL,
   [VIN] [varchar](20) NOT NULL,
   [Series] [varchar](50) NULL,
   [Model] [varchar](50) NULL
) ON [PRIMARY]
GO
```

2. Open BIDS and create a new Integration Services project with the following details:

Name	Maintaining data Integrity with Transactions
Location	C:\SSIS\Projects

3. When the new package loads up, right-click in the Connection Managers area and choose New OLE DB Connection. In the Configure OLE DB Connection Manager dialog box, choose localhost.Campaign and click OK. You've added a connection manager to connect to the Campaign database.
4. Drop an Execute SQL Task from the Toolbox onto the Control Flow surface. Rename this task as **Loading NewCustomer**.
5. Open the Execute SQL Task Editor by double-clicking its icon. In the Connection field's drop-down list, choose localhost.Campaign.
6. With SQLSourceType set to Direct Input, click in the SQLStatement field, and then click the ellipsis button that appears in the right corner of the field to open the Enter SQL Query dialog box. Type the following SQL statement in this dialog box.

```
INSERT INTO NewCustomer (CustomerID, FirstName, SurName)
VALUES ('N501', 'Will', 'Harrison')
```

   Click OK twice to close the Editor.
7. Repeat Steps 4 to 6 to add a second Execute SQL task to your package. Rename this task **Loading EmailAddress** and edit it to add the localhost.Campaign connection manager. Close the task after assigning the following SQL statement to it:

```
INSERT INTO EmailAddress (CustomerID, Email, Type) VALUES
('N501', 'wharrison@AffordingIT.co.uk', 'Work')
```

   Join the Loading NewCustomer task with this task using an on-success precedence constraint.
8. Repeat steps 4 to 6 to add a third Execute SQL task to your package. Rename this task **Loading Vehicle** and edit it to add the localhost.Campaign connection manager. Close the task after assigning the following SQL statement to it:

```
INSERT INTO Vehicle (CustomerID, VIN, Series, Model) VALUES
('N501', 'UV123WX456YZ789', 'X11 Series', 'Saloon')
```

   Join the Loading EmailAddress task with this third task using an on-success precedence constraint. Your package should look like the one shown in Figure 8-4.
9. Execute this package by pressing the F5 key. When all the tasks have changed to green and the package has completed execution, press SHIFT-F5 to return to design mode.

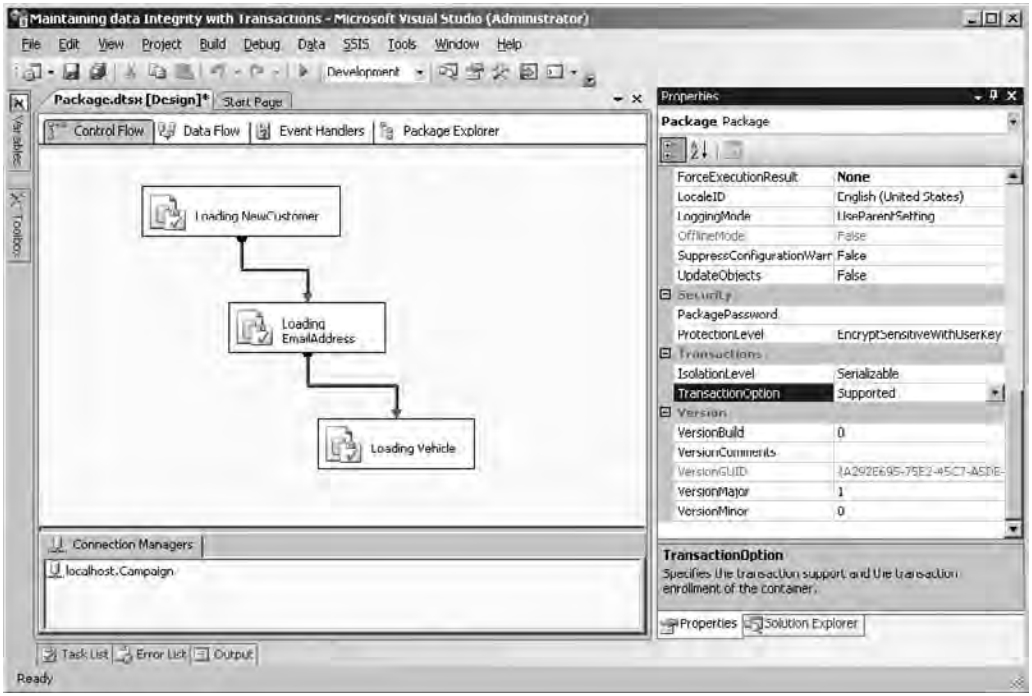


Figure 8-4 *Creating a simulation package for data consistency checks*

10. Switch to the SQL Server Management Studio and run the following query:

```
SELECT n.[CustomerID], [FirstName], [SurName], [Email],
[Type], [VIN], [Series], [Model]
FROM [Campaign].[dbo].[NewCustomer] n LEFT OUTER JOIN
[Campaign].[dbo].[EmailAddress] e
ON n.CustomerID = e.CustomerID
LEFT OUTER JOIN [Campaign].[dbo].[Vehicle] v
ON n.CustomerID = v.CustomerID
```

You will see the result contains customer details, the customer's e-mail address, and the vehicle details, all as you wanted (see Figure 8-5), irrespective to the fact that the data is stored in different tables.

11. Note the TransactionOption property of the package in Figure 8-4, which is set at the default value of Supported. This property has a Supported value for all the three tasks as well. Only the containers with TransactionOption set as Required can start a transaction, so no container started a transaction while this package was executed. So far, so good. Now, let's see what happens if a task fails and doesn't import a particular row.

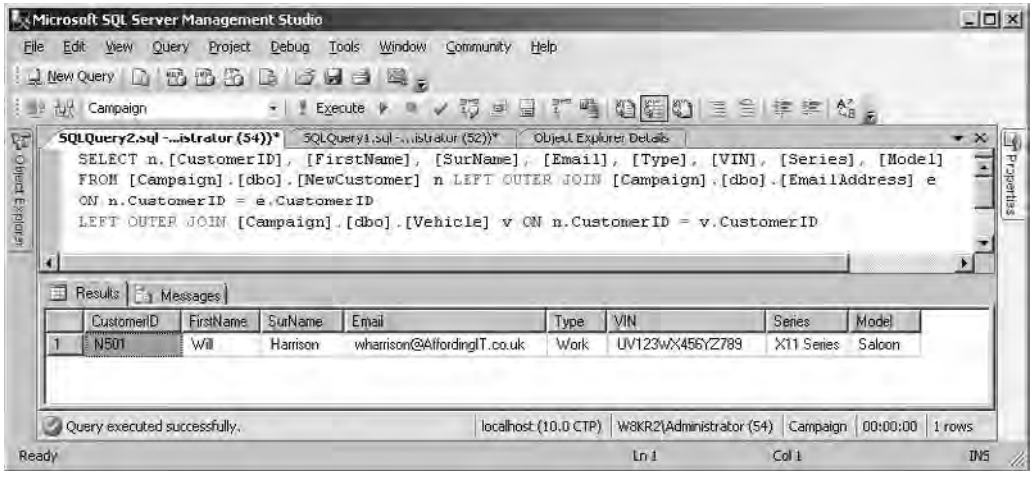


Figure 8-5 *The data consistency you want to see in your database*

Data uploading can fail for a particular record for various reasons. The most common is a data type mismatch or a constraint on the column. If you go to the Campaign database, expand the Vehicle table, and then look in the Columns node in the Object Explorer in SQL Server Management Studio, you will see the column properties of the Vehicle table. Note that the VIN field is a mandatory (not null) field and must have a known value. If no data is received for this field during the import process, that row will not be inserted.

12. Before you proceed, let's clean up our tables. Run the following queries in the Query pane to delete all the data from the existing tables:

```
DELETE [Campaign].[dbo].[NewCustomer]
DELETE [Campaign].[dbo].[EmailAddress]
DELETE [Campaign].[dbo].[Vehicle]
```

13. Switch to BIDS and modify the SQL statement in the Loading Vehicle task by removing the VIN information from the SQL statement; your statement should look as follows:

```
INSERT INTO Vehicle (CustomerID, Series, Model) VALUES
('N501', 'X11 Series', 'Saloon')
```

14. After you've made the change, press F5 to execute the package again. Keep in mind that you've still not implemented a transaction in the package. This time, you will see that the first two tasks execute successfully while the Loading Vehicle task fails and turns red. Stop debugging the package.

15. Switch to SQL Server Management Studio and execute the query written in Step 10 to see the results this time. You will see that the three fields—VIN, Series, and Model—have NULL values. You have discovered how your data can become inconsistent during the loading process.
16. Run the delete SQL statements written in Step 12 to cleanse this data from all the tables.

Exercise (Case I: Avoiding Inconsistency in a Single Container)

In real life, you either commit all the data from all the tasks in the tables or roll back the failing rows and throw those rows out and deal with them separately. In the following steps, you will see how you can roll back the data by using transactions.

17. Click anywhere on the blank surface of the Control Flow panel and press F4 to open the Properties window. Scroll down the Properties window and locate the Transactions section. Change the TransactionOption value from Supported to Required.
18. Press F5 to execute the package. You will again see that the first two tasks complete successfully whereas the Loading Vehicle task fails as expected. Stop debugging the package.
19. Switch to SQL Server Management Studio and execute the query written in Step 10 in the preceding sequence to see the results. This time, you will see no data at all in the output query. Setting the TransactionOption to Required resulted in the use of a transaction under which all the three tasks were executed. As the third task failed, the data loaded by first two tasks was rolled back.

Exercise (Case II: Transaction Spanning over Multiple Containers)

The preceding exercise demonstrated how you can use a transaction to avoid inconsistency in data during loading operation. This was quite simple, involving only one container in the package. What if a data-loading package uses multiple containers? Let's see how that can be handled.

20. Drop two Sequence Containers on the Control Flow surface from the Toolbox. Delete the precedence constraint connecting Loading EmailAddress with Loading Vehicle.
21. Select Loading NewCustomer and Loading EmailAddress along with the precedence constraint either by using the mouse or by pressing and holding the CTRL key on the keyboard while clicking the tasks one by one. With these tasks selected, drag and drop them inside the first Sequence container.

22. Drag and drop the Loading Vehicle task inside Sequence Container 1. Drag the green arrow from the Sequence Container and drop it on Sequence Container 1 to join these containers so that the tables are loaded in sequence. (Note that this step is not required for running of the package, though.) Your package will look like the one shown in Figure 8-6.
23. Your package is now using two sequence containers to load the data. Verify that the two new containers have their TransactionOption set to default value of Supported and the Package container's TransactionOption property still has the Required value set previously.
24. Press F5 to run the package. You will see that the first container with the two tasks in it successfully executes and turns green. However, the second sequence container with the Loading Vehicle task in it fails and turns red. Press SHIFT-F5 to switch back to design mode.
25. Switch to SQL Server Management Studio and run the SQL statement you wrote in Step 10 of the preceding sequence to see what's been loaded in the database. You will see that no data for the NewCustomer and EmailAddress tables has been added, even though the loading tasks were successful. This is

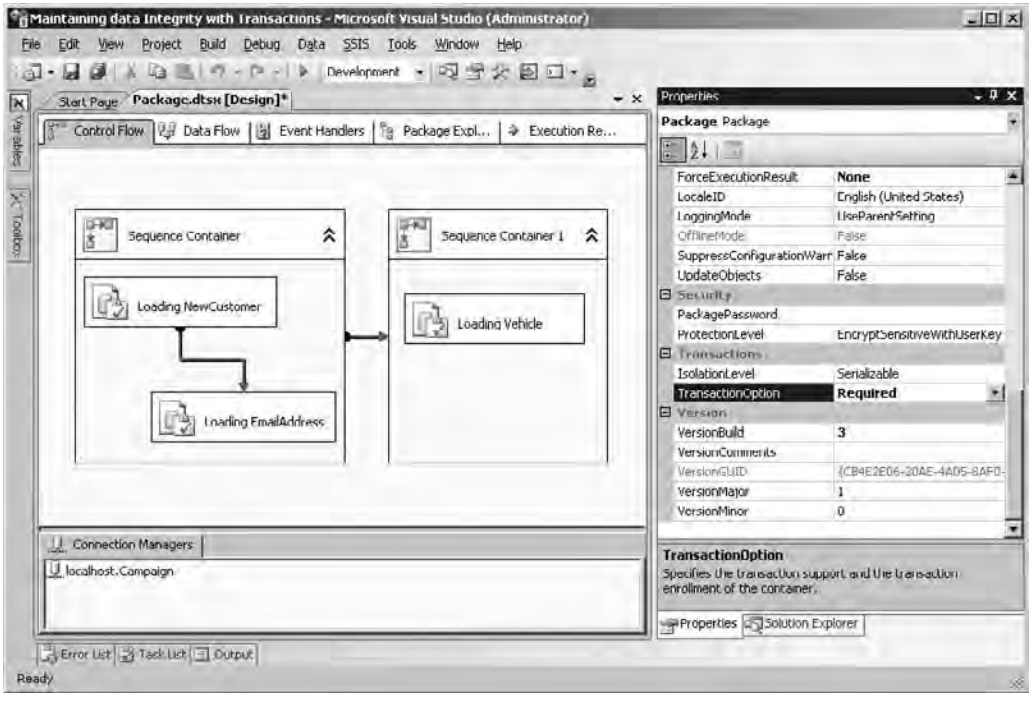


Figure 8-6 *Package consisting of multiple containers*

because both the containers were running under one transaction that was started by the package, and when one of the tasks in a container fails, the transaction rolls back all the work done by previous tasks. One lesson to learn from this exercise is that the parent container, which is the package in this case, must have its TransactionOption property set to Required to start a transaction, and the child containers need to have at least the Supported attribute for this property.

Exercise (Case III: Transaction Spanning over Multiple Packages)

In the last part of this exercise, you will use a transaction to roll back the inconsistent data when your loading process uses multiple packages. When you have multiple packages to process, you use the Execute Package task to embed them inside a single package to run them. The Execute Package task is basically a wrapper task that enables a package to be used inside another package. The Execute Package task is covered in Chapter 5.

26. Right-click the SSIS Packages node in the Solution Explorer window and choose New SSIS Package from the context menu. You will see that the new package has been added with the default name of Package1.dtsx and the screen is switched to the new package. Note that the Designer shows these two packages as tabs.
27. Go to Package.dtsx, right-click the localhost.Campaign Connection Manager, and choose Copy. Switch back to Package1.dtsx and paste this connection manager in the Connection Managers area.
28. Again go to Package.dtsx and cut the Sequence Container 1 with Loading Vehicle Task, return to Package1.dtsx, and paste this container on the Control Flow. You will see a validation error about the connection manager on the Loading Vehicle task. This is because the ID for the localhost.Campaign Connection Manager has been changed.
29. Double-click the Loading Vehicle task icon to open the editor. In the Connection field, choose localhost.Campaign Connection Manager from the drop-down list and click OK. You've divided the first package into two separate packages. To run these two packages as a single job, you need to create a new package and call these two packages using the Package Execute task.
30. Right-click the SSIS Packages node in the Solution Explorer window and choose New SSIS Package from the context menu. When the new blank package is loaded, drop two Execute Package tasks on the Control Flow surface.
31. Rename the first Execute Package task **Package** and the second task **Package1**. Join Package to Package1 using an on-success precedence constraint.
32. Double-click the Package icon to open the editor. Go to the Package page and change the Location field value to File System.

33. Click in the Connection field and then click the drop-down arrow and choose <New Connection…>. In the File Connection Manager Editor's File field, type **C:\SSIS\Projects\Maintaining data Integrity with Transactions\Package.dtsx** and click OK. You will see Package.dtsx displayed in the Connection field. Click OK to close the Execute Package Task Editor.
34. As in the last two steps, open the editor for the Package1 task, change the Location to File System, and add a file connection manager in the Connection field pointing to C:\SSIS\Projects\Maintaining data Integrity with Transactions\Package1.dtsx as the existing file. Close the Execute Package Task Editor after making these changes.
35. Click anywhere on the blank surface of the Control Flow panel and press F4 to open the Properties window for the package. Scroll down and locate the Transactions section and set the TransactionOption property to Required. This will run both the Execute Package tasks and hence the child packages in the context of a single transaction. However, before proceeding any further, verify that the TransactionOption is set to the default value on Package and Package1 tasks and on the Package1.dtsx package. The Package.dtsx will have this property set to Required, which is okay, as this will also enable it to join the transaction started by Package2 .dtsx. At this time, your package will look like the one shown in Figure 8-7.

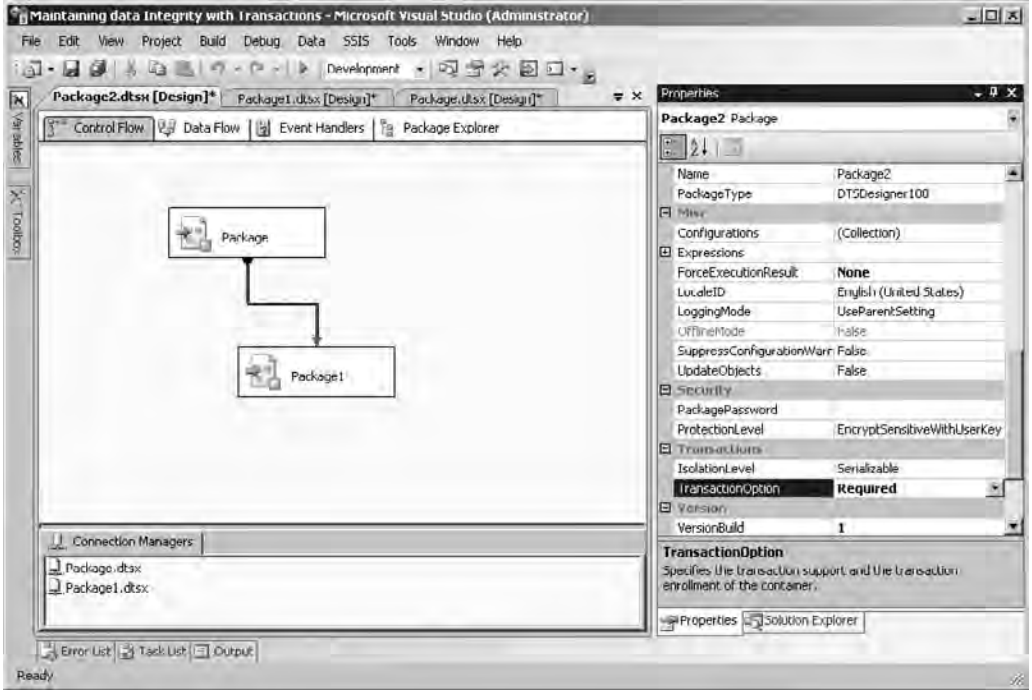


Figure 8-7 *Calling multiple packages using the Execute Package tasks*

36. Go to the Solution Explorer window, right-click Package2.dtsx, and then select Execute Package from the context menu. You will see that the Package.dtsx will execute successfully and then Package1.dtsx will execute, but it fails, and the components will turn red.
37. Switch to SQL Server Management Studio and run the command you created in Step 10 in the first sequence of steps to see the results. You will see that still no record has been added to the tables, despite the fact that Package.dtsx executed successfully. This is because both the packages were running under one transaction. And when the Loading Vehicle task failed in the Package1.dtsx package, the transaction rolled back not only all the tasks in this package but also the tasks in the other package, Package.dtsx.

Review

You've seen how you can use a transaction to combine various tasks and containers and even the packages to behave as a single unit and create atomicity among them that will commit or roll back as a unit. You've worked with the Sequence container to combine set of tasks as a logical unit and have learned a new trick of copying and pasting tasks among packages to increase productivity.

While all the preceding is useful when you want to use distributed transactions, you cannot use the distributed transactions in all situations. Sometimes you may need to use Native Transaction support. Native transactions are native to the RDBMS that is used, for instance. A simple case could be that you create and populate a temporary table in one task and want to use it later in another task. This kind of requirement cannot be met using the distributed transaction support. In SSIS, when you configure a task you specify a connection manager on each task. So, when a task is run, a connection is opened specifically for that task, and later this connection is closed when the defined operation on the task has been performed. The closure of a connection doesn't help to perform native transactions that need the same connection to be retained across all the tasks involved. SSIS provides you a Boolean property on the Connection Manager named intuitively the RetainSameConnection property that allows you to keep a connection open across all the involved tasks. To use this property, click the Connection Manager, then set the RetainSameConnection to True, and then use this connection manager in all the tasks that participate in native transaction process. One of the main benefits of using a native transaction is that you can build a logic-based commit or rollback of the transaction that is otherwise not possible with distributed transactions, which can commit or roll back only on success or failure of the tasks involved.

Restarting Packages with Checkpoints

If you're like most other information analysts and update your data warehouse every night, this feature will be of much interest to you. After having set up logging for your packages, every morning you'd be checking the logs for the last night's update process to see how the update went. You usually expect that the update process has been successful, but what if the update process has failed? You will have to rerun your package during the daytime—and I know you wouldn't be happy about this, because doing this work during business hours involves some serious implications. Your users will not get the latest updates and will experience poor performance of the involved database servers while you rerun the update process. If you've worked with DTS 2000 packages, you know that DTS 2000 doesn't support restating a package from the point of failure. You have to rerun the package from the start or manually run the tasks individually, which is quite involving and sometimes impossible to do. This is where Integration Services comes to the rescue by providing improved functionality of restarting a package.

By using checkpoints with Integration Services packages, you can restart your failed packages from the point of failure and can save the work that has completed successfully. Integration Services writes all the information that is required to restart a failed package in a *checkpoint* file. This file is created whenever you run a package the first time after a successful completion, and it is deleted when the package successfully completes. However, if an Integration Services package fails and is configured to use checkpoints, the checkpoint file is not deleted; instead, it is updated with information that is required to rerun the package from that point. When you rerun your package, Integration Services checks two things before executing the package: whether the package is configured to use checkpoints and whether the checkpoint file exists—i.e., whether the package failed while executing last time. If it finds that the package configured to use checkpoints has actually failed the last time it was run—i.e., the checkpoint file exists, it then reads the checkpoint file associated with the package, gets the required information from the file, and restarts the package from the point of failure.

The checkpoint file contains all the necessary information for a package to restart at the point of failure such as the execution results of all the completed units of work, the current values of variables involved, and package configuration information.

You decide the key positions in your package that would be good candidates for the point of restart and can be written as checkpoints in the file. For example, you would definitely designate a checkpoint immediately after the task that loads a large data set or downloads multiple large files from an FTP site. In case of failure of the package after successfully downloading files or completing loading the data set, the package will be restarted after these tasks, as the checkpoint defines the starting place. As mentioned earlier, the checkpoint file also contains the package configuration information—i.e., the information about the configurations under which the package was running.

This avoids reloading of package configurations, as this is read from the checkpoint file and hence maintains the original configurations into which the package was running at the time of failure.

To enable your package to record checkpoints information, you set the following properties at the package level:

- **CheckpointUsage** You can access this property in the Checkpoints section of the package Properties window. This property can have one of three values: Never, Always, or IfExists. The default value is Never, which means the checkpoints are not enabled and no checkpoint file will be created; hence, the package will always start processing from the beginning whenever it is executed. The second value is Always, which, if selected, will make the package always use a checkpoint file. If the package has failed in the previous execution and you've somehow deleted or lost the checkpoint file, the package will fail to execute. The third possible value is IfExists, which, when selected, makes the package use a checkpoint file if it exists and start the package from the point of failure in the previous execution. You can reuse a checkpoint file over and over for the same package. However, if the checkpoint file doesn't exist, the package will always start from the beginning. The checkpoint file is specific to a package. Before executing a package, SSIS checks if the PackageID in the checkpoint file is the same as that of the package. If there is a mismatch, SSIS won't execute the package.
- **SaveCheckpoints** After enabling your package to use checkpoints, you can set this property to True to indicate that checkpoints should be saved.
- **CheckpointFileName** Using this property, you can specify the path and the file into which you would like to save checkpoints.

Along with these properties, you also need to set the FailPackageOnFailure property, available in the Execution section in Properties window on the package and the containers, to True to specify that the package will fail when a failure occurs. This property helps in setting the checkpoints on the tasks that you want to make as points of restart. If you do not set this property on any task or container in the package, the checkpoint file will not include any information for the containers on failure and will restart the package from the beginning. It is interesting to note the following points concerning the smallest unit that can be restarted:

- The smallest unit that can be restarted is a task.
- The Data Flow task, which is a special task in Integration Services enclosing the data flow engine, can consist of several data flow transformations. This task is considered similar to any other Control Flow task as far as checkpoints are

concerned and cannot be started from halfway where it failed. If you have massive pipeline operations in your package and you're concerned about rerunning packages, it is better that you divide up the data transformations work between multiple Data Flow tasks.

- The Foreach Loop Container is also considered an atomic unit of work that will either commit or restart completely to iterate over all the values provided by the enumerator used.
- When used with For Loop Container, the checkpoint file will save the last value of the variable and hence will restart from the same point where it left off.

The use of an atomic unit of work actually calls for a discussion on transactions and checkpoints, as transactions convert the tasks and the packages involved into an atomic unit of work. Let's understand the checkpoints and their operation within the scope of a transaction in the following Hands-On exercise.

Hands-On: Restarting a Failed Package Using Checkpoints

In this exercise, you will simulate a package failure and configure your package with checkpoints to restart it from the point of failure.

Method

You will use the package you developed earlier in the last exercise and apply checkpoint configurations to it. In the second step, you will use transactions over the package to see its behavior.

Exercise (Apply Checkpoint Configurations to Your Package)

In the first part of this Hands-on, you configure the Integration Services package to use the checkpoints and execute the package to see it execution behavior.

1. Open BIDS and create a new Integration Services Project with the following details:

Name	Restarting failed package
Location	C:\SSIS\Projects

2. When a blank project is created, delete the Package.dtsx package under SSIS Packages node in the Solution Explorer window. Then, right-click the SSIS Packages node and choose Add Existing Package from the context menu.

3. In the Add Copy Of Existing Package dialog box, select Package Location as the File System. In the Package Path field, type **C:\SSIS\Projects\Maintaining data Integrity with Transactions\Package.dtsx** and click OK to add this package. Once the package has been added, open it in the Designer.
4. Drop an Execute SQL task from the Toolbox on to the Designer surface outside the Sequence container and rename this task **Loading Vehicle**. Double-click the task icon to open the editor. In the General page's Connection field, choose the Add localhost.Campaign Connection Manager and type the following SQL statement in the SQLStatement field:

   ```
   INSERT INTO Vehicle (CustomerID, Series, Model) VALUES
   ('N501', 'X11 Series', 'Saloon')
   ```

   You already know that this SQL statement is without the mandatory VIN field; hence it will fail the Loading Vehicle task. Join the Sequence Container with the Loading Vehicle task using an on-success precedence constraint.
5. Click anywhere on the blank surface of the Designer and press F4 to open the Properties of the package. First, make sure that the package is not configured to use transactions. Scroll down and locate the TransactionOption property, and change its value to Supported.
6. Scroll up in the Properties window and locate the Checkpoints section. Specify the following settings in this section:

SaveCheckpoints	True
CheckpointUsage	IfExists
CheckPointFileName	C:\SSIS\Projects\Restarting failed package\checkpoints.chk

7. Because we want to include the restart information of the Loading Vehicle task in the checkpoints file, click the Loading Vehicle task on the Designer surface. You will see that the context of Properties window changes to show the properties of the Loading Vehicle task. Locate the FailPackageOnFailure property in the Execution section and change its value to True.
8. Press F5 to execute the package. You already know the result of the execution. The Sequence Container and the two Execute SQL tasks in it successfully execute and turn green, but the Loading Vehicle task fails and shows up in red. Press shift-F5 to switch back to designer mode.
9. Let's see what has happened in the background while the package was executing. Open SQL Server Management Studio and run the following query to see the records imported into the database:

   ```
   SELECT n.[CustomerID], [FirstName], [SurName], [Email],
   [Type], [VIN], [Series], [Model]
   FROM [Campaign].[dbo].[NewCustomer] n LEFT OUTER JOIN
   [Campaign].[dbo].[EmailAddress] e
   ```

```
ON n.CustomerID = e.CustomerID
LEFT OUTER JOIN [Campaign].[dbo].[Vehicle] v
ON n.CustomerID = v.CustomerID
```

You will see that the customer information and its e-mail information have been loaded while the vehicle information fields have null values.

Using Windows Explorer, navigate to the C:\SSIS\Projects\Restarting failed package folder and note that the checkpoints.chk file has been created. Open this XML formatted file and note that it contains information about the failure of the package and the cylinder involved in the failure.

10. Change the SQL statement of the Loading Vehicle task to include the VIN information with the following query:

```
INSERT INTO Vehicle (CustomerID, VIN, Series, Model) VALUES
('N501', 'UV123WX456YZ789', 'X11 Series', 'Saloon')
```

11. Again execute the package. This time you will see that only the Loading Vehicle task is executed and the earlier two tasks and the Sequence container did not run at all (see Figure 8-8). This is because the package reads the checkpoint file before executing and finds the information about where to start executing. Press SHIFT-F5 to switch back to design mode.

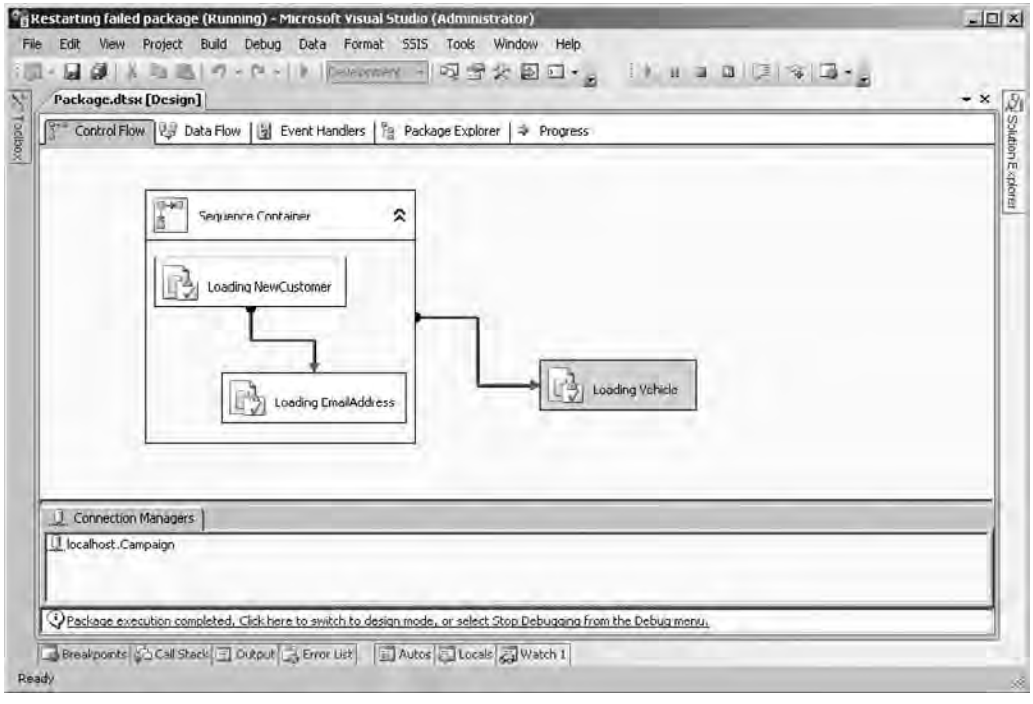


Figure 8-8 *Restarting package with checkpoints*

12. Explore to the C:\SSIS\Projects\Restarting failed package folder and note that the checkpoints.chk file does not exist.
13. Switch to SQL Server Management Studio and run the script specified in Step 9 to see the result set. You will see one record containing customer, e-mail, and vehicle information. Run the following queries to clear the tables:

```
DELETE [Campaign].[dbo].[NewCustomer]
DELETE [Campaign].[dbo].[EmailAddress]
DELETE [Campaign].[dbo].[Vehicle]
```

Exercise (Effect of Transaction on Checkpoints)

To set transactions on this package we need to set the TransactionOption value to Required. So, let's do it.

14. Click anywhere on the blank surface of the Control Flow Panel and press F4 to open the Properties window. Scroll down and locate the TransactionOption property in the Transactions section. Set it to the Required value so that it starts a transaction. But SSIS doesn't allow you to do this and throws an error as shown in Figure 8-9.

This behavior is different than Integration Services 2005, in which you could use transactions and checkpoints in the same package and Integration Services left proper usage and management of both of them to you. In that case the transactions roll back the information of the checkpoint file and cause that package to execute all over again. This is actually applicable to containers in simple packages also. But there is a potential for error or misbehavior when you are using Integration Services 2005 with checkpoints

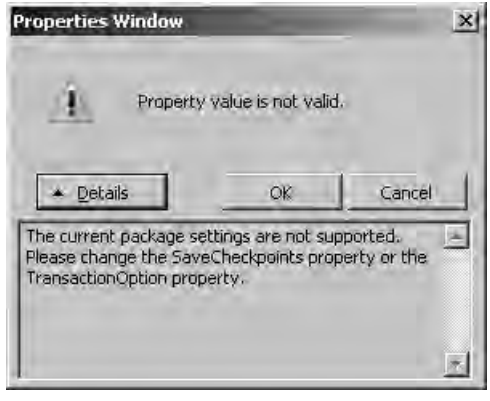


Figure 8-9 *Error thrown while trying to use transactions alongside checkpoints*

and transactions in a complex package; that is, if your package consists of a complex container hierarchy and a subcontainer commits before the parent container fails, the subcontainers do not get rolled back and also do not get recorded in the checkpoint file. This causes those subcontainers to be executed again when the parent container is restarted. Similarly, the Foreach Loop container does not record any information in the checkpoint file about the iterations it may have already done before failing and gets executed all over again when restarted. So, when you're planning to use checkpoints alongside the transactions, use caution and test thoroughly. Integration Services 2008 R2, by contrast, stops you doing that altogether due to the complexity and risk involved, and you can't use transactions and checkpoints in your packages at the same time.

Review

You've seen in this exercise that the checkpoints can help you restart a package precisely from the task where the package failed. You also understand that you need to be careful while using transactions and checkpoints on packages with complex container hierarchies in Integration Services 2005. On the other hand, Integration Services 2008 R2 doesn't allow you to implement checkpoints and transactions at the same time.

Expressions and Variables

You learned about variables and property expressions in Chapter 3 and have used them in various Hands-On exercises in subsequent chapters. With DTS 2000, use of variables was considered an advanced feature that allowed you to add some dynamic behavior to your packages. However, use of variables in Integration Services is made easier and has been tied into SSIS package design so much that the packages developed without using variables are reduced to ad hoc data operations, most of which can be done using the SQL Server Import and Export Wizard. On the other hand, use of property expressions is a new feature in Integration Services that provides an ability to set values for component properties dynamically using variables that are updated at run time by other tasks. Property Expressions allow you to evaluate values generated at run time by other tasks and use the evaluated values to update properties exposed by the concerned task at run time. This is quite a powerful feature, as it allows you to read and evaluate the values that exist only at run time and modify the property or behavior of other tasks in the package.

Though you've used variables and expressions in the Hands-On exercises earlier, here you will do another exercise that uses variables and particularly property expressions extensively to update properties of the send mail task to generate personalized mails.

Hands-On: Extending the Contacting Opportunities Package with Property Expressions

In Chapter 4, you created an Integration Services project called Contacting Opportunities that sends mail to persons who raised a query in October 2009. However, during the package development, you used static values in the To field, Subject field, and Message field. You are now to extend the package so that these values are read from the table and evaluated for each person to create personalized messages.

Method

You will add the Mailing Opportunities package from the Contacting Opportunities package to a new package to keep the package separate from a learning point of view. The Mailing Opportunities package sends mails to six persons who made enquiries in October 2009 using static values. To make the package dynamic, you will update e-mail addresses in the Prospects table with your e-mail address so that you can see the messages that are being generated and sent out by this package. Then you will create variables that will capture values from the Prospects table to pass on to property expressions at run time. Finally, you will create property expressions and will also learn about the DelayValidation property toward the end of this exercise.

Exercise (Build Property Expressions for Mailing Opportunities Package)

Having understood the method, follow the steps below to work with property expressions.

1. Open SQL Server Management Studio and run the following query on the local server.

   ```
   UPDATE [Campaign].[dbo].[Prospects]
   SET email = 'youremailaddress'
   WHERE ENQUIRYDATE BETWEEN '2009/10/01' AND '2009/10/31'
   ```

   Replace *youremailaddress* in the preceding query with your e-mail address and then execute this query to update the six records.

2. Open BIDS and create a new Integration Services project with the following details:

Name	Contacting Opportunities with Property Expressions
Location	C:\SSIS\Projects

3. When the blank project is created, delete the Package.dtsx package in the SSIS Packages node and then right-click the SSIS Packages node and choose Add Existing Package from the context menu.
4. In the Add Copy Of Existing Package dialog box, select Package Location as the File System from the drop-down list. Type **C:\SSIS\Projects\Contacting Opportunities\Mailing Opportunities.dtsx** in the Package Path field and click OK to add this package. Once the package has been added, double-click the Mailing Opportunities.dtsx package in the SSIS Packages folder to open it on the Designer.
5. Double-click the Iterating October Opportunities Foreach Loop container to open the editor. This package enumerates over the User::Opportunities variable. Earlier, no value was been picked up and used in the package, so let's fill the gap now. Go to the Variable Mappings page and click in the Variable column and then click the down arrow and choose <New Variable> from the drop-down list.
6. Leave Mailing Opportunities selected in the Container field in the Add Variable pop-up dialog box. Type Title in the Name field and leave the Namespace as User and Value Type set to String. Variables are case-sensitive, so type the name all in lowercase to avoid any issues later on. Click OK to add this variable. You will see that User::title has been added in the Variable column and assigned an Index value of 0.
7. Much as you did in Step 6, create four more variables and make sure they get the Index values as per the following table:

Variable	Index	Value Type
fname	1	String
lname	2	String
email	3	String
enquiry_date	4	DateTime

For the variable enquiry_date, you will need to assign a placeholder value in the format *09/09/2009*, as this type of variable cannot be defined without a value. The Variable Mappings settings should look similar to Figure 8-10. Click OK to close the editor.

8. Now that you've mapped the values from the table to the variables, it is time to make use of them in the package. Double-click the Mailing Opportunities Send Mail task icon to open the editor. Go to the Mail page where you will be using Property Expressions to derive and modify values assigned to various fields at run time. To be absolutely sure that the e-mail address in the To address of the e-mails is being read from the table and not from the value you've typed directly in the field, type **test@test.com** in the To field.

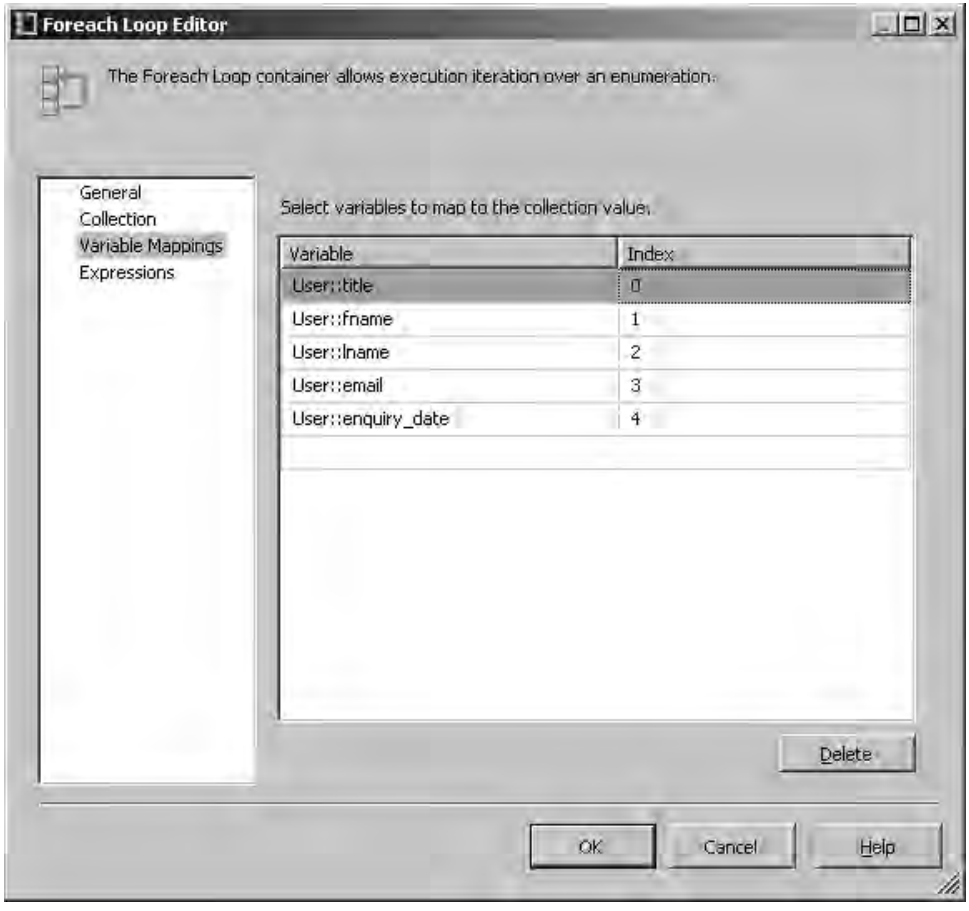


Figure 8-10 *Adding variables to the package*

9. Go to the Expressions page and click in the Expressions field. Then click the ellipsis button to open the Property Expressions Editor.
10. Click in the Property column and then click the drop-down arrow and select the ToLine property. Now you can either type an expression directly in the Expression field to evaluate a property or use an Expression Builder to build an expression by clicking the ellipsis button next to the Expression field. For now, click the ellipsis button to open the Expression Builder. In the top-left pane, expand Variables, locate the User::email variable, and drag and drop it into the Expression box. Click OK to return to the Property Expression Editor.
11. Similarly, add the Subject property in the next row in the Property column and then click the ellipsis button. First type **"Your enquiry dated " +** in the

Expression box and then expand the Type Casts node in the top-right pane. Locate the (DT_WSTR, <<*length*>>) type cast and add it after the plus sign in the expression. Replace <<*length*>> with 20 and then drag the User:: enquiry_date variable from the top-left Variables node into the expression. Click the Evaluate Expression button to check whether the expression has been built properly. Refer to Figure 8-11 to see how it should look. Click OK.

12. Again, click in the next row in the Property column, then click the drop-down arrow, and this time select the MessageSource property. Click the ellipsis button to open the Expression Builder. Build the following expression in the Expression box:

```
"Dear " + @[User::title] + @[User::lname] + "," + "

Thank you for your enquiry. One of our sales representatives will
be in touch with you. In the meantime, please go to our web site
for more information on our products. Thank you very much for
showing interest.

Kind regards,
Sales Support Team"
```

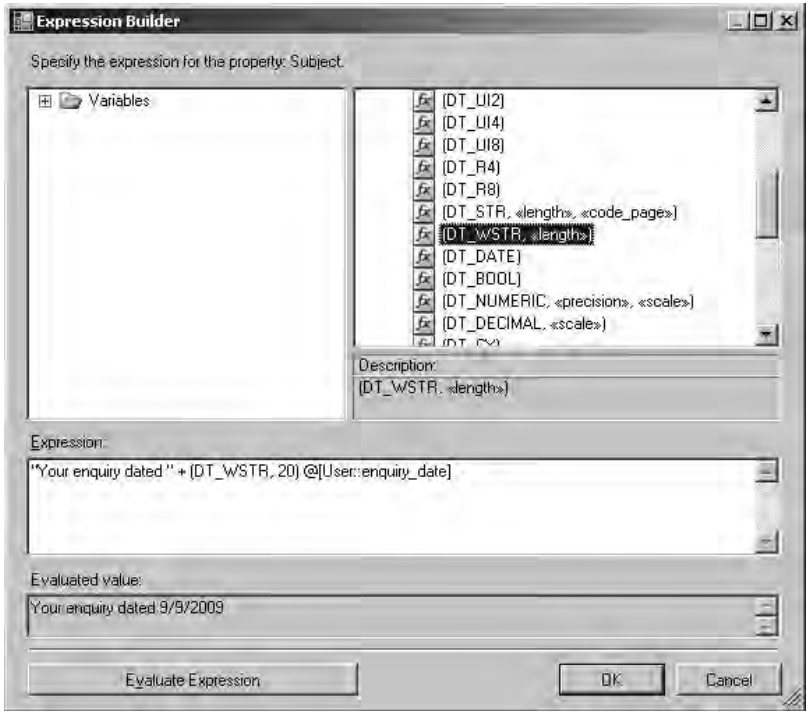


Figure 8-11 *Building an expression for the Subject property*

When you're done, click OK to close the Expression Builder. Your Property Expression Editor should now look as shown in Figure 8-12. Click OK twice to close the editor and the Mailing Opportunities Task Editor.

13. Press the F5 key to execute the package—but the package fails to execute and a package validation error appears, as shown in Figure 8-13.
14. The error message "No recipient is specified" indicates that during the validation process of execution, Integration Services found that no value was assigned to the To property. If you open the Mailing Opportunities Task Editor, you will find that the earlier assigned direct value of test@test.com no longer exists in the To field and the To field is blank. This is because when the package execution starts, validation happens before any other operation. At the validation time, Integration Services knows that the To property has to be populated from the ToLine property expression and hence ignores any direct value assigned in the field; it failed to find a value because the property expression further needed a value from User::email variable, which was not available because the package had not been executed yet. This type of error is quite normal when using Property Expressions that use variables at run time that are not yet available. To overcome this situation, Integration Services provides a facility to delay validation for such components of the package until actual run time for the component. Click the Mailing Opportunities task and press F4 to open the Properties window. Locate the DelayValidation property in the Execution section and change its value to True.

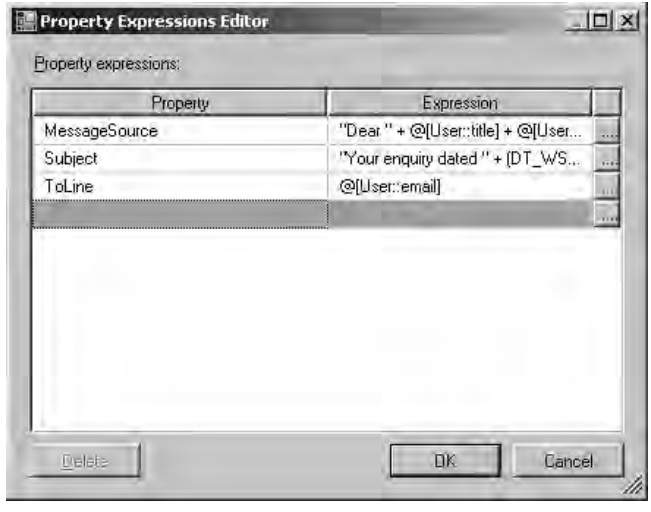


Figure 8-12 *Property expressions built for multiple properties*

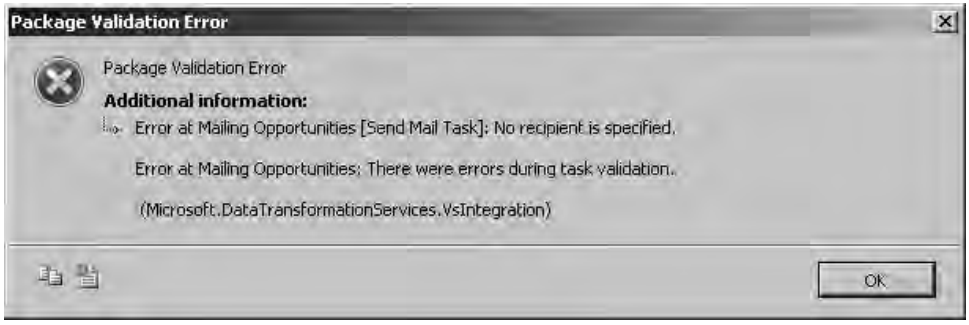


Figure 8-13 *Validation error due to ToLine property having blank value*

15. Press F5 to execute the package, and this time the package will run and complete successfully. If you check your mailbox after some time, you will see the six personalized messages.

Review

You've used property expressions in this exercise to evaluate properties from static values and also variables that in turn get their value only at run time. This proves how you can dynamically modify a package at run time based on the values of other attributes in the package. You also understood the use of the DelayValidation property, which is handy to use when you are developing a package that uses lots of property expressions. One more important point to learn is that the Property Expressions can be mapped to properties exposed through the Control Flow. For instance, the Derived Column Transformation, which is a Data Flow component, can expose expressions through its parent Data Flow task. So, when you are developing a custom component, you can tag a property that can be exposed through the Data Flow task so that you can add some dynamic expressions to it.

Handling Events at Package Run Time

When the packages are executed, variables change values, and data flows from sources to destinations and gets transformed along the way by various tasks. These tasks and processes come across changing data and environments at run time, which sometimes results in alerts or events being raised. Integration Services provides event handlers that can act as your agents and be present at the run time configured to respond to the events and add intelligence to the packages by properly handling the events raised.

As containers can have subcontainers, the packages can have event handlers. Event handlers are like subpackages waiting for the events to be raised so that they can come

to action. These powerful tools can extend package functionality greatly when properly implemented. Some of the tasks that can be done with custom event handlers include precharging a cache, sending you an e-mail, or raising a warning when a task fails, dropping the temporary tables created during the package run, processing log files post-execution, or perhaps following an alternative workflow on an event. Event handlers can be created for the packages, Foreach Loop containers, For Loop containers, Sequence containers, and Control Flow tasks as well.

Creating event handlers is like creating a work flow for a package. The Event Handlers tab is used in the SSIS Designer to create event handlers in your package; once created, the event handlers can be explored in the Package Explorer under the Event Handlers node. You can also define connection managers in the Event Handlers tab in case event handlers need to connect to any data source.

When the tasks, containers, or packages raise events during run time, event handlers come into play. The raised event is captured by Integration Services and passed to an event handler for further action. If an event has no event handler defined on the container, it is passed on to the parent container. The parent container runs the event handler in response to this event; however, if the parent container doesn't have an event handler defined, it is flagged up the hierarchy, and so on. This passing on of events up the container hierarchy applies not only at the package level, but can travel up the parent package if the package itself is run as a child package using the Execute Package task.

However, before you get carried away using this functionality, you need to bear in mind that every bit of functionality comes at a cost. If you try to handle all the events at the package level, your package will have too much to manage. In addition, it can be detrimental to performance to let events travel up the ladder when they could easily be handled at the task or container level. Alternatively, if you create event handlers at every level you may end up having too many alerts and warning messages to look through. You can strategically use event handlers with different types of events handled at different levels. To avoid the extra load, which can degrade performance, you can strike a balance by handling events at various levels and at the same time filter out unwanted events from traveling up the ladder. When you design your event handling strategy for a package, you don't need to worry about identifying where the event has been raised irrespective of where it is captured in the hierarchy, because the source of the event is retained and makes it easier for you to identify where the event has been raised.

Sometimes you may need to stop the events from bubbling up to the parent container. For instance, you may be running a task within a looping structure and may want to continue looping in case an error happens in a particular iteration. This is the classic case where you may want to stop the bubbling up of error events. You can do this by setting the value of a system variable System::Propagate to false in the event handler. This will leave the event bound in the task and won't let it propagate up to the

parent container. The way you do this is to attach an event handler to the task and then set the System::Propagate system variable, which resides in the event handler scope, to false. While in the event handler, you can see the system variables by clicking the Show System Variables button in the variables window menu bar. Locate the Propagate variable that will have the default value of True. You can change it here to False, you can write a little script using the Script task to set its value to False at run time, or you can write an expression to evaluate its value to false at run time. The System::Propagate method will work on execution errors; however, validation errors could still occur. You could also use the MaximumErrorCount property on the task to let the package run on errors by increasing the value to a higher number.

Let's do a simple Hands-On exercise to create event handlers for one of the packages created earlier in this chapter.

Hands-On: Creating Event Handlers in an SSIS Package

As the title suggests, you will be creating event handlers in an Integration Services package to understand their configurations and behavior.

Method

In this exercise, you will add the Package1.dtsx package you created earlier in the Maintaining Data Integrity with Transactions project and then will create event handlers to respond on OnTaskFailed and OnPostExecute events.

Exercise (Work with Event Handlers)

In this exercise, you create event handlers for OnTaskFailed and OnPostExecute events and see how various events can be used to create complex event handlers.

1. Open BIDS and create a new Integration Services project with the following details:

Name	Working with Event Handlers
Location	C:\SSIS\Projects

2. When the blank project is created, delete the Package.dtsx in the SSIS Packages node. Then right-click the SSIS Packages node and select Add Existing Package from the context menu.
3. Configure the Add Copy Of Existing Package dialog box to add the package1 .dtsx from the file system with the C:\SSIS\Projects\Maintaining data Integrity with Transactions\Package1.dtsx as the package path. Once the package has been added, double-click the Package1.dtsx package to open it in the Designer.

4. On the Designer surface, go to the Event Handlers tab and click the down arrow in the Executable field, which will open a Package Explorer window. Expand Package1 | Executables | Sequence Container 1 | Executables | Loading Vehicle (see Figure 8-14). Note that you can select a package object, a container, or a task from here. Once you've selected the object for which you want to build event handlers, click OK.
5. For the selected Loading Vehicle executable, you can specify the type of event handler you want to create. Click the down arrow in the Event Handler field and select OnTaskFailed; this indicates that when the Loading Vehicle task fails, this event handler will be executed. Take a moment to go through the other events available. For more details on each of these event types, refer to Microsoft SQL Server 2008 Books Online.
6. Click the Designer surface to create an OnTaskFailed event handler for the Loading Vehicle task. Now, drag and drop the Execute SQL task from the Toolbox onto the Event Handlers surface. Double-click the Execute SQL Task icon to open the editor.

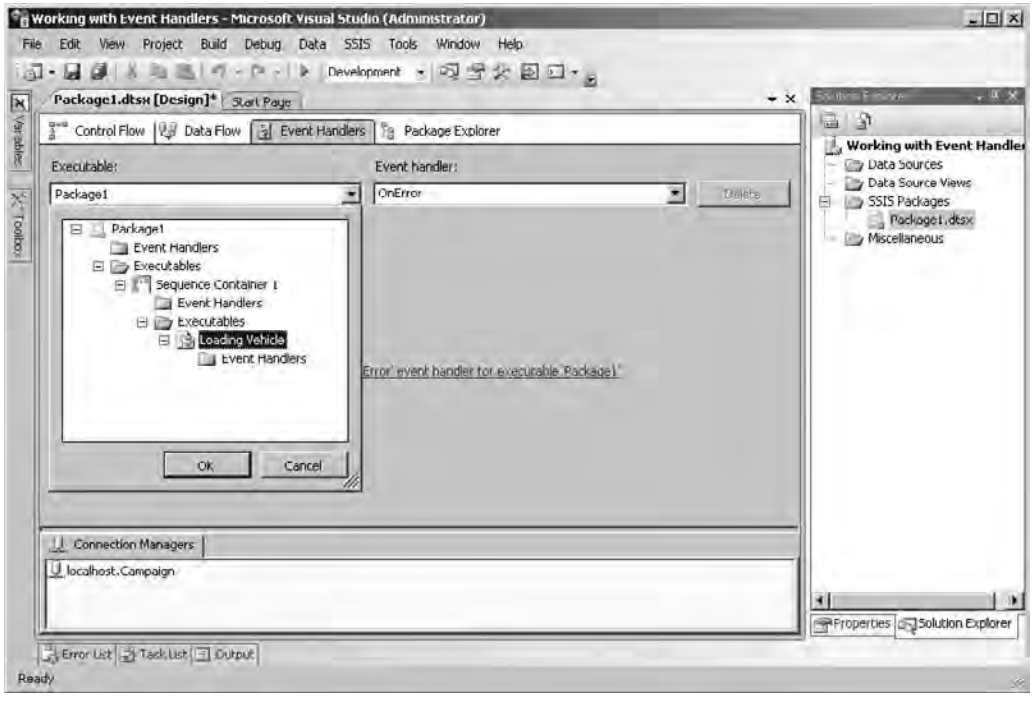


Figure 8-14 *Selecting a container for which to create event handlers*

7. In the General page, select localhost.Campaign Connection Manager in the Connection field. Type the following SQL statement in the SQLStatement field and click OK to close this editor:

```
INSERT INTO Vehicle (CustomerID, VIN, Series, Model) VALUES
('N501', 'UV123WX456YZ789' ,'X11–Series', 'Saloon')
```

   You've added an alternative workflow for the package that will run when the Loading Vehicle task fails.

8. Click in the Executable field and select Package1 executable; then click OK to close the drop-down box.
9. Click in the Event Handler field and choose the OnPostExecute event from the list. Click the Designer surface to create this event handler.
10. When the event handler is created, drag and drop the Send Mail task from the Toolbox onto the Event Handler surface. Double-click the icon of this task to open the editor.
11. Go to Mail page, click in the SmtpConnection field, and then click the arrow button and select <New Connection…>. Specify the name of your SMTP server in the SMTP Server field. If you've configured your local computer to route mails using SMTP service, you can specify localhost in this field. Click OK when you're done.
12. Type your e-mail address in the From and To fields. In the Subject field, type **Working with Event Handlers** and in the MessageSource field, type **The package execution has completed.** Click OK to close the editor. Now you've added a Send Mail task to the package, which will run when the executables in the package complete.
13. Press the F5 key to execute the package. You will see that the Control Flow components in the package fail while the event handlers successfully execute. Select the appropriate executable from the Executables field and the applied event handler in the Event handler field to see its status. Press SHIFT-F5 to switch back to design mode.
14. Run SQL Server Management Studio, connect to Database Engine, and run the following query in the New Query pane:

```
Select * from [Campaign].[dbo].[Vehicle]
```

   You will see one record in the result set that has been added in the Vehicle table by the event handler created on the Loading Vehicle task.

   Check your e-mail and you will see that you've received four mails instead of one. This is because of the multiple executables raising the OnPostExecute event to kick off this Send Mail task. This is why you need to be careful to design and configure a strategy for handling events at various levels.

Review

Event Handlers are like subpackages that can respond to events occurring at run time, and if properly designed and developed, they can let you have a good night's sleep most nights. In this exercise, you created simple event handlers for OnTaskFailed and OnPostExecute events and have seen how these event handlers wake up on the occurrence of these events and perform the work they are configured to perform. You can create quite a complex event handler to perform alternative actions if a particular event happens. However, you also need to be careful while using these event handlers, as the events are flagged up to the parent container at the cost of CPU cycles, unless you filter out the events not to be sent to parent container. Last, you can disable event handlers during design and testing using the DisableEventHandlers property on the objects on which the event handlers have been defined.

As a Data Source for Reporting Services Report

This is an excellent feature that allows you to use an SSIS package as a data source inside a reporting services report. You will study about a DataReader Destination of Data Flow task in Chapter 9 that can act as a data source for external applications. The Data Reader Destination is not actually a destination to write data to some data store; rather, it is an interface for ASP.NET applications to connect and read data from an SSIS package. This feature is not enabled by default in Reporting Services, so you have to enable it before using it. The following describes the steps you need to perform.

Enable SSIS as a Data Source

To enable SSIS as a data source, you need to configure two reporting services config files—RSReportDesigner.Config and RSReportServer.Config files.

1. First, locate the RSReportDesigner.Config file, which exists in the C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies folder in a default installation.
2. Open this file using either any text editor or Visual Studio 2008. The later displays XML in a nice format that is easy to work with. Locate the Extensions section and note that there are three different types of extensions—Render, Data, and Designer extensions. SSIS being a data source, we will add the following code in Data element as shown in Figure 8-15:

```
<Extension Name="SSIS"
Type="Microsoft.SqlServer.Dts.DtsClient.DtsConnection,Microsoft.SqlServer
.Dts.DtsClient, Version=10.0.0.0, Culture=neutral, PublicKeyToken=89845dcd808
0cc91"/>
```

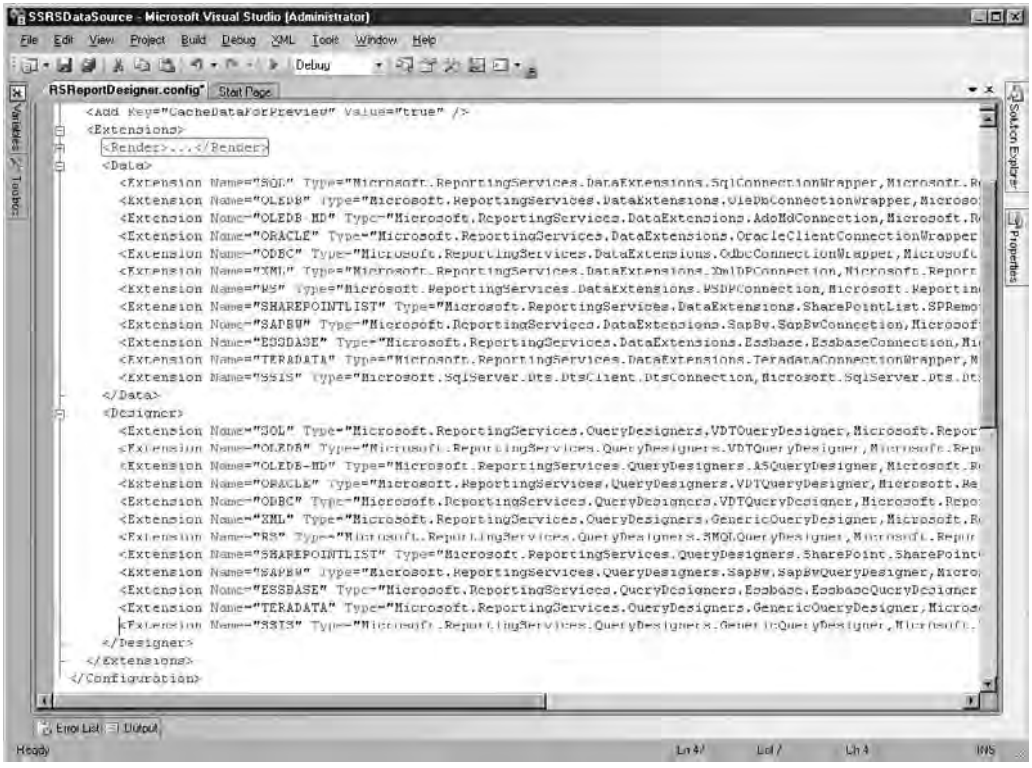


Figure 8-15 *Editing the RSReportDesigner.Config file*

3. And add the following code in the Designer element:

```
<Extension Name="SSIS"
Type="Microsoft.ReportingServices.QueryDesigners.
GenericQueryDesigner,Microsoft.ReportingServices.QueryDesigners"/>
```

4. Now locate and open the rsreportserver.config file, which should be located in C:\Program Files\Microsoft SQL Server\MSRS10_50.MSSQLSERVER\Reporting Services\ReportServer folder in a default installation. In the file locate the Extensions section and add the following code in the Data element:

```
<Extension Name="SSIS" Type="Microsoft.SqlServer.Dts.DtsClient
.DtsConnection,Microsoft.SqlServer.Dts.DtsClient, Version=10.0.0.0,
Culture=neutral, PublicKeyToken=89845dcd8080cc91"/>
```

5. You may have to restart the Reporting Services service for the changes to take effect.

Using SSIS as a Data Source

After you have enabled the SSRS to use SSIS as a data source, you can use this feature.

1. First you have to build an SSIS package in which your data flow task should have DataReader Destination as a destination. The name you assign to the DataReader Destination will be used in the Reporting Services, so keep it short. In our example, I'm using the default name—DataReaderDest.
2. Now start a new reporting services project in BIDS and add a data source. In the Type field, you will see SSIS listed as a data source. Select SSIS as shown in Figure 8-16 and then add **-f C:\SSIS\Packages\Sales.dtsx** in the connection string field. The SSRS data source will execute the package using DTEXEC command and needs only the connection path in the connection string.

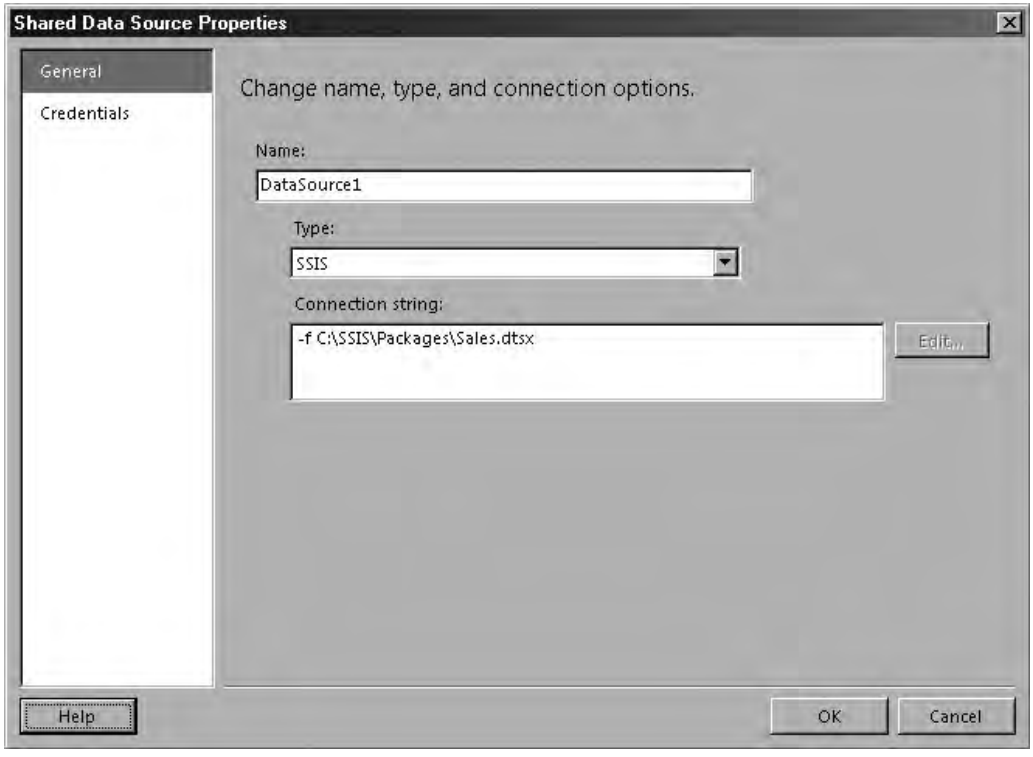


Figure 8-16 *Using SSIS as a data source in the SSRS Report and specifying the connection path*

3. Next add a report and select the data source created in the preceding step. Then in the query field specify the name of the DataReader Destination, which in our case is DataReaderDest as shown in Figure 8-17. You have done whatever was required to use the SSIS DataReader Destination as a source in an SSRS report. Now proceed to complete the report. When you run the report or preview it, you will see data served at run time.

This method is easily configurable, though Microsoft does not recommend it. The primary concern is the security settings, as the package runs under the Reporting Services service account, which might be an issue in some situations. You need to be careful when you deploy this method. However, this example is very useful in demonstrating that the DataReader Destination can serve data dynamically to external ASP.NET applications.

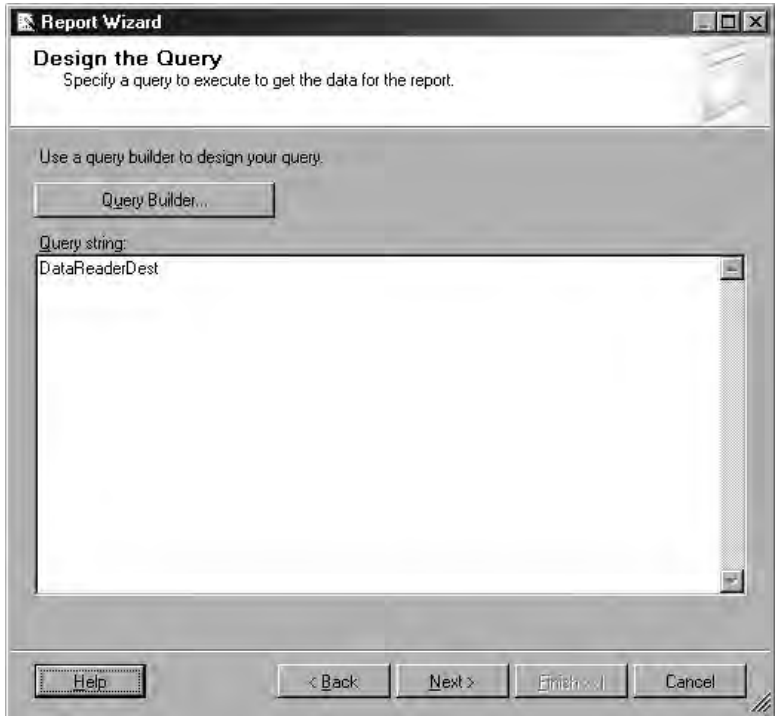


Figure 8-17 *Specifying the DataReader Destination name in the Query String*

Summary

This was an absorbing chapter, as you've worked through various advanced features provided in Integration Services to enhance the service quality of your packages. You've worked with logging and log providers and have enabled logging for a package in the Hands-On exercise. Then you worked with transactions and configured your package to use transactions to handle three different kinds of scenarios for processing transactional data. You also used checkpoints to restart your package from the point of failure and learned about the effect of using transactions on checkpoints. Later in the chapter you extended the Contacting Opportunities package to create personalized e-mails using variables and property expressions. In the last Hands-On exercise, you created event handlers in your package and saw how they become active when a specified event happens.

With this chapter, you've concluded the workflow configurations of Integration Services projects and their administration. In the next two chapters, you will learn all about the data flow engine and how to transform data using Integration Services Data Flow transformations. Until now, your packages did not include any data flow or Data Flow tasks; however, from now on, we will focus mainly on data flow components and their configurations and will not do much in the control flow. Note that you cannot exclude control flow from your packages, as it must be part of an Integration Services package.

Chapter 9

Data Flow Components

In This Chapter

- From Control Flow to Data Flow
- Data Flow Component Interfaces
- Considerations for Bringing Data into Data Flow
- Data Flow Sources
- Data Flow Transformations
- Data Flow Destinations
- Data Flow Paths
- Summary

Integration Services provides several data-oriented services and the related components that can be configured in the data flow task designer. The data flow within a package extracts the data from a data source; performs the transformations, computations, and derivations; and then loads the transformed data to the data destination. The assembly used to represent a data flow is called a *data flow pipeline.* The data flow engine of Integration Services is a data-driven execution engine that provides a steady flow of data streams onto which atomic computations or processes are applied.

The data flow constitutes three main components: data flow sources, data flow transformations, and data flow destinations. Additionally, these components use data flow paths to move the data from one component to the next in the data flow.

This chapter discusses the separation of data flow from the control flow and the benefits thus achieved. It then discusses the data flow components consisting of data flow sources, data flow transformations, data flow destinations, and data flow paths, plus the inputs, outputs, error outputs, and the external columns to explain their functions. At the end of the chapter, you will work through a Hands-On exercise to use these components.

From Control Flow to Data Flow

An Integration Services package relies on control flow and data flow as integral parts, though other parts such as event handlers and log providers also support the package. The data flow consists of three major components—data flow sources, data flow transformations, and data flow destinations—which are connected via data flow paths.

One of the key enhancements in Integration Services over DTS 2000 is the separation of control flow and data flow engines. The immediate predecessor to SSIS—i.e., SQL Server 2000 Data Transformation Services—had all the control and data flow features available in one engine and hence was difficult to use for creating complex packages to meet some stringent data requirements. Integration Services introduces two engines, called the integration services run-time engine and integration services data flow engine, which separate data flow from control flow.

Inherently, the requirements of control flow and data flow are different for operational and performance reasons. The requirements that drive the creation of an ETL package are to extract data, perform some transformations, and load; when you get to the design board, you realize that the package needs to populate variables along the way, wait for data to be available that might be populated by some other part of the package, perform operations in a particular order, move data from buffer to buffer, and so on. This order of tasks, availability of data, and ability to move data as fast as possible from a buffer to another buffer after applying transformations drive the designers in two opposing directions—applying control to the package work flow and applying transformations that are fast enough to the data flow. Although DTS 2000 fits the bill for tackling these

operations, it lacks the ability to strike a perfect balance between control and data flow requirements for the packages you design for various purposes.

The provision of a separate run-time engine and data flow engine provides better control over packages and enhances the way the packages can be designed with separate data flow. The run-time engine controls the order of the package workflow and provides associated services, such as the ability to define an alternative workflow on the occurrence of an event, to apply breakpoints on the tasks, to manage connections to the data sources, to log data, and to manage transactions and variables. The data flow engine provides high-performance data movement functionality, transformations optimized for speed, and great extensibility by allowing you to create custom transformations on top of prebuilt data sources, transformations, and destinations to load data capable of meeting most of your requirements. You can include multiple data flow tasks in a package, with each data flow task able to support multiple data sources, transformations, and destinations.

When you drop the Data Flow task on the Control Flow Designer surface, you invoke the data flow engine. The Data Flow task is a special task provided in Integration Services that allows you to create data movements and transformations for data in a package. The Data Flow task replaces the DTS 2000 tasks, such as the Data transformation task and the Data Driven Query task, which used to provide data movement and manipulation functionalities. Unlike other tasks, double-clicking the Data Flow task doesn't invoke the Properties or Editor dialog box; instead, it opens another designer surface represented by the Data Flow tab in BI Development Studio (BIDS). The Data Flow task, accessed through the Data Flow tab, provides many additional components such as data sources, transformations, and destinations to build your data movement and transformation part of the package. The following section covers the components available in the Data Flow tab.

Data Flow Component Interfaces

The Data Flow task consists of source and destination adapters that extract and load data between heterogeneous data stores; transformations that modify, summarize, cleanse, and extend data; and the paths that link these components together by connecting output of one component to the input of the other component, thus providing a sequence to the data flow. Some components accept multiple inputs and can have multiple outputs. As mentioned, the Data Flow task can have multiple data sources, transformations, and destinations. These components are designed for high performance with a focus on data movement and manipulation efficiency.

All these components are available in the Toolbox window of the Data Flow tab and can be added to the data flow just like control flow tasks. These components are categorized in three ways: data flow sources, data flow transformations, and data flow destinations. Data flow sources are used to extract data, data flow transformations help

in modifying the extracted data, and this modified data is finally loaded into the data silos using data flow destinations. While these components perform their functions, they use data flow paths to move data from a source component to the destination component. These data flow paths are similar to pipes in a pipeline. A data flow path is quite interesting and helpful to debug and visualize the data. You can see the data and metadata of the data flowing through data flow path using data viewers. Data flow paths are covered in detail later in the chapter.

Each of the data flow components may have one or more inputs, outputs, or error outputs associated with it. The data flow source reads the data from the external data source, such as a flat file or a table of a relational database, using its external interface. It then makes this data available to the downstream components. A data flow source uses outputs to send the data in one or more columns via the data flow path to the inputs of the transformations or destinations. A transformation receives data at its inputs and sends data out at its outputs. Finally, a destination receives data at its inputs and sends data out through its external interface to the destination. These components can optionally be configured to have error outputs as well that can contain all the input columns and two extra columns, one for error code and one for an error column to indicate the reason for failure and the failing column. Let's discuss these interfaces in detail.

External Metadata

This interface writes or reads data to and from external data stores (for example, an Excel worksheet) and keeps a copy of their metadata. During design time, when you create a data flow and add sources to your package, the metadata of the data from the sources is copied to the external columns on data flow sources. Similarly, a data flow destination keeps a copy of the destination metadata in its external columns as a snapshot of the destination data store and compares the data schema with this snapshot before writing to the data store. So, as the external columns keep a schema snapshot of the external data stores, they help in the process of package validation. So if a data store is to be created at run time that doesn't exist at design time, or you make changes to a data store without updating the package, you will get validation errors or warnings displayed by SSIS components. Some of these warnings can be handled by using the ValidateExternalMetadata property on the components, which works like DelayValidation property on the tasks and delays the validation until the run time of the component.

Inputs

Inputs receive data from the data flow path in the input columns. The components keep a copy of the metadata in the input interface as well, and the data received at the inputs can be validated against the cached schema. You can configure an input

to fail a component, ignore failure, or redirect a row in case it receives errors in the input columns. In an Integration Services data flow, only the destinations and transformations have inputs. You might think that as the data flow sources bring in data to the data flow, they have inputs, but this is not the case. The data flow sources use external columns to bring data into the data flow. Inputs and outputs (discussed in the next section) are used when the data flows from one data flow component to another data flow component. Whenever external data sources or storages are involved, the data flows directly through the data flow sources or data flow destinations via the external interface.

Outputs

The data is sent out from the outputs of the sources and transformations through the data flow path to the inputs of the downstream components that could be transformations or destinations. The output columns' data can be validated against the external column schema depending on the data flow components configurations. The output columns are exposed in the Input and Output Properties tab of the Advanced Editor, where you can configure them.

Error Outputs

When the data flows through the processing logic of the component, errors and mismatches can occur. These errors could be related to the data type mismatch between the input data and the corresponding metadata of an external column, or the data coming in may be longer than the length defined in the corresponding external column. These kinds of mismatches may cause data rows to raise errors or truncations. You can configure the error handling in the component to fail the component on an error, ignore the error, or redirect the failing rows to an error output. You can specify the error handling behavior for each column using one of three options. Whenever the failing rows are redirected to error output, two additional columns, ErrorColumn and ErrorCode, are also added to indicate the failing column and the code of failure.

Considerations for Bringing Data into Data Flow

Integration Services uses the data flow engine to extract data from a data source, transform and modify the data once it is in the data flow, and then load it to an external data store. The data flow engine uses data flow sources for extracting data from a data store. Figure 9-1 shows how the data is being extracted and parsed by a data source; it also shows how an erroneous record will be handled.

In Figure 9-1, two records demonstrate how data flows through a data flow source. As the data flow source reads the data from the external source, it translates the source

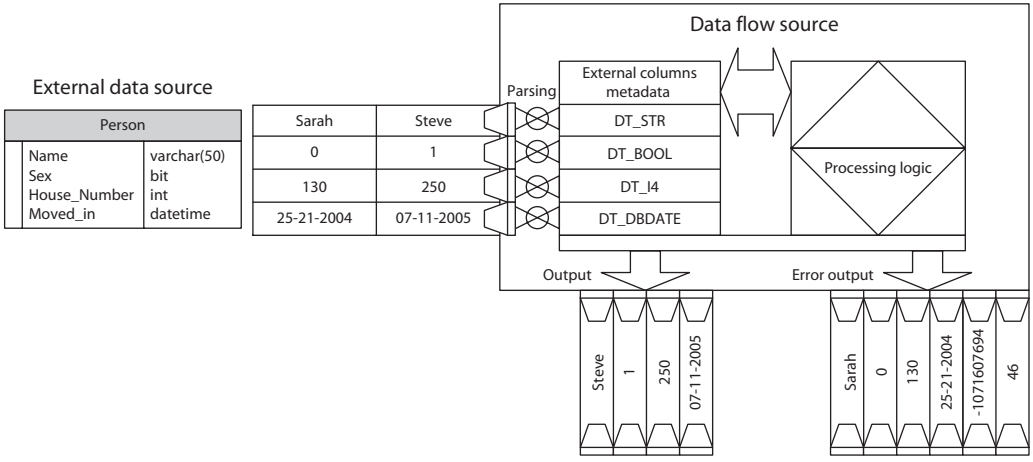


Figure 9-1 *A data flow source extracting and parsing data*

data into the Integration Services data type. During design time, when you drop the data flow source onto the Data Flow Designer surface and configure it to connect to an external data source from which data will be read, the data flow source copies the metadata—i.e., the schema of the data—to its external columns metadata. At run time, when the data is being pulled in, the incoming data columns are parsed into the Integration Services data types using the external columns metadata. In the example, the names Steve and Sarah are parsed into the DT_STR data type, sex, which is indicated using 1 and 0, and have been assigned DT_BOOL; House_ Number has been assigned DT_I4; and the Moved_in data has been assigned the DT_ DBTIMESTAMP data type. Integration Services provides 29 different data types to cover various types of data, such as character data, numerical data, Boolean, dates, text, and image fields. Integration Services provides a wide range of data types and is particular about how they are used; hence, if data has a data type that does not match with the data types available in Integration Services, an error occurs. We will explore many data types along the way as we progress with our Hands-On exercises; however, to know more about each data type, refer to Microsoft SQL Server 2008 Books Online.

The process that converts the source data into Integration Services data types is called *data parsing*. Data flow components can be configured to use either *fast parsing* or *standard parsing*. Fast parsing supports the most commonly used data types with a simple set of routines and can be configured at the column level. Because it uses simple routines and does not check for many other data types, it can achieve high levels of performance. However, it is not available for most of the data flow components

and can parse only a narrow range of data types. For example, fast parsing does not support locale-specific parsing, special currency symbols, and date formats other than year-month-date. Also, fast parsing can be used only when using a Flat File source, a data conversion transformation or derived column transformation, and a Flat File destination, because these are the only components that convert data between string and binary data types. You may use fast parsing for performance reasons when your data meets these requirements, but for all other occasions, you will be using standard parsing. Before using fast parsing, check out the Books Online to find out the data types supported by the fast parsing.

Standard parsing uses a rich set of parsing routines that are equivalent to OLE DB parsing APIs and supports all the data type conversions provided by the automation data type conversion APIs available in Oleaut32.dll and Ole2dsip.dll. For example, standard parsing provides support for locale-sensitive parsing and international data type conversions.

Returning to the data flow, as the data rows arrive and are parsed, they are validated against the external columns metadata. The data that passes this validation check at run time is copied to the output columns; the data that doesn't pass is treated slightly differently. You can define the action the component can take when a failure occurs. Typical errors you can expect at the validation stage include data type mismatches or a data length that exceeds the length defined for the column. Integration Services handles these as two different types of errors—data type errors and data length or data truncation errors. You can specify the action you want the data flow component to take for each type of error from the three options—fail the component, ignore the error, or redirect the failing row to error output fields. You can specify one of these actions on all columns or different actions for different columns. If you redirect the failing rows to the error output fields and link the error output to a data flow destination, the failing rows will be written to the data flow destination you specified.

The error output contains all the output columns and two additional columns for the failing rows, ErrorCode and ErrorColumn, which indicate the type of error and the failing column. In Figure 9-1, note that the record holding data for Sarah has a wrong date specified and hence fails during extract process. As the source was configured to redirect rows, the failing row data is sent to the Error Output. Also, note that the two rightmost columns indicate the type of error and the column number. Every output field is assigned an ID automatically, and the number shown in the ErrorColumn is the ID number of the column failing the extract process. If two columns fail on the same row, the column that fails first is captured and reported. So, in case of multiple failures on a row, you might not know about them till the package fails again after you have fixed the first error.

After pulling in the data rows, a data flow source passes the output rows to the next data flow component—generally a transformation—and the failing rows are passed to another data flow component—which could be a data flow destination, if configured

to redirect the failing rows. You can also redirect the failing rows to an alternate branch in the data flow via a transformation in which you can apply logic to correct the failing rows and bring them back into the main data flow after corrections. This is a highly useful capability that, if used properly, can reduce wastage and improve data quality. When the output of a data flow source is connected to the input of a data flow transformation, the data flows from the input of the transformation, through the processing logic of transformation, and then to its output. Based on the logic of transformation, some rows may fail the process and may be sent to the error output. The main difference to note in comparison to a data flow source is that the transformations do not have external columns and have input columns instead. Figure 9-2 shows the functional layout of a data flow transformation.

Finally, after passing through the data flow source and the data flow transformations, the data will reach a data flow destination so that it can be written to an external data store. Like data flow sources, a data flow destination can also read the schema information from

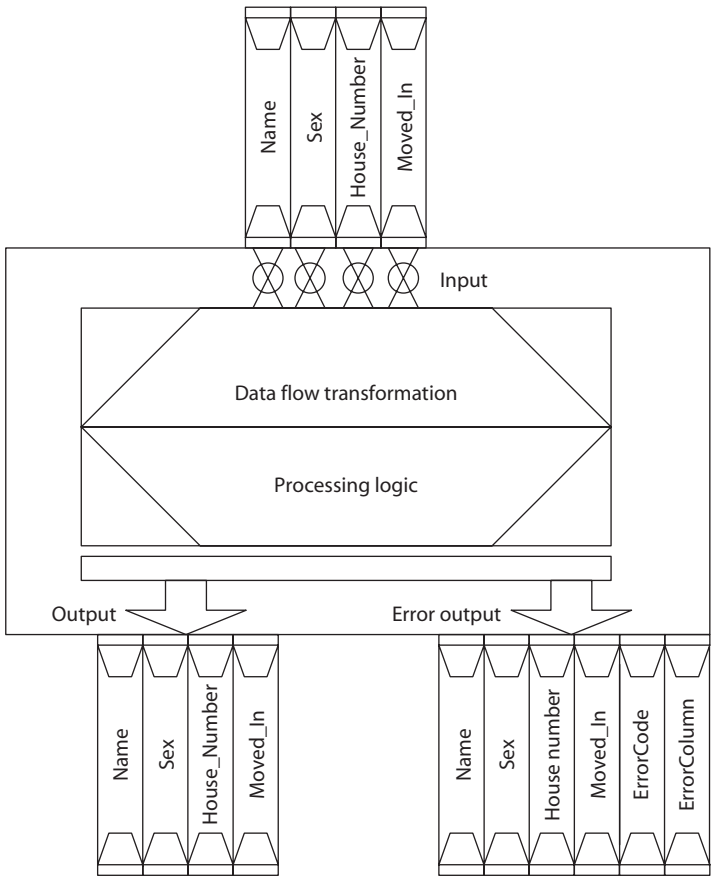


Figure 9-2 *Data flow transformation showing Input, Output, and Error Output columns*

the external data store and copy the metadata to the external columns. When the data flows through the data flow destination, it gets validated against this metadata before being written to the external data store. If configured to redirect the failing rows, the failing rows may be sent to the error output columns and the rest of the successful data is written to the external data store. Note that a data flow destination does not have output columns. Figure 9-3 shows the data flow through a data flow destination.

At this point it is worth mentioning that a pipeline or a data flow path usually terminates in a data flow destination; however, that is not necessary. Sometimes, you may prefer to terminate a data flow path in a transformation. For example, while testing and debugging, you can break and terminate the data flow path by adding a Row Count transformation to see what happens at a particular point in the data flow path while ignoring rest of the pipeline. With the preceding description of how the data is extracted from a data source, and how it flows through the transformations and destinations before being written to the external data store, let's study these components to learn more about them and to learn how many types of data sources or destinations are available in the data flow.

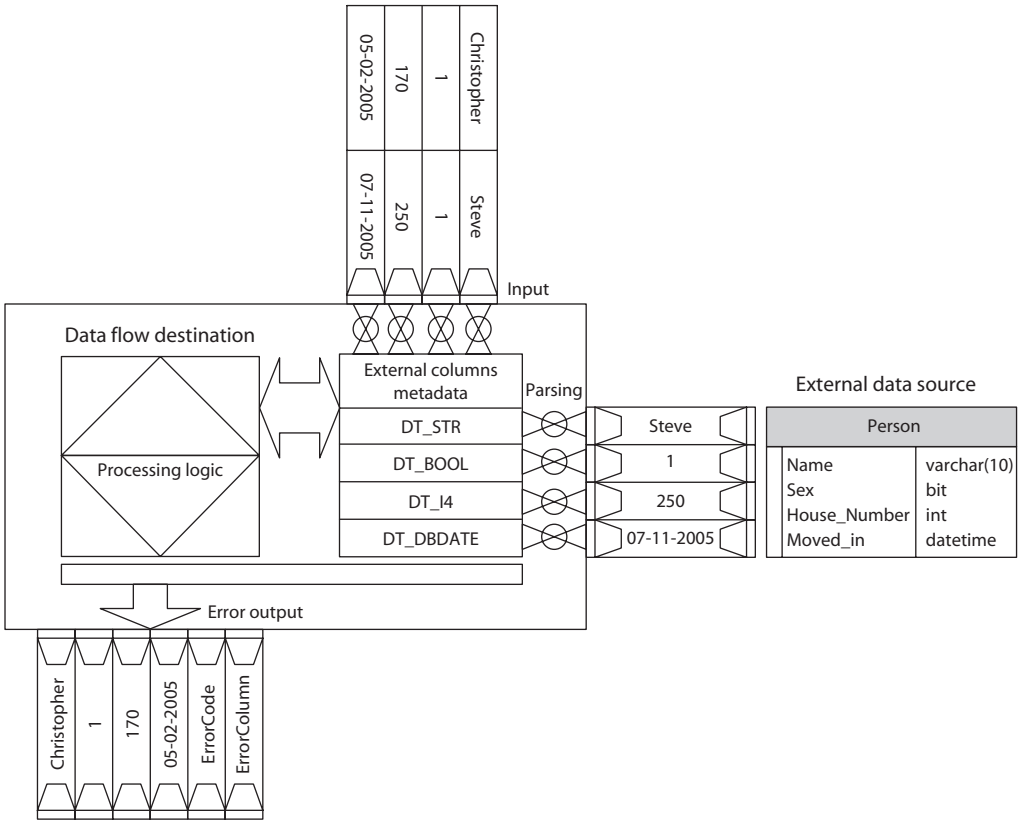


Figure 9-3 *Data flow destination showing the flow of data to the external data store*

Data Flow Sources

While building a data flow for a package using BIDS, your first objective is to bring the data inside the Integration Services data flow so that you can then modify the data using data flow transformations. Data flow sources are designed to bring data from the external sources into the Integration Services data flow. A data flow source reads the external data source, such as a flat file or a table in relational database, and brings in the data to the SSIS data flow by passing this data through the output columns on to the downstream component, usually a data flow transformation. During design time, the data flow source keeps a snapshot of the metadata of external columns and can refer to it at run time. If the data in certain rows doesn't match with this schema at run time, the data sources can be configured to redirect those rows to the error output columns that can be dealt with separately. Integration Services provides six preconfigured data flow sources to read data from a variety of data sources, plus a script component that can also be scripted as a data flow source. If existing data flow sources do not meet your requirements, you can always build yourself a custom data flow source using the Integration Services object model. Scripting a data flow source has been discussed in Chapter 11.

The following two tables list the preconfigured data flow source adapters and the interfaces they have.

Source	Description
ADO NET Source	Extracts data from .NET Framework data providers, such as SQL Server 2008 using ADO.NET Connection Managers.
Excel Source	Extracts data from an Excel worksheet using Excel Connection Manager.
Flat File Source	Reads data from a text file using Flat File Connection Manager.
OLE DB Source	Extracts data from OLE DB–compliant relational databases using OLE DB Connection Manager.
Raw File Source	Extracts data from a raw file using a direct connection.
Script Source Component	Hosts and runs a script that can be used to extract, transform, or load data. Though not shown under data flow sources in the Toolbox, the Script Component can also be used as a data flow source. This component is covered in Chapter 11.
XML Source	Reads data from an XML data source by specifying the location of the XML file or a variable.

Data Flow Source	Input	Output	Error Output	Custom UI	Connection Manager
ADO NET source	No	1	1	Yes	ADO.NET Connection Manager
Excel source	No	1	1	Yes	Excel Connection Manager
Flat File source	No	1	1	Yes	Flat File Connection Manager
OLE DB source	No	1	1	Yes	OLE DB Connection Manager
Raw File source	No	1	No	Yes	Not required
XML source	No	1	1	Yes	Not required

From the preceding table, you can make out that not all data flow sources have Error Output and not all data flow sources require a connection manager to connect to the external data source; rather, some can directly connect to the data source such as XML Source. But the important thing to understand here is that data flow sources don't have an input. They use the external columns interface to get the data and use output columns to pass the data to downstream components. We will study more about each of the data flow sources in the following topics.

ADO NET Source

When you need to access data from a .NET provider, you use an ADO.NET Connection Manager to connect to the data source and then can use the ADO NET source to bring the data inside the data flow pipeline. You can configure this source to use either a table or a view or use an SQL command to extract rows from the external .NET source. The ADO NET source has one regular output and one error output. The ADO NET source has a custom UI though you can also use advanced editor to configure some of the properties that are not exposed in the custom UI. The custom UI has three pages to configure—Connection Manager, Columns, and Error Output.

- **Connection Manager** Specify the ADO.NET Connection Manager here that the ADO NET Source uses to connect to the data source. Select one of the ADO.NET Connection Managers already configured in the package from the drop-down list provided under the ADO.NET Connection Manager field, or you can use the New button to add a new ADO.NET Connection Manager.
- **Columns** This page shows you the list of columns read from the external .NET data source and cached into the External Columns interface. It also shows you the corresponding Output Columns that, by default, have the same names as the cached schema in the External Columns. You can change the Output Column names here if you want to call them differently inside your pipeline.

- **Error Output** When outputting the columns read from the external data source, some rows may fail due to wrong data coming through. These failures can be categorized as errors or truncations. Errors can be data conversion errors or expression evaluation errors. The data may be of wrong type—i.e., alphabetical characters arriving in an integer field—causing errors to be generated. Truncation failures may not be as critical as errors—in fact, sometimes they are desirable. Truncation lets the data through, but truncates the data characters for which the length becomes more than the specified length—for example, if you specify the city column as VARCHAR(10), then all the characters after first ten characters will be truncated when it exceeds the ten-character length. You can configure this component for data type errors or truncations of data in columns to fail, ignore the error, or redirect the failing row to error output. See Figure 9-4 for an explanation.

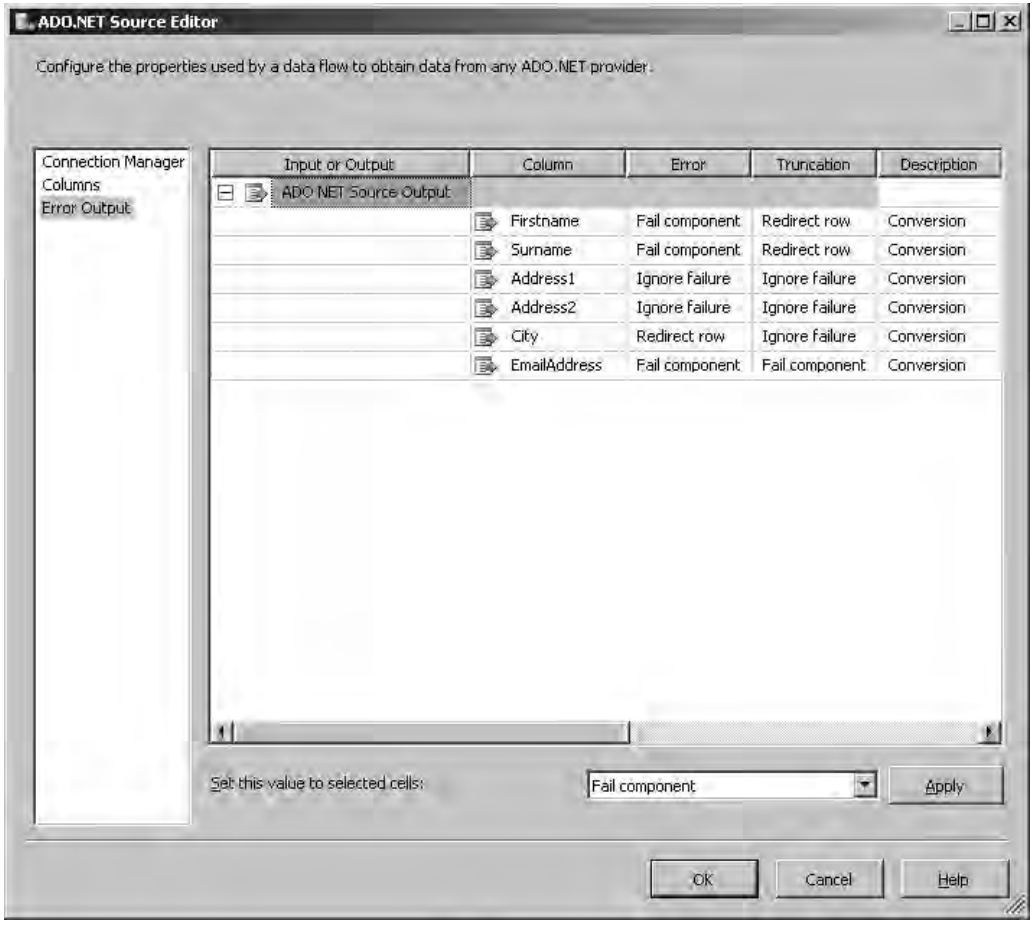


Figure 9-4 *Setting error conditions for data type mismatches and truncation errors*

- **Fail Component** This is the default option for both errors and truncations and will fail the data flow component, ADO NET Source in this case, when an error or a truncation occurs.
- **Ignore Failure** You can ignore the error or the truncation and carry on outputting the data from this component.
- **Redirect Row** You can configure the ADO NET Source to redirect the failing row to the error output of the source adapter, which will be handled by the components capturing the error output rows.

If you have many columns to configure for different settings, you may find using the Set This Value To Selected Cells field easier to apply a value to all of your selected cells.

Excel Source

When you need to work with data in Excel files, you can either use the OLE DB Source with Microsoft Jet 4.0 OLE DB Provider or simply use an Excel source to get the data into the pipeline. You will configure the Excel Connection Manager to connect to an Excel workbook and then use that connection manager inside an Excel source to extract the data from a worksheet and bring it in to the pipeline. You can treat an Excel workbook as a database and its worksheets as tables while configuring the Excel source. A range in the Excel workbook can also be treated as a table or a view on database. The Excel source adapter has one regular output and one error output.

This component has its own user interface, though you can also use Advanced Editor to configure its properties. When you open the Excel Source Editor in the data flow designer you will see that this source adapter also has three pages to configure.

- **Connection Manager** The Connection Manager page opens by default, where you can select the connection manager from the drop-down list in the OLE DB Connection Manager field. The Data Access Mode drop-down list provides four options. Depending upon your choice in the Data Access Mode field, the interface changes the fields to provide relevant information.
  - **Table or view** This option treats the Excel sheet as a table or a view and extracts data from the specified worksheet in the Name Of The Excel Sheet field.
  - **Table name or view name variable** When you select this option, the subsequent field changes to Variable Name field. This option works similar to the Table Or View option, except instead of expecting the name of the worksheet or Excel range to be specified directly, it lets you specify the name of a variable in the Variable Name field, from which it can read the name of the Excel range or the worksheet.

- **SQL Command** Selecting this option in the Data Access Mode field changes the interface to let you provide an SQL statement in the SQL Command Text field to access data from an Excel workbook. You can either type in SQL directly or use the provided Build Query button to build an SQL query. (I recommend you to use this query builder to access data from the Excel sheet, even if you know how to write complex SQL queries for SQL Server, because there are some lesser-known issues on accessing data from Excel workbooks using SQL.)

  You can also use a parameterized SQL query, for which you can specify parameter mappings using the Parameters button. When you click Parameters, you get an interface that lets you map a parameter to a variable.

- **SQL Command From Variable** This option works as an SQL command, except it reads the SQL statement from a variable specified in the Variable Name field.

- **Columns** You can map external columns to output columns in this page and can change the names of output columns.
- **Error Output** As explained in the ADO NET Source, you can configure this page to fail the component, ignore the error, or redirect the row in case an error occurs in a data column.

While the Excel Source Editor allows you to configure the properties for Excel source, you may need to use the Advanced Editor to configure the properties not exposed by the Excel Source Editor. These properties include assigning a name and description to the component, specifying a timeout value for the SQL query, or most important, changing the data type for a column. While working with the Advanced Editor, get acquainted with the various options available in its interface.

Flat File Source

The Flat File source lets your package read data from a text file and bring that data into the pipeline. You configure a Flat File Connection Manager to connect to a text file and specify how the file is formatted. Also, you will specify the data type and length of each column in the Flat File Connection Manager that will set guidelines for the Flat File source to handle it appropriately. The Flat File source can read a delimited, fixed width, or ragged right–formatted flat file. To know more about these file types, refer to Chapter 3.

The Flat File source has a custom user interface that you can use to configure its properties. Also, as with the Excel source adapter, its properties can be configured using an Advanced Editor. When you open the Flat File Source Editor, the Connection

Manager page opens up by default. You can select the Connection Manager from the drop-down list provided in the Flat File Connection Manager field. The flat files contain nothing for the null values, and if you want to keep these null values, check the box for "Retain null values from the source as null values" in the data flow option. By default, this check box is unchecked, which means the Flat File source will not keep null values in the data but will replace null values with the appropriate default values for each column type—for example, empty strings for string columns and zero for numeric columns. Note that the file you are trying to access must be in a delimited format. This is because the fixed width and/or ragged right–format files do not contain blank spaces; you need to pad the fields with a padding character to the maximum width so that the data cannot be treated as null values by the Flat File source adapter. These format settings for the flat file are done in the Flat File Connection Manager.

The Columns and Error Output pages can be configured as described in the Excel source adapter. You can use Columns page on the Flat File source to map external columns to the output columns and the Error Output page to configure the error and truncation behavior when the mismatched data comes along, using the three options of Fail Component, Redirect Row, or Ignore Failure.

This source has two important custom properties—FastParse and UseBinaryFormat—that are exposed in the Advanced Editor. The configurations for both these properties are done in the Input and Output Properties tab of the Advanced Editor for the Flat File source. Depending on the data you are dealing with, you can set the FastParse option for each of the columns by selecting the column in the Output Columns section and then going to the Custom Properties category of the column properties. By default, the FastParse option is set to False (see Figure 9-5), which means the standard parsing technique will be used. (Remember that standard parsing is a rich set of algorithms that can provide extensive data type conversions, where as fast parsing is relatively simplified set of parsing routines that supports only the most commonly used data and time formats without any locale-specific data type conversions.)

The second property, UseBinaryFormat (also shown in Figure 9-5), allows you to let the binary data in the input column pass through to the output column without parsing. Sometimes you have to deal with binary data, such as data with the packed decimal format, especially when you're receiving data from a mainframe system or the data source is storing data in the COBOL binary format—for example, you might be dealing with IBM EBCDIC–formatted data. In such cases, you may not prefer the Flat File source to parse the data; rather, you would like to parse it separately using special rules that are built based on how it has been packed into the column in the first place. By default UseBinaryFormat is set to false, which means the data in the input column will be parsed using Flat File source parsing techniques. To use this property, set UseBinaryFormat to true and the data type on the output column to DT_BYTES, to let the binary data be passed on to the output column as is.

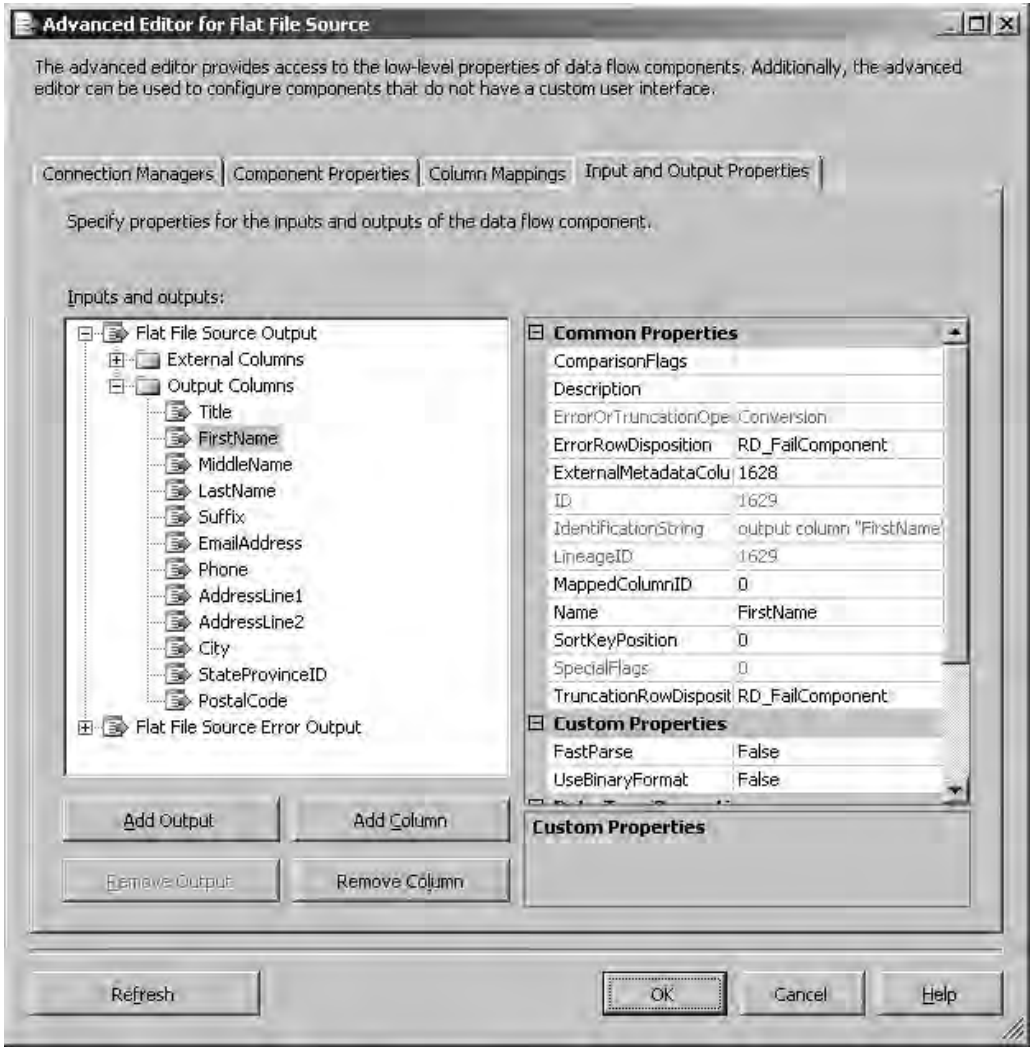


Figure 9-5 *Flat File Advanced Editor showing FastParse and UseBinaryFormat properties*

OLE DB Source

You will be using an OLE DB source whenever you need to extract data from a relational database that is OLE DB–compliant, such as Microsoft SQL Server, an Oracle database server, IBM's DB2 database server, an Access database, or even an Excel file. An OLE DB provider is used in an OLE DB Connection Manager to

connect to the database, and an OLE DB source uses the OLE DB Connection Manager to bring the data into the data flow. Just to remind you here that if you use an Integration Services Data Source inside an OLE DB Connection Manager, you can actually make Data Source or Data Source view available through the OLE DB Connection Manager to all of your projects within an Integration Services project. Not all OLE DB providers are the same, you should check the documentation of the OLE DB provider that you want to use in your package, as some limitations are associated with different types of OLE DB providers. The OLE DB source has one regular output and one error output.

This data flow component has a custom user interface that is similar to the Excel source adapter discussed earlier, and its properties can also be configured using the Advanced Editor. When you open an OLE DB Source Editor, it takes you into the Connection Manager page. Here you can specify an OLE DB Connection Manager from the drop-down list provided in the OLE DB Connection Manager field. The OLE DB Source provides you the following four levels of data access mode options:

- **Table Or View** Specify the name of the table or the view from which you want to extract data.
- **Table Name Or View Name Variable** Specify the variable name, which will be the holding name of the table or the view.
- **SQL Command** Write an SQL statement to extract data. You have the option of using a parameterized SQL query here.
- **SQL Command From Variable** Specify the variable name that will be holding an SQL statement to be passed on to OLE DB source.

For more details on these options, refer to the section "Excel Source." where they have been discussed in detail.

In the Columns page, you can see the Output Columns mappings to the External Columns. The matching name will be written for you in the Output Column, which you can change if you wish to do so.

In the Error Output page, you can specify how the OLE DB source should handle an error or a truncation for each column.

Raw File Source

All the prebuilt Integration Services sources other than this Raw File Source require a connection manager to connect to a data source. The Raw File source doesn't require a connection manager. The Raw File source establishes a direct connection to the file containing the raw data and brings that raw data into the data flow. The raw data

written in the raw file is native to the source and requires no parsing or translation during import, so the Raw File source can extract the data much faster. You can use raw files to stage data due to its fast read and write operation; for example, in scenarios for which you export data from your system for later transformation and loading back into the similar system, this may be an ideal choice. The Raw File source has only one output and no error output.

The Raw File source has been given a custom user interface in this version of Integration Services, but that doesn't mean that you can't use the Advanced Editor to configure its properties. The custom UI is very simple and contains only two pages.

- **Connection Manager** Though this page is named like the ones in other components where you select a connection manager, this component doesn't use any SSIS connection manager; rather, it connects to the raw file directly. In the Access Mode field, you can specify a filename or choose to get the filename from a variable. Depending on your choice of access mode, the interface changes the available fields to collect relevant information. Choosing File Name lets you specify the raw filename in the File Name field, and choosing File Name From Variable lets you specify the name of the variable that holds the raw filename in the Variable Name field.
- **Column Mappings** Shows the mappings of external columns to output columns and allows you to rename the output columns.

Script Component Source

The preconfigured data flow sources in Integration Services have only one output available—for example, Flat File source, Excel source, and OLE DB source all have single outputs. If you need to output data to more than one downstream component in the data flow, you can't do that using these preconfigured components. The only option you have in this case, or in other similar cases when existing source components do not meet your requirements, is to create your own data flow source. You can write yourself a custom component, but an easier option is to use the script component as a source. The script component has not been shown under data flow sources as a data source, but it can be configured as a data source. When you drop the script component onto the Data Flow Designer surface, you are asked to select whether you want to configure this component as a source, a transformation, or a destination. Based on your choice, the script component customizes the interface and options appropriate for the purpose. As a source, the script component doesn't have any input and only one output to start with. You can add additional outputs using the Add Output button in the Inputs and Outputs page. Configuring a script component as a data flow source is covered in Chapter 11.

XML Source

XML Source reads the XML data from an XML file or from a variable containing XML data and brings that data into the data flow. This source has a custom user interface to edit properties but also uses the Advanced Editor for configurations of some of its properties.

When you open the XML Source Editor, the Connection Manager page appears, where you specify how you want to connect to and access data from the XML data file. Depending upon the option you choose, the interface changes to collect the relevant information:

- **XML file location** Lets you specify the location and the filename for the XML data file in the XML Location field.
- **XML file from variable** Allows you to use a variable to specify the XML data filename and location. You then provide name of the variable containing XML data file details in the Variable Name field.
- **XML data from variable** Access XML data directly from a variable by specifying the name of the variable in the Variable Name field.

Next you can choose schema options: use an inline schema or provide an XML schema definition file in the XSD format. When the XML data file contains the XSD schema itself to validate its structure and data, you will be using inline schema option; otherwise, you will have to supply an external schema definition file (XSD). If you don't have an XSD file with you, you can generate this file by clicking the Generate XSD button and providing a location and name for the XSD file. This file is required to interpret the relationships among the elements in the XML data file.

In the Columns page, you can map output columns to external columns and in the Error Output page, you can specify how the XML Source should handle errors and truncations of data for each column. This source can have multiple regular outputs and multiple error outputs.

Data Flow Transformations

Once the data has been captured by source adapters and passed on to data flow path, you can modify this data using a wide range of data flow transformations provided in SSIS. You can use data flow transformations to aggregate column values, update column values, add columns to the data flow, merge data, and accomplish many more data modifications. The data flow transformations can have single or multiple inputs or outputs, depending upon the type of transformation you choose. A data

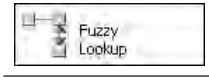
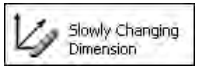
flow transformation receives data on its input columns from the output columns of a data flow source or a transformation. After applying transformations on to the data, the data flow transformation provides the output through its output columns to the input columns of the downstream component, which can be another data flow transformation or a data flow destination. Some of the data flow transformations can also have error outputs. The data flow transformations send the data rows that fail to transform to the error output columns that can be dealt with separately. Data flow transformations do not have external columns, but you can create one with external columns programmatically.

The following tables list the 29 data flow transformations grouped together in categories on the basis of the function they perform in Integration Services. Future service packs or add-ons may bring in more transformations. These transformations provide a rich set of functionalities in many areas such as data cleansing, data standardization, BI functionalities, loading slowly changing data warehouse dimension tables, pivoting and unpivoting, and a facility to write script using the Script Component transformation. However, if you still need a functionality that can't be met by preconfigured components, you can write custom transformations with synchronous outputs or asynchronous outputs. Transformations with synchronous outputs make modifications to data inline—i.e., as the data rows flow through the component one at a time—whereas transformations with asynchronous outputs cannot process each row independently of all other rows—for example, an aggregate transformation needs all the rows before it can perform an operation across rows. You will learn more about synchronous and asynchronous components and programming options in Chapter 11.

The following sections contain brief definitions about the data flow transformations, however, the details of configurations and usage exercises will be covered in next chapter.

Business Intelligence Transformations

This category groups together the transformations that allows you to perform business intelligence operations such as data cleansing, data standardizing, text mining, and running DMX prediction queries.

Transformation	Description
	Performs data cleansing and Standardizes values in column data.
	Uses fuzzy matching to cleanse or standardize data.
	Configures the updating of slowly changing dimension in data warehouse dimension tables.

Transformation	Description
Term Extraction	Extracts a term from text in the input columns and write the extracted term to an output column.
Term Lookup	Performs a lookup and count for the terms in a table that are defined in a lookup table.
Data Mining Query	Runs DMX queries for performing prediction queries against data mining models.

Row Transformations

The transformations in this category allow you to update column values or create new columns on a row-by-row basis.

Transformation	Description
Character Map	Applies string functions to string data type columns.
Copy Column	Creates new columns in the transformation output by copying input columns.
Data Conversion	Converts data type of a column and optionally copies the converted data to a new output column.
Derived Column	Creates new derivations of data by applying expressions using a combination of input columns, variables, functions, and operators; the results of this derivation can be used to modify an existing column or can be copied in to a new column in the output.
Export Column	Exports data from a pipeline column in to a file. This transformation can be especially useful to export DT_TEXT, DT_NTEXT, or DT_IMAGE data type data from the pipeline into a file.
Import Column	Reads data from a file and add it to the columns in the data flow.
Script Transformation Component	Hosts and runs a script that can be used to transform data.
OLE DB Command	Updates, inserts, or deletes rows using SQL commands in a data flow.

Rowset Transformations

The transformations in this category work on collections of rows and allow you to perform operations such as aggregate, sort, sample, pivot, and unpivot row sets.

Transformation	Description
Aggregate	Performs aggregate functions such as average, sum, and count and copy the results to the output columns.
Sort	Sorts input columns in ascending or descending order and copies the sorted data to the output columns.
Percentage Sampling	Creates a sample data set by specifying a percentage to randomly select input rows.
Row Sampling	Creates a sample data set by specifying the exact number of output rows to randomly select input rows.
Pivot	Pivots the input data on a column value to get a less normalized but compact view of data.
Unpivot	Creates a more normalized version of a de-normalized table.

Split and Join Transformations

This category groups the transformations that allow you to distribute rows to different outputs, multicast input rows, join multiple inputs into one output, and perform lookup operations.

Transformation	Description
Conditional Split	Routes data rows to different outputs, depending on the content of the data.
Multicast	Distributes the input data set to multiple outputs.
Union All	Merges multiple input data sets into one output.
Merge	Merges two sorted data sets into a single output data set.

Transformation	Description
Merge Join	Joins two sorted input data sets using a FULL, LEFT, or INNER join into a single output.
Cache Transform	Prepares a cache file to be used by a lookup transformation by writing data in the pipeline to a cache connection manager.
Lookup	Perform lookups by joining data in the input columns with reference data set columns.

Auditing Transformations

These transformations are used to add audit information and count rows.

Transformation	Description
Audit	Includes the system environment information in the data flow.
Row Count	Counts the rows flowing through the data flow and writes the final count in a variable.

Data Flow Destinations

A data flow destination is the last component in the Integration Services data flow and writes the data received to an external data store. This component has no output columns, though it may have error output columns for redirecting error rows. A data flow destination writes the data received at its input columns to an external data store or to an in-memory data set via its external columns interface. You can configure the data flow destination to handle errors in data in the same way as you configure a data flow source. You can choose to fail the component, ignore the error, or redirect the error rows to the error output columns. If the data flow destination can't write some of the data rows to the external data store due to errors in data or data not matching with the external columns, it can redirect those rows to the error output columns depending how it is configured to redirect error rows.

Integration Services provides 12 preconfigured destinations plus a script component that can be used as a destination. The following tables list the available destinations and the functions they perform.

Destination	Description
ADO.NET Destination	Writes data to ADO.NET–compliant databases.
Data Mining Model Training	Passes the data to train data mining models through the data mining model algorithms.
DataReader Destination	Exposes data in the data flow using ADO.NET DataReader interface for applications such as SQL Server Reporting Services.
Dimension Processing	Loads and processes an SQL Server Analysis Services Dimension.
Excel Destination	Writes data from a data flow to an Excel workbook.
Flat File Destination	Writes data from a data flow to a flat file.
OLE DB Destination	Loads data from a data flow to an OLE DB–compliant data stores.
Partition Processing	Loads and process an SQL Server Analysis Services partition.
Raw File Destination	Writes data to a raw file.
Recordset Destination	Creates and populates an in-memory ADO record set.
Script Destination Component	Hosts and runs a script that can be used to load data. Though not shown under Data Flow Destinations in the Toolbox, Script Component can be used as a data flow destination. This component is covered in Chapter 11.
SQL Server Compact Destination	Inserts rows into an SQL Server Compact database.
SQL Server Destination	Bulk-loads data into an SQL Server table or view.

All these destinations have an input, and some of them have an error output to meet most of your requirements to write data to external stores. However, if you find that existing data flow destinations do not do what you want, you can custom build a destination to suit your requirements in the Integration Services object model. The following table lists input and outputs available in each data flow destination.

Data Flow Destination	Input	Output	Error Output	Custom UI	Connection Manager
ADO NET destination	1	No	Yes	Yes	ADO.NET Connection Manager
Data Mining Model Training destination	1	No	No	Yes	Analysis Services Connection Manager
DataReader destination	1	No	No	No	Not Required
Dimension Processing destination	1	No	No	Yes	Analysis Services Connection Manager
Excel destination	1	No	1	Yes	Excel Connection Manager
Flat File destination	1	No	No	Yes	Flat File Connection Manager
OLE DB destination	1	No	1	Yes	OLE DB Connection Manager
Partition Processing destination	1	No	No	Yes	Analysis Services Connection Manager
Raw File destination	1	No	No	Yes	Not Required
Recordset destination	1	No	No	No	Not Required
SQL Server Compact destination	1	No	1	No	OLE DB Connection Manager
SQL Server destination	1	No	No	Yes	OLE DB Connection Manager

ADO NET Destination

You can use an ADO NET destination to load data into a table or view of an ADO.NET–compliant database. If you have a database into which you can load data using either an ADO NET destination or an OLE DB destination, it is generally preferred to use the OLE DB destination due to its performance and resilience; however, an ADO NET destination does provide more options for you to load data into the databases when an OLE DB destination cannot be used. The ADO.NET Destination Editor has three pages: Connection Manager, Mappings, and Error Output. The Connection Manager page enables you to specify an ADO.NET Connection Manager and the table or view into which the data is to be loaded. You have options in this page to create a new connection manager or a new table. In the Mappings page, you can map input columns to the destination columns. Finally, in the Error Output page, you can configure the error response conditions.

Data Mining Model Training Destination

Data changes with time, and the type of information that can be gathered from the data changes along with it. Mining data for useful and relevant information requires regular data mining model training so that it can keep up with the challenges of ever-changing data. The data mining model training destination passes data through the data mining model algorithms to train the data mining models.

To train the data mining models using this destination, you need a connection to SQL Server Analysis Services, where the mining structure and the mining models reside. For this, you can use Analysis Services Connection Manager to connect to an instance of Analysis Services or to the Analysis Services project. The Data Mining Model Training Editor has two tabs, Connection and Columns, in which you can configure the required properties. In the Connection tab, you specify the connection manager for Analysis Services in the Connection Manager field and then specify the mining structure that contains the mining models you want this data to train. Once you select a mining structure in the Mining structure field, the list of mining models is displayed in the Mining models area, and this destination adapter will train all the models contained within the specified mining structure. In the Columns tab, you can map available input columns to the Mining structure columns. The processing of the mining model requires data to be sorted, which you can achieve by adding a sort transformation before the data mining model training destination.

DataReader Destination

When your ADO.NET–compliant application needs to access data from the data flow of an SSIS package, you can use the DataReader destination. Integration Services can provide data straight from the pipeline to your ADO.NET application in cases where you need dynamic processing to happen when users request using the ADO.NET DataReader interface. SSIS data processing extension facilitates provision of the data via the DataReader destination. An excellent use of the DataReader destination is as a data source for an SSRS report.

The DataReader destination doesn't have a custom UI but uses the Advanced Editor to expose all the properties organized in three tabs. You can specify Name, Description, LocaleID, and ValidateExternalMetadata properties in the Common Properties section of the Component Properties tab. In the Custom Properties section, you can specify a ReadTimeout value in milliseconds, and if this value is exceeded, you can choose to fail the component in the FailOnTimeout field.

In the Input Columns tab, you can select the columns you want to output, assign each of them an output alias, and specify a usage type of READONLY or READWRITE from the drop-down list box. Finally, the Input And Output Properties tab lists only the input column details, as DataReader destination has only one input and no error output.

Dimension Processing Destination

One of the frequent uses of Integration Services is to load data warehouse dimensions using the dimension processing destination. This destination can be used to load and process an SQL Server Analysis Services dimension. Being a destination, it has no output and one input, and it does not support an error output.

The dimension processing destination has a custom user interface, but the Advanced Editor can also be used to modify properties that are not available in the custom editor. In the Dimension Processing Destination Editor, the properties are grouped logically in three different pages. In the Connection Manager page, you can specify the connection manager for Analysis Services to connect to the Analysis Services server or an Analysis Services project. Using this connection manager, the Dimension Processing Destination Editor accesses all the dimensions in the source and displays them as a list for you to select the one you want to process. Next you can choose the processing method from add (incremental), full, or update options. In the Mappings page, you can map the Available Input Columns to the Available Destination Columns using a drag-and-drop operation.

The Advanced page allows you to configure error handling in the dimension processing destination. You can choose from several options to configure the way you want the errors to be handled:

- By default, this destination will use default Analysis Services error handling that you can change by un-checking the Use Default Error Configuration check box.
- When the dimension processing destination processes a dimension to populate values from the underlying columns, an unacceptable key value may be encountered. In such cases, you can use the Key Error Action field to specify that the record be discarded by selecting the DiscardRecord value, or you can convert the unacceptable key value to the UnknownMember value. UnknownMember is a property of the analysis services dimension indicating that the supporting column doesn't have a value.
- Next you can specify the processing error limits and can choose to either ignore errors or stop on error. If you select Stop On Error option, then you can specify the error threshold using the Number Of Errors option. Also, you can specify the on error action either to stop processing or to stop logging when the error threshold is reached by selecting the StopProcessing or StopLogging value.
- You can also specify specific error conditions such as these:
  - When the destination raises an error of Key Not Found, you can select it to be IgnoreError or ReportAndStop, whereas, by default, it is ReportAndContinue.
  - Similarly, you can configure for Duplicate Key error for which default action is to IgnoreError. You can set it to ReportAndStop or ReportAndContinue if you wish.
  - When a null key is converted to the UnknownMember value, you can choose to ReportAndStop or ReportAndContinue. By default, the destination will IgnoreError.

- When a null key value is not allowed in data, this destination will ReportAndContinue by default. However, you can set it to IgnoreError or ReportAndStop.
- You can specify a path for the error log using the Browse button.

Excel Destination

Using the Excel destination, you can output data straight to an Excel workbook, worksheets, or ranges. You use an Excel Connection Manager to connect to an Excel workbook. Like an Excel Source, the Excel destination treats the worksheets and ranges in an Excel workbook as tables or views. The Excel destination has one regular input and one error output.

This destination has its own custom user interface that you can use to configure its properties; the Advanced Editor can also be used to modify the remaining properties. The Excel Destination Editor lists its properties in three different pages.

In the Connection Manager page, you can select the name of the connection manager from the drop-down list in the OLE DB Connection Manager field. Then you can choose one of these three data access mode options:

- **Table or view** Lets the Excel destination load data in the Excel worksheet or named range; specify the name of the worksheet or the range in the Name Of The Excel Sheet field.
- **Table name or view name variable** Works like the Table Or View option except that the name of the table or view is contained within a variable that you specify in the Variable Name field.
- **SQL command** Allows you to load the results of an SQL statement to an Excel file.

In the Mappings page, you can map Available Input Columns to the Available Destination Columns using a drag-and-drop operation. In the Error Output page you can configure the behavior of the Excel destination for errors and truncations. You can ignore the failure, redirect the data, or fail the component for each of the columns in case of an error or a truncation.

Flat File Destination

Every now and then you may require outputting some data from disparate sources to a text file, as this is the most convenient method to share data with external systems. You can build an Integration Services package to connect to those disparate sources, extract data using customized extraction rules, and output the required data set to a text file

using the flat file destination adapter. This destination requires a Flat File Connection Manager to connect to a text file. When you configure a Flat File Connection Manager, you also configure various properties to specify the type of the file and how the data will reside in the file. For example, you can choose the format of the file to be delimited, fixed width, or ragged right (also called mixed format). You also specify how the columns and rows will be delimited and the data type of each column. In this way, the Flat File Connection Manager provides a basic structure to the file, which the destination adapter uses as is. This destination has one output and no error output.

The Flat File destination has a simple customized user interface, though you can also use the Advanced Editor to configure some of the properties. In the Flat File Destination Editor, you can specify the connection manager you want to use for this destination in the Flat File Connection Manager field and select the check box for "Overwrite data in the file" if you want to overwrite the existing data in the flat file. Next you are given an opportunity to provide a block of text in the Header field, which can be added before the data as a header to the file. In the Mappings page, you can map Available Input Columns to the Available Destination Columns.

OLE DB Destination

You can use the OLE DB destination when you want to load your transformed data to OLE DB–compliant databases, such as Microsoft SQL Server, Oracle, or Sybase database servers. This destination adapter requires an OLE DB Connection Manager with an appropriate OLE DB provider to connect to the data destination. The OLE DB destination has one regular input and one error output.

This destination adapter has a custom user interface that can be used to configure most of the properties alternatively you can also use the Advanced Editor. In the OLE DB Destination Editor, you can specify an OLE DB connection manager in the Connections Manager page. If you haven't configured an OLE DB Connection Manager in the package yet, you can create a new connection by clicking New. Once you've specified the OLE DB Connection Manager, you can select the data access mode from the drop-down list. Depending on the option you choose, the editor interface changes to collect the relevant information. Here you have five options to choose from:

- **Table or view** You can load data into a table or view in the database specified by OLE DB Connection Manager. Select the table or the view from the drop-down list in the name of the table or the view field. If you don't already have a table in the database where you want to load data, you can create a new table by clicking New. An SQL statement for creating a table is created for you when you click New. The columns use the data type and the length same as that of the input

columns, which you can change if you want. However, if you provide the wrong data type or a shorter column length, you will not be warned and may get errors at run time. If you are happy with the CREATE TABLE statement, all you need to do is provide a table name replacing the [OLE DB Destination] string after CREATE TABLE in the SQL statement.

- **Table or view—fast load** The data is loaded into a table or view as in the preceding option; however, you can configure additional options here when you select fast load data access mode. The additional fast load options are:
  - **Keep identity** During loading, the OLE DB destination needs to know whether it has to keep the identity values coming in the data or it has to assign unique values itself to the columns configured to have identity key.
  - **Keep nulls** Tells the OLE DB destination to keep the null values in the data.
  - **Table lock** Acquires a table lock during bulk load operation to speed up the loading process. This option is selected by default.
  - **Check constraints** Checks the constraints at the destination table during the data loading operation. This option is selected by default.
  - **Rows per batch** Specifies the number of rows in a batch in this box. The loading operation handles the incoming rows in batches and the setting in this box will affect the buffer size. So, you should test out a suitable value for this field based on the memory available to this process during run time on your server.
  - **Maximum insert commit size** You can specify a number in this dialog box to indicate the maximum size that the OLE DB destination handles to commit during loading. The default value of 2147483647 indicates that these many rows are considered in a single batch and they will be handled together—i.e., they will commit or fail as a single batch. Use this setting carefully, taking into consideration how busy your system is and how many rows you want to handle in a single batch. A smaller value means more commits and hence the overall loading will take more time; however, if the server is a transactional server hosting other applications, then this might be a good idea to share resources on the server. However, if the server is a dedicated reporting or data mart server or you're loading at a time when the other activities on the server are less active, then using a higher value in this box will reduce the overall loading time.

Make sure you use fast load data access mode when loading with double-byte character set (DBCS) data; otherwise, you may get corrupted data loaded in your table or view. The DBCS is a set of characters in which each character is represented by two bytes.

The environments using ideographic writing systems such as Japanese, Korean, and Chinese use DBCS, as they contain more characters than can be represented by 256 code points. These double-byte characters are commonly called Unicode characters. Examples of data types that support Unicode data in SQL Server are nchar, nvarchar, and ntext, whereas Integration Services has DT_WSTR and DT_NTEXT data types to support Unicode character strings.

- **Table name or view name variable** This data access mode works like table or view access mode except that in this access mode you supply the name of a variable in the Variable Name field that contains the name of the table or the view.
- **Table name or view name variable—fast load** This data access mode works like table or view—fast load access mode except here you supply the name of a variable in the Variable Name field that contains the name of the table or the view. You still specify the fast load options in this data access mode.
- **SQL command** Load the result set of an SQL statement using this option. You can provide the SQL query in the SQL Command Text dialog box or build a query by clicking Build Query.

In the Mappings page, you can map Available Input Columns to the Available Destination Columns using a drag-and-drop operation, and in the Error Output page, you can specify the behavior when an error occurs.

Partition Processing Destination

The partition processing destination is used to load and process an SQL Server Analysis Services partition and works like a dimension processing destination. This destination has a custom user interface that is like the one for the dimension processing destination. This destination adapter requires the Analysis Services Connection Manager to connect to the cubes and its partitions that reside in an Analysis Services server or the Analysis Services project.

The Partition Processing Destination Editor has three pages to configure properties. In the Connection Manager page, you can specify an Analysis Services Connection Manager and can choose from the three processing methods—Add (incremental) for incremental processing; Full, which is a default option and performs full processing of the partition; and Data only to perform update processing of the partition. In the Mappings page, you can map Available Input Columns to the Available Destination Columns using a drag-and-drop operation. In the Advanced page you can configure error-handling options when various types of errors occur. Error-handling options are similar to those available on the Advanced page of dimension processing destination.

Raw File Destination

Sometimes you may need to stage data in between processes, for which you will want to extract data at the fastest possible speed. For example, if you have multiple packages that work on a data set one after another—i.e., a package needs to export the data at the end of its operation for the next package to continue its work on the data—a raw file destination and raw file source combination can be excellent choices. The raw file destination writes raw data to the destination raw file in an SSIS native form that doesn't require translation. This raw data can be imported back to the system using the raw file source discussed earlier. Using the raw file destination to export and raw file source to import data back into the system results in high performance for the staging or export/import operation. However, if you have binary large object (BLOB) data that needs to be handled in such a fashion, Raw File destination cannot help you, as it doesn't support BLOB objects.

The Raw File Destination Editor has two pages to expose the configurable properties. The Connection Managers page allows you to select an access mode—File name or File name from variable—to specify how the filename information is provided. You can either specify the filename and path in the File Name field directly or you can use a variable to pass these details. Note that the Raw File destination doesn't use a connection manager to connect to the raw file and hence you don't specify a connection manager in this page; it connects to the raw file directly using the specified filename or by reading the filename from a variable.

Next, you can choose from the following four options to write data to a file in the Write Option field:

- **Append** Lets you use an existing file and append data to the already existing data. This option requires that the metadata of the appended data match the metadata of the existing data in the file.
- **Create Always** This is a default option and always creates a new file using the filename details provided either directly in the File Name field or indirectly in a variable specified in the Variable Name field.
- **Create Once** In the situations where you are using the data flow inside a repeating logic—i.e., inside a loop container—you may want to create a new file in the first iteration of the loop and then append the data to the file in the second and higher iterations. You can achieve this requirement by using this option.
- **Truncate And Append** If you've an existing raw file that you want to use to write the data into, but want to delete the existing data before the new data is written into it, you can use this option to truncate the existing file first and then append the data to this file.

In all these options, wherever you use an existing file, the metadata of the data being loaded to the destination must match with the metadata of the file specified.

In the Columns tab, you can select the columns you want to write into the raw file and assign them an output alias as well.

Recordset Destination

Sometimes you may need to take a record set from the data flow to pass it over to other elements in the package. Of course, in this instance you do not want to write to an external storage and then read from it unnecessarily. You can achieve this by using a variable and the recordset destination that populates an in-memory ADO record set to the variable at run time.

This destination adapter doesn't have its own custom user interface but uses the Advanced Editor to expose its properties. When you double-click this destination, the Advanced Editor for Recordset destination opens and displays properties organized in three tabs. In the Component Properties tab, you can specify the name of the variable to hold the record set in the Variable Name field. In the Input Columns tab, you can select the columns you want to extract out to the variable and assign an alias to each of the selected column along with specifying whether this is a read-only or a read-write column. As this source has only one input and no error output, the Input And Output Properties tab lists only the input columns.

Script Component Destination

You can use the script component as a data flow destination when you choose Destination in the Select Script Component Type dialog box. On being deployed as a destination, this component supports only one input and no output, as you know data flow destinations don't have an output. The script component as a destination is covered in Chapter 11.

SQL Server Compact Destination

Integration Services stretches out to give you an SQL Server Compact destination, enabling your packages to write data straight to an SQL Server Compact database table. This destination uses the SQL Server Compact Connection Manager to connect to an SQL Server Compact database. The SQL Server Compact Connection Manager lets your package connect to a compact database file, and then you can specify the table you want to update in an SQL Server Compact destination.

You need to create an SQL Server Compact Connection Manager before you can configure an SQL Server Compact destination. This destination does not have a custom user interface and hence uses the Advanced Editor to expose its properties. When you double-click this destination, the Advanced Editor for SQL Server

Compact destination opens with four tabs. Choose the connection manager for a Compact database in the Connection Manager tab. Specify the table name you want to update in the Table Name field under the Custom Properties section of the Component Properties tab.

In the Column Mappings tab, you can map Available Input Columns to the Available Destination Columns using a drag-and-drop operation. The Input and Output Properties tab shows you the External Columns and Input Columns in the Input Collection and the Output Columns in the Error Output Collection. SQL Server Compact destination has one input and supports an error output.

SQL Server Destination

We have looked at two different ways to import data into SQL Server—using the Bulk Insert Task in Chapter 5 and the OLE DB destination earlier in this chapter. Though both are capable of importing data into SQL Server, they suffer from some limitations. The Bulk Insert task is a faster way to import data but is a part of the control flow, not the data flow, and doesn't let you transform data before import. The OLE DB destination is part of the data flow and lets you transform the data before import; however, it isn't the fastest method to import data into SQL Server. The SQL Server destination combines benefits of both the components—it lets you transform the data before import and use the speed of the Bulk Insert task to import data into local SQL Server tables and views. The SQL Server destination can write data into a local SQL Server only. So, if you want to import data faster to an SQL Server table or a view on the same server where the package is running, use an SQL Server destination rather than an OLE DB destination. Being a destination adapter, this has one input only and does not support an error output.

SQL Server destination has a custom user interface, though you can also use the Advanced Editor to configure its properties. In the Connection Manager page of the SQL Destination Editor, you can specify a connection manager, a data source, or a data source view in the Connection Manager field to connect to an SQL Server database. Then select a table or view from the drop-down list in the Use A Table Or View field. You also have an option to create a new connection manager or a table or view by clicking the New buttons provided. In the Mappings page, you can map Available Input Columns to the Available Destination Columns using a drag-and-drop operation.

You specify the Bulk Insert options in the Advanced page of the SQL Destination Editor dialog box. You can configure the following ten options in this page:

- **Keep identity** This option is not checked by default. Check this box to keep the identity values coming in the data rather than using the unique values assigned by SQL Server.

- **Keep nulls** This option is not checked by default. Check this box to retain the null values.
- **Table lock** This option is checked by default. Uncheck this option if you don't want to lock the table during loading time. This option may impact the availability of tables being loaded to other applications or users. If you want to allow concurrent use of SQL Server tables that are being loaded by this destination, uncheck this box; however, if you are running this package at a quiet time—i.e., when no other applications or users are accessing the tables being loaded, or you do not want to allow concurrent use of those tables—it is better to leave the default setting.
- **Check constraints** This option is checked by default. This means any constraint on the table being loaded will be checked during loading time. If you're confident the data being loaded does not break any constraints and want faster import of data, you may uncheck this box to save processing overhead of checking constraints.
- **Fire triggers** This option is not checked by default. Check this box to let the bulk insert operation execute insert triggers on target tables during loading. Selecting to execute insert triggers on the destination table may affect the performance of the loading operation.
- **First row** Specify a value for the first row from which the bulk insert will start.
- **Last row** Specify a value in this field for the last row to insert.
- **Maximum number of errors** Provide a value for the maximum number of rows that cannot be imported due to errors in data before the bulk insert operation stops. Leave the First Row, Last Row, and Maximum Number Of Errors fields blank to indicate that you do not want to specify any limits. However, if you're using the Advanced Editor, use a –1 value to indicate the same.
- **Timeout** Specify the number of seconds in this field before the bulk insert operation times out.
- **Order columns** Specify a comma-delimited list of columns in this field to sort data on in ascending or descending order.

Data Flow Paths

First, think of how you connect tasks in the control flow. You click the first task in the control flow to highlight the task and display a green arrow, representing output from the task. Then you drag the green arrow onto the next task in the work flow to create a connection between the tasks, represented by the green line by default. The green line, called a *precedence constraint,* enables you to define some conditions when the following tasks can be executed. In the data flow, you connect the components in the same way you

did in the control flow—drag the output of a data flow component and drop it onto the input of the next component, and the line formed connecting the data flow components on the Data Flow Designer surface is called the *data flow path.* They may look similar on the Designer, but there are major differences between a precedence constraint and a data flow path, as both represent different functionalities in their own right.

Note that the data flow path line is thinner than the precedence constraint line and can be either green or red, depending on whether it is representing Output Path or Error Output Path. In the Control Flow, when you connect a task to another using a precedence constraint and click the task again, you will see another green arrow, indicating that you can configure multiple precedence constraints for the tasks; However, in the data flow the data flow paths are limited to the number of outputs and error outputs available to the source component. There's another important difference—the data flow path actually simulates a pipe connecting the pipeline components in the data flow path (remember that *data flow* is also known as *pipeline*) through which data flows, whereas a precedence constraint specifies a condition when the next task can be executed in the workflow.

When you click a component, for example OLE DB source, in the Data Flow Designer surface, depending upon the outputs available from the component, you may see a combination of green and red arrows. Some components have both output and error output paths; some have only one and some have no output, such as destinations. Our example component, OLE DB source, has both output and error output available and hence shows both green and red arrows. After you connect a component to another component using a data flow path on the Data Flow Designer, you can configure the properties of the data flow path using Data Flow Path Editor. This editor can be opened by choosing the Edit command from the context menu or simply by double-clicking the path. Once in the Data Flow Path Editor, you will be able to configure properties such as name, description, and annotation of the path on the General page; you can see the metadata of the data columns flowing through the path on the Metadata page; and you can add data viewers on the Data Viewers page. We will configure the properties of the data flow path in the following Hands-On exercise.

Hands-On: An Introduction to the Data Flow Task

The purpose of this Hands-On exercise is to introduce you to the Data Flow task and how you can monitor the data flowing through the package by exporting data from [Person].[Contact] table of AdventureWorks database to a flat file.

Method

We will not do much research in this package but will keep it simple as this is just an introduction to data flow. You will drop a Data Flow task on the control flow and then

go on to configure this Data Flow task. As you know by now that the Data Flow task has its own development and designer environment in BIDS, which opens up when you double-click the Data Flow task or by clicking the Data Flow tab.

Exercise (Configure an OLE DB Connection Manager and Add a Data Flow Task)

To begin this exercise, create a new package, configure a connection manager for connecting to the AdventureWorks database, and then add a Data Flow task to the package.

1. Start BIDS. Create a New Project with the following details:

Template	Integration Services Project
Name	Introduction to Data Flow
Location	C:\SSIS\Projects

2. When the blank solution is created, go to Solution Explorer and rename the package.dtsx file to **My First Data Flow.dtsx.**
3. As we will be exporting data from Adventure Works database, we need to have a connection manager to establish a connection to the database. Right-click anywhere in the Connection Managers area and choose New OLE DB Connection from the context menu. In the Configure OLE DB Connection Manager dialog box, click New to specify settings for the Connection Manager dialog box. Specify localhost or your computer name in the Server Name field and leave the Use Windows Authentication radio button selected. Choose the AdventureWorks database from the drop-down list in the Select Or Enter A Database Name field. Test the connection to the database using the Test Connection button before closing the open windows by clicking OK twice.
4. Go to the Toolbox; drag and drop the Data Flow task onto the Control Flow Designer surface. Right-click the Data Flow task and choose Rename from the context menu. Rename the Data Flow task as **Export PersonContact**.
5. Double-click Export PersonContact and you will be taken to the Data Flow tab, where the Data Flow Task field displays the currently selected task: Export PersonContact. Using this field, you can select the required Data Flow task from the drop-down list when your package has multiple Data Flow tasks.
6. Go to the Toolbox, and you will notice that the available list of tasks in the Toolbox has changed. The Data Flow tab has a different set of Integration Services components that are designed to handle data flow operations and are divided into three sections: Data Flow sources, Data Flow transformations, and Data Flow destinations. See the list of components available under each section.

Exercise (Add an OLE DB Source and a Flat File Data Flow Destination)

Now you can build your first data flow using an OLE DB source and a Flat File destination.

7. From the Data Flow Sources section in the Toolbox, drag and drop the OLE DB Source onto the Data Flow Designer surface. Double-click the OLE DB source to open the OLE DB Source Editor. You will see that the OLE DB Connection Manager field has automatically picked up the already configured connection manager. Expand the list of Data Access Mode to see the available options. Leave the Table Or View option selected.
8. When you click in the name of the table or the view field, the Data Flow source goes out using the connection manager settings to display you a list of tables and views. Select [Person].[Contact] table from the list. Click Preview to see the first 200 rows from the selected table. Once done, close the preview window.
9. Click the Columns page from the left pane of the editor window. Note that all the external columns have been automatically selected. Uncheck the last five columns—PasswordHash, PasswordSalt, AdditionalContactInfo, rowguid, and ModifiedDate—as we do not want to output these columns. The Output Column shows the names given to the output columns of OLE DB source, though you can change these names if you wish to do so (see Figure 9-6).
10. Go to the Error Output page and note that the default setting for any error or truncation in data for each column is to fail the component. This is fine for the time being. Click OK to close the OLE DB Source Editor.
11. Right-click the OLE DB source and choose the Show Advanced Editor context menu command. This will open the Advanced Editor dialog box for OLE DB source, in which you can see its properties exposed in four different tabs. The Connection Managers tab shows the connection manager you configured in earlier steps. Click the Component Properties tab and specify the following properties:

Name	Person_Contact
Description	OLE DB source fetching data from [Person].[Contact] table of AdventureWorks database.

Go to the Column Mappings tab to see the mappings of the columns from Available External Columns to Available Output columns. Go to the Input and Output Properties tab and expand the outputs listed there. You will see the External Columns, Output Columns, and the Error Output Columns. If you click a column, you will see the properties of the column in the right side. Depending upon the category of the column you're viewing, you will see different

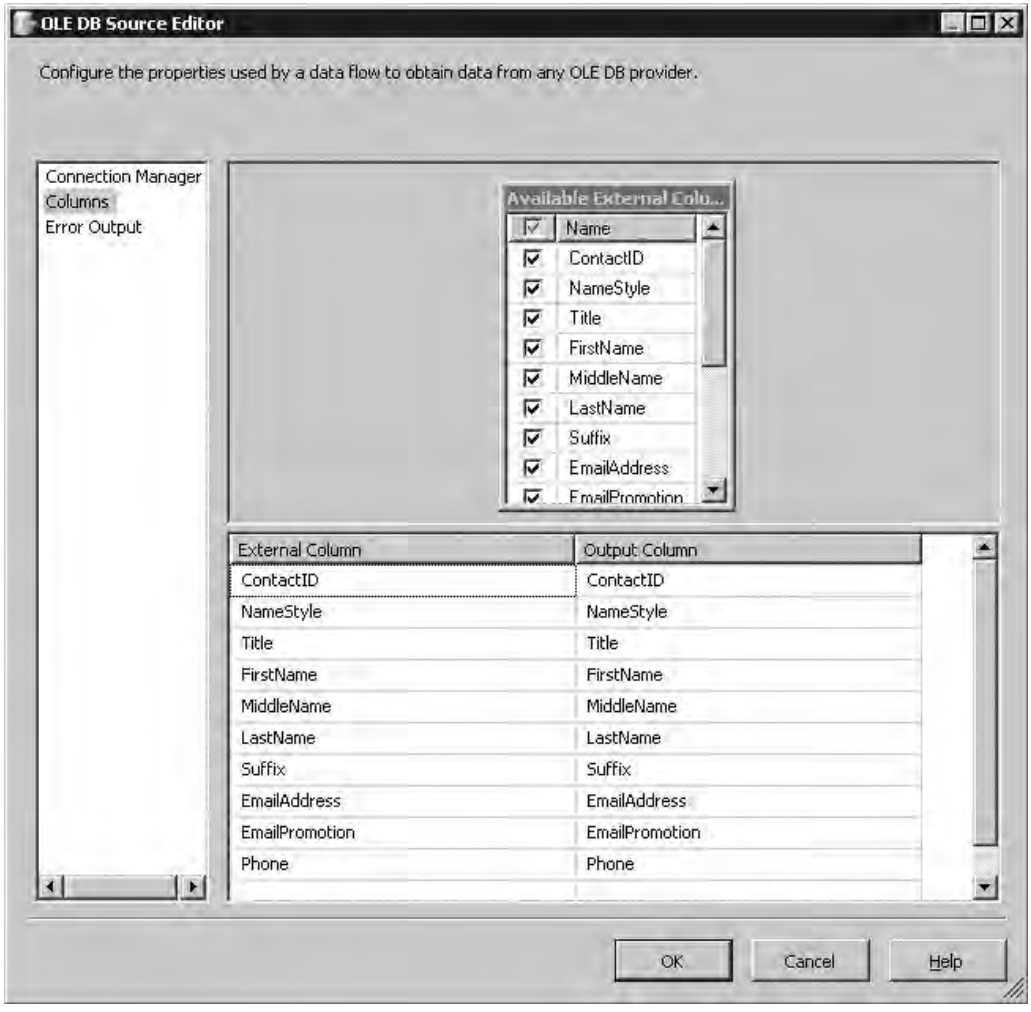


Figure 9-6 *You can select the external columns and assign output names for them.*

levels and types of properties that you may be able to change as well. Click OK to close the Advanced Editor and you will see the OLE DB source has been renamed. If you hover your mouse over the OLE DB source, you will see the description appear as a screen tip. Make a habit to clearly define the name and description properties of the Integration Services components, as this helps in self-documenting the package and goes a long way in reminding you what this component does, especially when you open a package after several months to modify some details.

12. Go to the Toolbox and scroll down to the Data Flow destinations section. Drag and drop the Flat File destination from the Toolbox onto the Designer surface just below the Person_Contact OLE DB source. Click the Person_Contact and you will see a green arrow and a red arrow emerging from the source. Drag the green arrow over to the Flat File Destination to connect the components together.
13. Double-click the Flat File destination to invoke the Flat File Destination Editor. In the Connection Manager page, click the New button shown opposite the Flat File Connection Manager field. You will be asked to choose a format for the flat file to which you want to output data. Select the Delimited radio button if it is not already selected and click OK. This will open a Flat File Connection Manager Editor dialog box. Specify C:\SSIS\RawFiles\PersonContact.txt in the File Name field and check the box for the Column names in the first data row option. All other fields will be filled in automatically for you with default values. Click the Columns page from the left pane of the dialog box and see that all the columns you've selected in the OLE DB source have been added. This list of columns is actually taken by the Flat File Destination's input from the output columns of OLE DB source. If you go to the Advanced page, you will see the available columns and their properties.

    Click OK to add this newly configured connection manager to the Flat File Connection Manager field of the Flat File Destination Editor dialog box. Leave the Overwrite Data In The File option checked.
14. Go to the Mappings page to review the mappings between Available Input Columns and Available Destination Columns. Click OK to close the Flat File destination. Rename the Flat File destination to **PersonContact Flat File**.

Exercise (Configure the Data Flow Path and Execute the Package)

In this part, you will configure the Data Flow path that you've used to connect the two components in the last exercise to view the flow of data at run time.

15. Double-click the green line connecting the Person_Contact and PersonContact Flat File components to open the Data Flow Path Editor. In the General page of the editor, you can specify a unique Name for the path, type in a Description, and annotate the path. The PathAnnotation provides four options for annotation: Never for disabling path annotation, AsNeeded for enabling annotation, SourceName to annotate using the value of Source Name field, and PathName to annotate using the value specified in Name field.
16. The Metadata page of the Data Flow Path Editor shows you the metadata of the data flowing through it. You can see the name, data type, precision, scale, length, code page, sort key position, comparison flags, and source component of each column. The source component is the name of component that generated the column. You can also copy this metadata to the clipboard if needed.

17. In the Data Viewers page you can add data viewers to see the actual data that is flowing through the data flow path. This is an excellent debugging tool, especially when you're trying to find out what happened to the data. Let's add a data viewer. Click Add to configure a data viewer. In the General tab of the Configure Data Viewer dialog box, choose how you want to view the data by selecting from Grid, Histogram, Scatter Plot (x,y), and Column Chart types of the data viewers. Depending upon your choice of data viewer type, the second tab is changed appropriately.

    - **Grid** Shows the data columns and rows in a grid. You can select the data columns to be included in the grid in the Grid tab.
    - **Histogram** Select a numerical column in the Histogram tab to model the histogram when you choose this data viewer type.
    - **Scatter Plot (x,y)** Select this option and the second tab changes to Scatter Plot (x,y), in which you can select a numerical column each for the x-axis and the y-axis. The two columns that you select here will be plotted against each other to draw a point for each record on the Scatter Plot.
    - **Column Chart** Visualize the data as column charts of counts of distinct data values. For example, if you are dealing with persons and use city as a column in the data, then the Column Chart can show the number of persons for each city drawn as columns on the chart.

    For our exercise, choose Grid as a data viewer type and leave all the columns selected in the Grid tab. Click OK to return to the Data Flow Path Editor dialog box, where you will see a grid-type data viewer has been added in the Data Viewers list. Click OK to close this editor and you will see a Data Viewer icon alongside the Data Flow path on the Designer.

18. The package configuration is complete now, but before we execute the package, it is worth exploring two of the properties of the data flow engine that affect the flow of data buffer by buffer through it. Right-click anywhere on the Data Flow Designer surface and choose Properties. Scroll down to the Misc section in the Properties window and note the following two listed properties:

DefaultBufferMaxRows	10000
DefaultBufferSize	10485760

These properties define the default size of the buffer as 10MB and the maximum rows that a buffer can contain by default as 10,000. These settings give you control to optimize the flow of data through the pipeline.

19. Press the F5 key on the keyboard to execute the package. As the package starts executing you will see a Grid Data Viewer window. As the package executes and starts sending data down the pipeline, the data viewer gets attached to the data flow and shows the data in the buffer flowing between the two components. If you look on the status bar at the bottom of the data viewer window, you can see the counts of the total number of rows that have passed through the data viewer, the total number of buffers when the viewer is detached, and the rows displayed in this buffer. On the top of the data viewer window, you can see three buttons: Copy Data allows you to copy the data currently shown in the data viewer to the Clipboard, Detach toggles to Attach when clicked and allows you to detach the data viewer from the data flow and lets the data continue to flow through the path without being paused, and the green arrow button allows you to move data through the data flow buffer by buffer. When the package is executed, the data is moved in the chunk sizes (buffer by buffer) limited by the default buffer size and the default buffer maximum rows, 10MB and 10,000 by default. Clicking this green arrow button will allow the data in the first buffer to pass through and the data in the second buffer will be held up for you to view (see Figure 9-7). Click the green arrow to see the data in the next buffer.

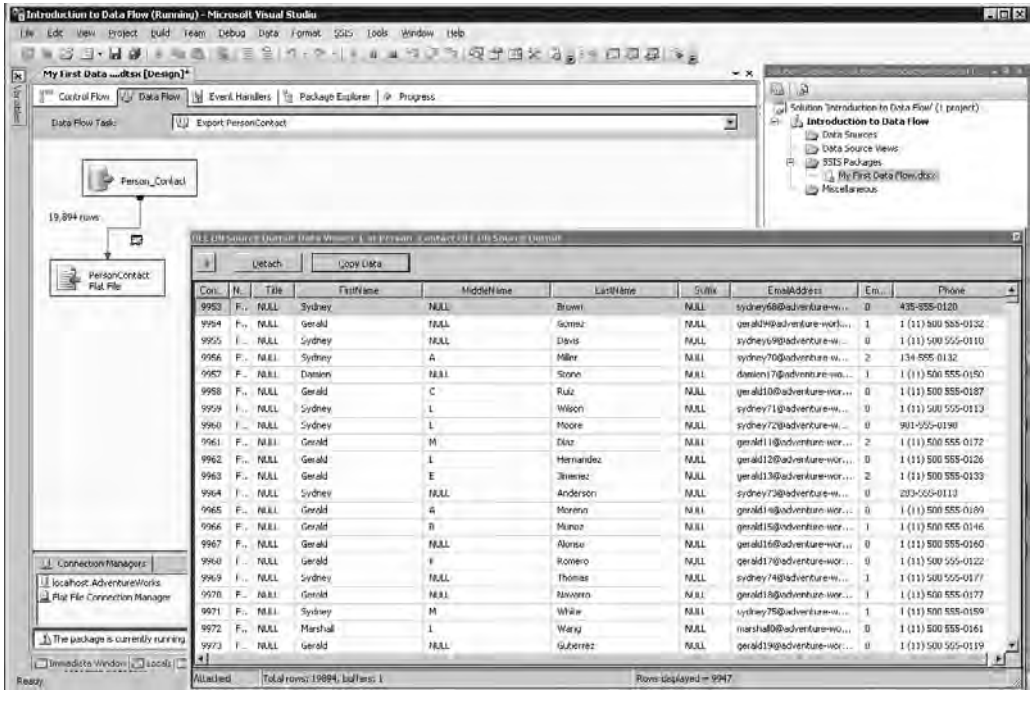


Figure 9-7 *Data Viewer showing the data flow in the grid*

20. After a couple of seconds, you will see the data in the next buffer. This time the total rows will be shown at a little less than 20,000 and the rows displayed will be a little less than 10,000. The total number of rows is also shown next to the Data Flow path on the Designer surface. This number of rows may vary for a different data flow depending upon the width of the rows. Click Detach to complete the execution of the package.
21. Press SHIFT-F5 to stop debugging the package. Press CTRL-SHIFT-S to save all the items in this project.

Review

In this exercise, you built a basic data flow for a package to extract data from a database table to a flat file. You've also used the Data Viewer to see the data flowing past and learned how to optimize the data buffer settings to fine-tune the data flow buffer by buffer through the data flow.

Summary

You are now familiar with the components of data flow and know how the data is being accessed from the external source by the data flow source, passes through the data flow transformations, and then gets loaded into the data flow destinations. You have studied the data flow sources, data flow destinations, and data flow path in detail in this chapter and have briefly learned about the data flow transformations. In the next chapter, you will learn more about data flow transformations by studying them in detail and doing Hands-On exercises using most of the transformations.

This page intentionally left blank

Chapter 10

Data Flow Transformations

In This Chapter

- Row Transformations
- Split and Join Transformations
- Rowset Transformations
- Audit Transformations
- Business Intelligence Transformations
- Summary

Most of the business requirements will need you to work with data and make it suitable for deriving some business value out of it. The data and schema are modified in Integration Services using data flow transformations. In this chapter, you will explore the preconfigured transformations in detail. You will also be working with Hands-On exercises that will cover most of the transformations from the perspective of their usability. As you progress in the chapter, the Hands-On exercises will include more and more complex transformations and business scenarios, and of course they will be more interesting as well. To keep such a progression and your interest in the chapter, the transformations may not appear in the order in which they appeared in Chapter 9. We'll start with the easiest ones first.

Row Transformations

Row transformations are simple compared to other types of transformations. Starting with the simplest transformation—i.e., Copy Column transformation—to copy an input column to output columns, we will study the Character Map transformation, the Data Conversion transformation, and then the Data Derivation transformation. The next two transformations deal with columnar data—the Export Column transformation exports partial data from the pipeline into a file, and the Import Column transformation reads data from a file and adds it into the pipeline. The ability to perform more complex functions will increase as we go till we reach the last two transformations, Script Component and OLE DB Command, in this category, which can be programmed to perform the functions you want. Let's start our journey with Copy Column transformation.

Copy Column Transformation

The Copy Column transformation is probably the simplest of the transformations available in the SQL Server Integration Services. It does precisely what it says—it copies a column in the data flow to add a new column in the data flow. Sometimes you may need to output a column two times for other applications, and as SSIS does not provide the flexibility of mapping a column twice in the destination adapters, you would use this transformation to fill in. This new column then can be given a name within the transformation. To perform the desired function, this transformation has been designed to support one input and one output.

The Copy Column transformation has a custom user interface defined by the Copy Column Transformation Editor. In the Copy Column Transformation Editor, you can select the columns you want to copy either from the upper half by checking the boxes in front of columns in the list under Available Input Columns or select from the drop-down list provided in the Input column. You can select multiple columns to be copied or choose to create multiple copies of a column in the same transformation. The selected columns will be copied and added as new columns to the data flow output

columns of this transformation. When you select a column in the Input column, the Output Alias shows the Input column name prefixed with a *Copy Of* string by default. You can change this alias and specify a more appropriate name for the column. As this transformation does not cause any changes to the data schema or data quality, this transformation doesn't introduce any errors in the data flow and hence doesn't have error output.

Character Map Transformation

You use this transformation to apply string functions to the string data type columns. Configuring this transformation and applying any of the following string functions to one of the columns are simple operations. The converted data can then either be populated in a new column or it can perform an in-place data conversion. An *in-place change* means that the data will be modified in the existing column and no new column will be added to the data flow. The string operation you specify is applied row by row as the data flows through this transformation. To perform a function, this transformation supports one input, one output, and one error output.

To add the Character Map transformation to the data flow and open the editor UI, you need to drop this transformation in your data flow task and join with an upstream component before you double-click it. The user interface in the Character Map Transformation Editor is simple and intuitive. In the upper part, a list provides the Available Input Columns with a check box in front of each column. Select a column by clicking the check box, and a line is added in the lower half of the window with the selected column shown in the Input column. Alternatively, you can select a column for applying string operations by clicking in the Input column and selecting a column from the drop-down list. Note that the Character Map transformation can be used only for columns with string data types. If you configure this transformation with a column that has a data type other than string, validation errors will occur with the error message stating that the input column has an unsupported data type. The lower half of the transformation UI has four columns.

- **Input Column** Here you can select the column you want to apply Character Map transformations.
- **Destination** The Destination column allows you to specify whether you want the modified data to be copied into a New Column, which is a default option, or treated for an in-place change. You select either option by clicking in the column and choosing it from the drop-down list. When you select the New Column value, the Output Alias field shows Column Name prefixed with *Copy Of* for the new column, and when you choose In-Place Change, the Output Alias shows Column Name as is.

- **Operation** The Operation column allows you to select from any of the string operations listed in the following table. You can select multiple operations by clicking the multiple check boxes in the list; the Operation column will show the selected operations in a comma-delimited list. However, there are some restrictions on the selection of multiple operations on the same column—for example, you cannot select Uppercase and Lowercase operations to be applied on the same column in one transformation. Refer to Microsoft SQL Server 2008 Books Online for more details on these restrictions.

Operation	Description
Lowercase	Convert characters of a column to lowercase.
Uppercase	Convert characters of a column to uppercase.
Byte reversal	Reverses the byte order of a column data.
Hiragana	Convert katakana characters of a column data to hiragana characters.
Katakana	Convert hiragana characters of a column data to katakana characters. Both hiragana and katakana are part of Japanese syllabary.
Half width	Convert full-width characters of a column data to half-width characters.
Full width	Convert half-width characters of a column data to full-width characters.
Linguistic casing	Apply linguistic casing instead of the system rules.
Simplified Chinese	Convert traditional Chinese characters of a column data to simplified Chinese characters.
Traditional Chinese	Convert simplified Chinese characters of column data to traditional Chinese characters.

- **Output Alias** This column is used to specify a name to the newly added data column in the data flow.

You can specify the error handling behavior of this transformation by clicking the Configure Error Output button. As this component applies string transformations to the text columns, truncations can occur in the transformed data. For example, when you convert Half Width data to Full Width data, the column length may not support that and can cause the data to be truncated. In such cases, you can use the Configure Error Output button to specify the action that this component should take. When you click this button, the Configure Error Output dialog box opens, where you can specify whether errors should fail the component, be ignored, or redirect the failing rows to the error output for each of the input or output columns that you've selected or created in the transformation.

Data Conversion Transformation

When data is coming from disparate sources into a data warehouse or a data mart, data type mismatches can occur during the data upload process. One of the functions performed during data loading to a data warehouse or a data mart is to convert data type of the column to the one that matches the data type of the destination column. This transformation supports one input, one output, and one error output to perform its functions.

When the data is read from the data source, depending on the data and the data source, a data type is assigned to the Input column, which may not be exactly what you want. For example, a date column read from a text file is generally assigned a string data type that needs to be converted into a date time data type before loading into a destination expecting a date time data type. This problem was handled in DTS 2000 using CAST or CONVERT functions of T-SQL within the Execute SQL task. SSIS provides Data Conversion transformation as a preconfigured task to perform this operation, though you still can write T-SQL code using OLE DB Command transformation within the data flow.

The Data Conversion transformation allows you to convert the data type of an input column to a different data type and add the converted data to a new output column. During data conversion, the data of the input column is parsed and then converted into a new Integration Services data type before being written to a new output column. By now you know that Integration Services provides two types of parsing routines—locale-insensitive fast parsing and the locale-sensitive standard parsing routine. You can specify the parsing routines for each of the output columns in the Advanced Editor for Data Conversion. To specify the parsing method, open the Advanced Editor for Data Conversion, go to the Input And Output Properties tab, click the appropriate output column by expanding the Output Columns folder under Data Conversion Output. Go to the Custom Properties section of the selected output column and choose between the False and True values for the FastParse option.

In the Data Conversion Transformation Editor, you can select the input columns that you want to convert to a different data type in the input column or select check boxes in the list of Available Input Columns. You can select multiple columns for conversion or apply multiple conversions to the same column. As the converted data is copied into a new output column, you can define an output alias for the newly created column in the data flow. In the Data Type column, you can specify the data type to which you want to convert the selected input column. The Data Type column helps you to select a data type for the column from the drop-down list of available Integration Services data types. Depending on the data type you select, you can set other attributes of the new data type such as length, precision, scale, and code page. While specifying the code page for the new string data type output column, you must keep the code

page for the new output column the same as the input column code page when you are converting data between string data type columns.

As Data Conversion transformation creates new columns with modified data types and modifies the data flow, error and truncation occurrences in data are possible. To handle these, this transformation provides an error-handling mechanism via the Configure Error Output button. In the Configure Error Output dialog box, you will find a list of all the output columns that you have created in this transformation. You can specify whether to fail the component, ignore the error, or redirect the failing row against error and truncation for each of the column.

Derived Column Transformation

The Derived Column transformation enables you to perform more complex derivations on an input column data. Until now you've been performing simple operations on the data, such as copying a column, applying string transformations using Character Map transformation, or even changing data types. With this transformation in your toolkit, you can derive your columns using complex derivation rules. Using this transformation, you will be able to perform operations such as deriving sensible information from the Notes or Comments column, concatenating two text strings using two columns or a column and a variable to populate in a single column, performing mathematical operations on a numerical column, using conditional derivations, and much more. This transformation is designed to have one input, one regular output, and one error output.

As this transformation will be making changes to the data and the data schema, an error or a truncation can occur in the data. To handle such an instance, this transformation provides a Configure Error Output button to invoke the Configure Error Output dialog box, where you can specify to fail the transformation, ignore the error, or redirect the failing rows to an error output for each of the derived column for errors and for truncations.

This is a very versatile transformation within data flow task that is repeatedly used within your packages and is also used in other components and transformations. For instance, the transformations you have studied so far are special applications of a derived column transformation, though with a slightly different UI. The user interface of a derived column transformation resembles the Expression Builder dialog box. In the Derived Column field in the lower half of the window, you can either select the Add As New Column option to add the derived data into a new column or choose to replace an existing column. Depending on your choice, the Derived Column Name field will be filled in with a default name that you can change to a more appropriate name.

The Expression field is the most important field in the Derived Column Transformation Editor. Here you specify the expression to derive the data. You can use any combination of

variables, input columns, mathematical functions, string functions, date/time functions, null functions, type casts, and operators to build this expression. After specifying an expression to derive data, you can specify the data type, length, precision, scale, and code page for the data. Once you have successfully configured one row in the Expression Builder, you can add more rows to derive additional columns and hence can perform multiple derivations using the same transformation. The left-upper half of the window lists variables and input columns that you can drag and drop in the Expression field. Similarly, the right-upper half of the window lists functions and operators that can also be embedded in the expression simply by dragging and dropping.

There are six categories of such functions and operators.

- **Mathematical Functions** Allow you to perform mathematical operations such as returning absolute positive value with ABS, rounding a numeric value to the closest integer value using ROUND, or calculating the square of a numeric value using SQUARE.
- **String Functions** Allow you to perform string operations such as FINSTRING, SUBSTRING, TRIM, LOWER, UPPER, LEN, and RIGHT.
- **Date/Time Functions** Allow you to manipulate date and time values using functions such as DAY, MONTH, YEAR, DATEADD, DATEPART, and DATEDIFF.
- **Null Functions** Allow you to find whether an expression is a null using the ISNULL function or return a null value of specified data type using functions such as NULL(DT_WSTR, <length>).
- **Type Casts** Allow you to cast your expressions in a specified data type; for instance, the (DT_I4) function will convert the result of your expression into an integer with length 4.
- **Operators** Allow you to perform various types of operations on the expressions, including add, subtract, multiply, divide, concatenate, equal, greater than or equal to, logical and, logical or, and conditional operation. You will be using these versatile operators quite frequently, so make sure you acquaint yourself with all the available operators.

While configuring the Derived Column transformation is not difficult, writing expressions for derivations of the data could be challenging sometimes, especially for a newbie. So, here are some of the cookies for you to get started quickly. These examples

are only for you to practice and do not cover most of the instances you might come across in real life; however, you might find them a good starting point.

- To trim a column CustomerName, you can use either TRIM([CustomerName]) or RTRIM(LTRIM([CustomerName]))
- The following expression returns the first position of the semicolon (;) in the string.

  ```
  FINDSTRING("abcd;cc;c",";",1)
  ```

- You can combine functions with other functions to solve complex issues. For example, using SUBSTRING with FINDSTRING functions, you can truncate the preceding string up to the characters till the first semicolon.

  ```
  SUBSTRING("abcd;cc;c",1,FINDSTRING("abcd;cc;c",";",1) - 1)
  ```

- In the following example, you use simple operators to derive PercentageProfit column and round off the result to a precision of 2.

  ```
  ROUND(((SalePrice – CostPrice) * 100) / CostPrice, 2)
  ```

- In this example, you are working on MiddleName column. When you get a null value in this column and do not want to pass it through but prefer to replace null with a blank string value, you would use the IsNull function along with a conditional operator to return a blank string.

  ```
  IsNull(MiddleName) ? " " : MiddleName
  ```

  The conditional operator evaluates a Boolean condition, which is on the left side of this expression—i.e., before the question mark sign—IsNull(MiddleName—and if true it returns the first value after the question mark sign, or if the Boolean evaluates to false, it returns the second value, which is shown after the colon in the preceding expression. Another similar example is with numeric data when DamageCost contains null, but you want to pass a 0 instead.

  ```
  ISNULL(DamageCost) ? 0 : DamageCost
  ```

  You can build quite complex expressions using this conditional operator, as you can also nest conditional operations as well.

- This example shows use of a conditional operator along with an OR operator to create a bit more complex Boolean condition. Here in the Country field you get two-digit country codes, which are okay for all other countries other than the UK, for which two values exist in data: UK and GB. You want to pass only UK if the country code is either UK or GB and let other codes pass through as is.

  ```
  Country == "UK" || Country == "GB" ? "UK" : Country
  ```

- Last is again an example of conditional operator, but this time it is using dates. Here you return a 1900-01-01 date in the date time format if EndDate is null; else, you let the date time pass through as is.

  ```
  ISNULL(EndDate) ? (DT_DBTIMESTAMP)"1900-01-01 00:00:00.000" : EndDate
  ```

  As you have used a DT_DBTIMESTAMP type cast here, similarly you can cast a date time into any other required data type format using one of the several date time format type casts.

Export Column Transformation

The Export Column transformation has one input, one output, and one error output. Using this transformation, you can export data from the data flow to an external file. For example, you can export images from the data flow and save them to individual files. To use this transformation, two columns must exist in the data flow: one that contains data—i.e., images—and the second that contains the paths for the files to which you want to save those images. By properly configuring the Export Column transformation, you can save each image in every row to a separate file specified by the column that contains the file path information. When you open the transformation editor after connecting an input, you can select columns that you want to extract in the Extract Column field and select the column in the File Path Column that specify file paths for the extracted data. The Extract Column and the File Path Column fields provide drop-down lists of the columns in the data flow; however, the Extract Column displays only the columns that have the DT_TEXT, DT_NTEXT, or DT_IMAGE data type.

You will be able to select Allow Append and Force Truncate check boxes as well. The Allow Append check box allows the transformation to append data to an existing file, and the Force Truncate check box allows transformation to delete and re-create the file before writing data into it. The editor's user interface doesn't allow you to select both the check boxes simultaneously; however, if you set these properties programmatically, the component fails. At run time, the transformation exports data row by row and writes the exported data to the files. Depending on your selections of Allow Append and Force Truncate options and the existence of files to which you want to save data, the transformation decides the appropriate action. For example, if the file to which data is to be inserted does not exist, the transformation creates a new file and writes the data to it irrespective of your settings of these check boxes. However, if the file to which data is to be inserted already exists, the transformation uses information from the check boxes. In this case, if the Allow Append check box is selected, the transformation opens the file and writes the data at the end of the file, and if the Force Truncate option is checked, the transformation deletes and re-creates the file and writes the data in to the file. If you do not select any of the check boxes and the file exists, a run-time error occurs because you are effectively not letting the transformation append or truncate the file, prohibiting it from writing data to the file.

The last option you may find on the user interface is a check box to Write Byte-Order Mark. A Byte-Order Mark (BOM) is a ZERO-WIDTH NO-BREAK SPACE character used to denote Endianness of a string. This is particularly useful for images saved in the TIFF format that store pixel values as words, and BOM makes a difference in performance. Images stored in JPEG or GIF format are not word-oriented and Byte-Order does not matter. Note that BOM is written only when the data type is DT_NTEXT and data is written to a new file.

Import Column Transformation

As you can expect, the Import Column transformation is the reverse of the Export Column transformation and has one input, one output, and one error output. Using this transformation, you can read data from files and add the data to the columns in the data flow. For example, you can add images using this transformation in the data flow along with the text fields. An input column in the transformation input contains the file paths for the files that contain data that you want to add to the data flow. At run time, this transformation processes each row, gets the file path information from the input column, and then loads the data from it to an output column in the data flow.

This transformation doesn't have its own user interface but uses the Advanced Editor to expose its properties for configurations. When you open the Advanced Editor for Import Column transformation, you can select an input column that contains the file path information in the Input Columns tab. Then you can add an output column into which you want to copy the data in the Input And Output Properties tab. To add an output column, select the Output Columns collection in the Import Column Output node and click Add Column. When this new column is added, select the desired data type for the column. The data type of the output column must be DT_TEXT, DT_NTEXT, or DT_IMAGE. Note the ID of this column and expand Input Columns collection in the Import Column Input node. You will see the column that you've selected in the Input Columns tab earlier. Scroll down in the Custom Properties section to locate the FileDataColumnID property and specify the ID of the new output column in this property. By specifying the ID of the output column in the FileDataColumnID property of the input column, you tell the transformation ID of the column into which you want to receive the imported data. If the data type of the new output column is DT_NTEXT, you can also specify True in the ExpectBOM field if the file is expected to begin with a BOM.

Script Component

You have learned about this Data Flow component in the Data Flow Sources and Destinations. By now you understand that this component can also be used as a

transformation. The basic functionality of this component remains the same—i.e., it enables you to write custom code and include that custom code as a component inside the package. Using the Script component as a transformation, you can write custom code to call functions that are not available in SSIS—i.e., you can call the .NET assembly to use working code that is available outside SSIS; build transformations that are not available in SSIS; or apply multiple transformations with custom code to enhance the performance of your package. You can in fact write custom code to do all the transformations within the same Script component; although you could but probably you shouldn't for simplicity, modularity, and performance reasons. All this is covered in greater detail in Chapter 11.

OLE DB Command Transformation

The Execute SQL task of DTS 2000 is commonly used to perform a variety of functions varying from work flow to data-related operations such as truncating tables and log files; creating or dropping tables and views; and updating, deleting, or inserting data in the database objects. As the control flow and the data flow engines have been separated in Integration Services, so the Execute SQL task has been made available in both the engines of Integration Services to perform the appropriate functions within that engine.

The Execute SQL task provided in the control flow of Integration Services is designed to perform the workflow functions such as truncating tables, creating and dropping tables, and updating variable values using the results of an SQL statement, while the OLE DB Command transformation in the data flow engine provides for performing data-related operations. This transformation allows you to run an SQL statement that can insert, update, or delete rows in a database table. As this transformation is designed to be a Row transformation, the SQL statement is actually run for each row of the data flowing through this transformation. The real power of the OLE DB transformation lies in the ability to run a parameterized query to perform insert, update, or delete operations against each row. This also means that you can use a stored procedure with parameters to run against each row of the data. Though it is easy to use this component, as it processes rows one by one, for a considerable size of rowset, the performance will be noticeably less when compared to alternate set-based operations such as staging and loading.

The OLE DB transformation has one input and one output and also supports one error output; however, it doesn't have a custom user interface and hence uses the Advanced Editor to expose its properties. In the Connection Managers tab of the Advanced Editor for OLE DB Command, you specify the connection to the database that you want to update, delete, or insert into.

In the Component Properties tab, you specify common and custom properties for this component. The properties you can specify are:

- The number of seconds before the command times out in the CommandTimeout field. By default, this field has a value of 0, which indicates an infinite timeout value.
- The code page value in the DefaultCodePage field when the data source is unable to provide the code page information.
- The LocaleID, which can be changed from the default value to any other Windows LocaleID.
- The ValidateExternalMetadata value, which can be changed from the default value of True to False if you do not want your component to be validated during the validation phase.
- The SQLCommand field, where you can type in the SQL statement that this transformation runs for each row of the input data. You can use parameters in your SQL statement and map these parameters to the input columns so that the SQL statement is modifying the data, made available by OLE DB Connection Manager, on the basis of values in the input columns. By default, these parameters are named as Param_0, Param_1, and so forth; and you cannot change these names. However, if you use a stored procedure with sql variables in it, then you can use the variable names that make the mappings of parameters easier. This option has been explained later in the chapter (refer to Figure 10-9).

The mappings between parameters used in SQL statements and input columns are defined in the Column Mappings tab. In this tab, after typing the SQL statement in the SQLCommand field, you will see the Available Destination Columns populated with parameters for you. (You may have to click Refresh to see the parameters.) The automatic population of parameters is dependent on the ability of the OLE DB provider you've specified earlier. For some third-party OLE DB providers that do not support deriving parameter information from the SQL statement, you will need to manually create parameter columns in the External Columns node in OLE DB Command Input by going to Input And Output Properties tab; assigning them names such as Param_0, Param_1, and so on; and specify a value of 1 to the DBParamInfoFlags custom property of the column. You can then manually create mappings between Available Input Columns and Available Output Columns by using the drag-and-drop technique.

Split and Join Transformations

These transformations can create multiple copies of input data, split input data into one or more outputs, merge multiple inputs, or add columns to the pipeline by looking up exact matches in the reference table. After reading through the descriptions for these split and join transformations, you will complete your first Hands-On exercise for this chapter that will cover some of the Row transformations also.

Conditional Split Transformation

Sometimes you will need to work with a particular data set—i.e., a subset—separately from the data that is coming in the pipeline. For example, you may want to apply different business rules to your different types of customers. In such cases, you will need to split data to match the specific criteria into the multiple data sets using conditions that are defined in the Conditional Split transformation. This transformation allows you to create more than one output and assign a condition (filter) to the output for the type of data that can pass through it. Among the outputs, one output has to be a default output for the rows that meet no criteria. When an input data row hits the Conditional Split transformation, it passes the data row through a set of conditions one by one and will route the data row to the output to which it matches the criteria first. Each row can only be diverted to one output, and this output has to be the first one for which the condition evaluates to True. This transformation has one input, one error output, and as you can make out, multiple outputs.

The user interface of the Conditional Split Transformation Editor is similar to that of a property expression. You can select variables and columns from the top-left section and functions and operators from the top-right section of the editor window. You can build an expression using variables, columns, functions, and operators in the Condition field for an output. As you add a condition, an output is created with an order number specified in the Order column and a name is assigned to it in the Output Name field. The order number plays a vital role in routing the rows to outputs, as the row is matched to these conditions in an ascending order and the row is diverted to the output for which the condition becomes true first. Once the row has been diverted, the rest of the conditions are ignored, as a row is always sent only to one output. If a row doesn't meet any condition, in the end it will be sent to the default output of the transformation. The Output Name can be changed to a more appropriate name that suits your data set. This transformation has a mandatory default output built in for you.

The syntax you use to build an expression for a condition uses the expression grammar. For example, if you want to split customers on the basis of countries—

e.g., you want to separate out rows for UK, USA, and rest of the world—your Conditional Split transformation will have three outputs with the following configurations:

Order	Output Name	Condition
1	Case 1	[Country] == "UK"
2	Case 2	[Country] == "USA"

The default output will collect all the rows for which Country is neither UK nor USA.

Multicast Transformation

The Conditional Split transformation enables you to divert a row to a specific output and split a data set. It doesn't allow you to divert a row to more than one output and hence does not create a copy of the data set. However, sometimes you may need to create copies of a data set during run time so that you can apply multiple sets of transformations to the same data. For example, working within the same ETL, you may want to treat your sales data differently while preparing data for balance sheets, than while using it to calculate commission paid to the salespersons. Another example could be when you want to load data to your relational database as well as the analytical reporting data mart, with both having different data models and obviously different loading requirements. In this case, you will create two sets of data, apply different transformations to the data to bring them in line with the data model requirements, and then load the data to the destination database. This transformation has one input and supports multiple outputs to perform multicast operation.

The Multicast transformation creates copies of a data set by sending every row to every output. It does not apply any other transformation to the data other than diverting a data set to more than one output. The transformation is simple to use and doesn't require much to configure. In fact, when you open the Multicast Transformation Editor, you will see nothing to configure with the two blank panes aligned side by side. This is because you configure this transformation by connecting its outputs to inputs of multiple components and not by setting properties on the attributes. On the Data Flow surface, when you click the Multicast transformation after connecting its first output to a downstream component, you will see that it provides another output (a green arrow emerging from the transformation) for you to connect to another downstream component. If you double-click the Multicast transformation after connecting the second output to another downstream component, you will see two outputs listed in the left pane of the editor window, and the right pane shows the properties of the output selected in the left pane. You can modify the name of the output and write a description for it if you want and that's it.

Union All Transformation

This component works similar to the Union All command of T-SQL and combines two or more inputs into a single output. To support this functionality, this transformation has multiple inputs and one output and does not support an error output. During run time, the transformation picks up all the rows from one input, sends them to the output, then picks up all the rows from the second input, sends them to the output after combining them with the rows of the first input, and this goes on until all the inputs have been combined. Note that this transformation can select inputs in any random order based on when the data is available at the inputs. Also, it does not sort the records in any way. All the rows from a particular input will be output together.

On the Data Flow Designer when you connect an input to this transformation, this transformation copies the metadata from this first input to its output so that the columns of this first input are mapped to the output created by the transformation. The columns having matching metadata of any input you connect after the first input are also mapped to the corresponding Output columns. Any column that does not exist in the first input, but is a part of subsequent input, will not be copied in the output columns by the transformation. You must create and map this column manually in the Output Column Name field of the output with the corresponding Input column. If you don't create and map a column that exists in subsequent inputs but not in the first input, that column will be ignored in the output. You can also change mappings in the editor by clicking in the Input field and choosing an option from the drop-down list.

Merge Transformation

As its name suggests, the Merge transformation combines two inputs into a single output but requires that the inputs be sorted and the columns have matching metadata. To perform merge functionality, this transformation has two inputs and one output. It does not have an error output. The Merge transformation can even provide you sorted output by inserting rows from inputs based on their key values. This transformation is similar to the Union All transformation with some key differences:

- The Merge transformation requires sorted inputs, whereas the Union All transformation does not.
- The Merge transformation can combine only two inputs, whereas Union All can combine more than two inputs.
- The Merge transformation can provide sorted output, whereas Union All cannot.

When you add a Merge transformation into a package's data flow, you won't be able to open its editor until you have attached two sorted inputs that have properly defined

sort-key positions for the columns on which they are sorted. The input data sets must be sorted—for example, using sort transformation—which must be indicated by the IsSorted property on the outputs and the SortKeyPosition property of the output columns of the upstream components. These properties are discussed in Chapter 15, where you can refer to Figure 15-4 and Figure 15-5 to see how these properties are used.

Once the sorted inputs are connected to the inputs of this transformation and you are able to open the editor, you will see the output columns mapped to the Input columns of both the inputs. This transformation creates the output by copying the metadata of the first input you connect to it. You can make changes to the mappings. However, the metadata of the mapped columns must match. If the second input has a column that is not in the Output columns list, you can add and map this column manually.

Merge Join Transformation

The Merge Join transformation joins two sorted inputs into one output using a full, left, or inner join. For merging two inputs, this transformation supports two inputs and one output and does not support an error output. The joins work exactly as it does in T-SQL. For the benefit of those who haven't worked with joins, here is a quick refresher for these joins.

Inner Join

An inner join returns the rows that have a join key present in both the data sets. For example, suppose you have a table for all the customers, a table for all the products that your company has sold in last ten years, and a date table. An inner join will return a list of customers and the products who purchased any product in year 2009.

Left Outer Join

A left outer join returns all the rows from the first (left side of the join) table and the rows from second table for the matching key. For the rows that don't have a matching key in the second table, the corresponding output columns are filled with nulls. For the customers and products example, a left outer join will list all the customers with the products listed against the customers who have made purchases in 2009 and nulls against the customers who didn't make a purchase in 2009.

Full Outer Join

A full outer join returns all the rows from both tables joined by the key—simply put, it is a list of all the data. If the rows from first table don't have a matching key in the second table, the corresponding second table columns are filled with nulls. Similarly, if the rows from the second table don't have a matching key in the first table, the corresponding first table columns are filled with nulls.

When you add a merge join transformation into a data flow, you won't be able to open the editor until you have attached two sorted inputs—one to the merge join left input and the other to the merge join right input. The input data sets must be sorted physically—for example, using a sort transformation—which must be indicated by the IsSorted property on the outputs and the SortKeyPosition property of the output columns of the upstream components. These properties are discussed in Chapter 15, where you can refer to Figure 15-4 and Figure 15-5 to see how these properties are used.

Once the sorted inputs have been connected to the left and right inputs of this transformation, you can open the editor. You can choose among inner join, left outer join, or full outer join types in the Join Type field. You can also swap your inputs by clicking the Swap Inputs button for a Left Outer Join. After selecting the join type, you can specify the join keys if they have not already been picked up by the transformation. The joined columns must have the matching metadata, and the join key must be in the same order as specified by the sort key. Next, you can select the output columns by selecting the check boxes or using the drop-down lists in the Input columns in the lower section of the Transformation Editor. You will also be able to assign an alias to the Output column. Once you have selected the required columns for the outputs, you can close the editor, as there is no other configuration in the editor; however, there are couple of important properties that you may configure for this transformation. You can access these special properties in the properties window when you press the F4 key while this transformation is selected.

The first performance-related property is the MaxBuffersPerInput property, which lets you specify an integer value for the number of buffers for each input that you want to suggest. Merge Join is a semi-blocking transformation; that is, it needs a set of rows before it could start outputting any record. You can well imagine this as the transformation needs a good set of rows or buffers for each input to be able to join them and start spitting out the resultant rows. The second reason for this transformation to collect large sets of data is to avoid deadlocking between the threads that are processing data. If you are working with really large data sets, the Merge Join transformation can cache huge amounts of data, putting high pressure on memory available on the computer. This could affect performance of other applications on the server. To avoid such a situation, you can throttle the memory requirements using the MaxBuffersPerInput property. While this property allows you some control, it is just a suggestive value and not a hard value that forces the Merge Join to use only the specified number of buffers. If the transformation feels that there is a risk of thread deadlocking or it doesn't have enough data to perform, it will increase the number of buffers required to some other value than what has been specified. However, it does stay within specified limits if no such risk exists. The default value for this property is five buffers per input, which works well with most of the requirements. You can specify a larger value to improve performance if you are working with large data sets and have

enough free memory on the server. Alternatively, you can decrease the value from the default value if the server is already struggling with memory pressure. But as you can imagine, reducing this value to 0 will disable all throttling and can adversely affect performance. You need to be a bit more cautious when configuring this transformation and configure a more realistic setting, as the memory usage could go too much off the mark when configured too low for a large data set. Another issue has been observed when both the input data streams hit this transformation at wildly different times. Testing your data flow is the best recommended approach for finding the right balance with this transformation.

Last, you can specify to treat null values equal by using the TreatNullsAsEqual property of the transformation; the default is True for this property. If you decide not to treat nulls as equal, the transformation then treats nulls similar to the database engine.

Cache Transform

The Lookup transformation in Integration Services 2008 can use an in-memory cache or a cache file to load reference data to perform lookups on the data in pipeline. The in-memory cache or the cache file needs to be created for the lookup transformation before it starts execution. This means you can create a reference data cache either before the package starts running or during the same package execution, but before the lookup transformation. Using a cache transformation, you can write distinct data rows to a cache connection manager that, depending upon how you have configured it, can write this data into cache or can also persist this cached data to a file on the hard disk. The user interface of the Cache Transformation Editor is simple and has only two pages. In the Connection Manager page, you specify the Cache Connection Manager, and in the Mappings page, you map Input columns to the destination columns being written to a Cache Connection Manager. Here, you have to map all the input columns to the destination columns; otherwise, the component will throw an error. If you change the data type of a column later—e.g., you increase the length of an input column—the metadata can be corrected in the Cache Connection Manager, which displays the columns and their metadata in the Columns tab. While configuring the Cache Connection Manager, you will also need to configure an Index Position for the columns. By default, the index position is 0 for all the columns, indicating that these columns are non-index columns. You can assign positive integer values such as 1, 2, 3… and so on to the columns that are participating in indexing. The number assigned to an index position on a column indicates the order in which the Lookup transformation compares rows in the reference data set to rows in the input data source. The selection of the columns as index columns depends upon which columns you need to look up while configuring a Lookup transformation. For example, if you

are matching for car details based on the manufacturer, the model description, and the body style, you will assign the index positions as shown in this table:

Column Name	Index Position	Index Position Description
Manufacturer	1	This column participates in the lookup operation and should be mapped to the input column in a lookup transformation. This is the first column on which the comparison will be made.
Model Description	2	This column participates in the lookup operation and should be mapped to the input column in a lookup transformation. This is the second column on which the comparison will be made.
Body Style	3	This column participates in the lookup operation and should be mapped to the input column in a lookup transformation. This is the third column on which the comparison will be made.
All other columns	0	These columns do not participate in the lookup operation mappings; however, they can be selected in order to add them into the data flow.

As the Cache Connection Manager writes data to memory, it gets tied up to the cache. Hence, you cannot use multiple cache transforms to write to the same Cache Connection Manager. The cache transform that gets called first during the package execution writes data to the Cache Connection Manager while all subsequent cache transforms fail. As mentioned earlier, you can create a cache file using a cache transform in a separate package or in the same package that runs before the Lookup transformation. So, once the reference data is persisted to a cache file, which is a raw file, you can use this cache file among multiple data flow tasks within the same package, between multiple packages on the same server, or between multiple packages on different servers. The only consideration you have to keep in mind is that the data in the cache file will be current to the time it is loaded into the cache file. If your reference data doesn't change that often, you can use this technique to perform quite fast lookup operations.

Lookup Transformation

With the ever-increasing use of the Web to capture data, lookup operations have become quite important. Web page designers tend to ask users to fill in the most critical data in a web form, and the form fills in rest of the information for them. For example, you may ask a visitor on your site to fill in a street address and postal code during registration, and based on these two pieces of information, you can fill the city and state address fields. This is done by looking up a database table that contains all the postal information keyed in with postal codes. So, you simply look for the row that contains the particular postcode and you will be able to complete the address fields. You can perform such lookup operations in Integration Services also—using

the Lookup transformation. While loading a data warehouse, you will frequently use a Lookup transformation to look up surrogate keys in a dimension table by matching the business key.

This transformation lets you perform lookups by joining the input columns in the data flow with columns in a reference data set. The resulting values can be included in the data flow in the new columns, or you can replace the existing column values. For example, when a value of a record in the postal code column in the data flow is equal to a record in the postal code column in the reference data set, the lookup transformation will be able to get data for all the other address columns. This equality operation makes this join an *equijoin*, requiring that all the values in the data flow match at least one value in the reference data set. In a complex lookup operation, you may be joining multiple columns in the input to the multiple columns in the reference data set.

This transformation has undergone a major upgrade since its predecessor in Integration Services 2005. The user interface is also changed and provides more flexibility and performance improvements. The Lookup Transformation Editor provides five pages to configure its properties. When you open the editor, you will choose the options in the General page to decide how you want to configure this transformation. Let's start with the simplest one—the Lookup transformation allows you to configure its outputs on the basis of how you want to handle the non-matching rows. The "Specify how to handle rows with no matching entries" option in the General page provides you four options to choose from:

- Ignore failure
- Redirect rows to error output
- Fail component (default)
- Redirect rows to no match output

When the transformation looks up an input data key against the reference data set, you can either have a match or no match. When the input row key is found in the reference data set, the row is called a match row and is sent to the match output. If a key in the input row is not found in the reference data set, the row is called a no-match row and this option allows you to decide how you want to handle such rows. In Integration Services 2005, no-match rows are treated as errors and are sent to error output, whereas in Integration Services 2008, you can choose to redirect non-matching rows to a no-match output. Choosing an option other than "Redirect rows to no match output" will treat no-match rows as error rows and will send them to error output.

Next you can select how you would like to cache your reference data set. The Cache Mode section provides three options to choose from:

- **Full cache** Selecting this option will force the reference set to be prepared and loaded into the cache memory before the lookup operation is performed. For example, if you are looking for postal addresses, all the rows with postal information will be loaded into the memory before actual lookup is performed against this reference data set. As you can imagine, lookup against a reference set that is held in memory will be faster compared to the lookup operation where the component has to make a connection to the outside data store to get the lookup values. Full cache mode is the best-performing option if you have enough physical memory on the server to hold the complete reference data set. When you choose this option, two things happen in the user interface:
  - The Advanced page is disabled. The options in this page are not used when using full cache mode.
  - The Cache Connection Manager option becomes available. The Lookup transformation can use either an OLE DB Connection Manager or a cache connection manager in the case of full cache mode. This feature is a new addition in this version. The Cache Connection Manager can read the reference data from a cache file that can be prepared separately from the lookup process. You can actually prepare the reference data set separately, in the same Data Flow task before the Lookup transformation, in an earlier Data Flow task in the same package, or even in a separate package using a Cache transform. When the data is prepared and persisted to a cache file (.caw), the Lookup transformation can load the data from the cache file faster than from the relational sources using a Cache Connection Manager. So, by utilizing full cache mode and sharing a cache file between multiple lookups within the same package or in multiple packages, you can achieve high levels of performance, especially when the reference data set is not small. The only caveat here is that your reference data set must not change. If it changes with each data load, you have to be careful while using the Cache transform and the Cache Connection Manager in your package.
- **Partial cache** Selecting this option will enable you to apply limits on the use of memory for caching reference data. When you select the partial cache mode, the Advanced page options become available, but the Cache Connection Manager option is disabled, as it is used only in the full cache mode. So, you can only use the OLE DB Connection Manager to collect reference data from a relational source. In the Advanced page, you can limit the cache size to a value that you can

specify in the Cache Size fields. Two Cache Size fields have been provided in the Partial caching section—one is for a 32-bit system and the other one applies to a 64-bit system. What happens with partial cache mode is that the partial reference data rows are loaded into the cache based on the cache size value and as the lookup operation applies, the input data rows are matched against the reference data in the cache. If a match is found, the row is directed to the match output, and if a match is not found in the cache, a query is sent to the OLE DB source to find if a match exists. Again, if a match is found the row is sent to the match output; however, if a match is not found in the reference database the non-matching row is directed to the error output or no-match output, depending on how you've chosen to handle the no-match rows. This trip to an external reference database to find a match or no match could be quite expensive and can hurt the performance of a lookup operation. If your reference data is not changing while the package is running and you've sufficient free memory available on the server, you can choose to store the no-matching rows in the cache using the option "Enable cache for rows with no matching entries." You can also apply a limit on the amount of cache no matching rows can use by specifying a percentage value in the Allocation From Cache field. The next time the no-matching row with the same key appears in the input data set, the lookup transformation already knows about this, as it has it in the cache and doesn't do a trip out to the database. This process goes on till the cache is completely filled up, at which time the Lookup transformation removes the least frequently used rows from the cache.

You can also improve the query performance by selecting the Modify The SQL Statement check box. Using this option, you can use an SQL statement to select a subset of the reference data set. For example, if you have selected a table or a view as the reference data set in the Connection page that contains address details for the whole country and your input records contain records for a state only, you don't need to load all the reference data set in the memory; rather, you need to load only the particular state's address details. To limit the reference data set dynamically, you can also use values or parameters in the WHERE clause of the SQL statement and use the Parameters button to map parameters to input columns. When you use parameters in the WHERE clause mapped to input columns, these parameters are first evaluated from the input data set at run time and limit the data set that needs to be referenced and hence increases overall performance.

- **No cache** This option does not load reference data set into the cache; rather, it makes a connection to the external data source using an OLE DB Connection Manager to perform the lookup operation. This option works like partial cache mode with a difference that all the input rows are matched against data in the database table, other than the last retrieved row, which always resides in the cache.

When you choose this option, the Partial Caching section in the Advanced option is disabled as there is no cache to configure. However, the Custom query can be used to modify the SQL statement that is applied in the Connection page.

You can specify the connection settings for the reference data set in the Connection page. The graphical interface in this page changes to match the options you have selected in the General page. When you select full cache mode, you can choose a Connection type of either a Cache Connection Manager or an OLE DB Connection Manager. If you choose the Cache Connection Manager option, the Connection page has only one field in which you can specify the Cache Connection Manager. However, if you choose an OLE DB Connection Manager in the case of full cache mode or are using a partial cache mode or no cache mode, the interface of the Connection page changes to allow you to specify settings for an OLE DB Connection Manager. In this case, the reference data set can be a table or view in a database or can be a result of an SQL statement. So, you specify an OLE DB Connection Manager, select a table or view or an SQL query if you prefer to use an SQL query in the Connection page. In this page, you can either select an existing table or create a new table by clicking the New button provided for the "Use a table or a view" option and can populate this table at run time before this Lookup transformation executes. If you select "Use results of an SQL query," you can either type your SQL statement directly in the Option field or you can build the query with the Query Builder, which you can invoke by clicking Build Query. You can also read the query from a file by clicking Browse.

You specify the join columns in the Columns tab and can also select reference columns that you want to include in the data flow. To make a join, drag a column from Available Input Columns and drop it on the column you want to join with in the Available Lookup Columns list. You won't be able to map or join two columns that don't have matching data types. To select a column to include in the output, check the box provided for the Available Lookup Columns. As you select a column, it will be listed in the lower half of the editor window, where you can select the lookup operation and assign an output alias as well to the selected column. You can add the selected column as a new column or replace the values of the same column in the input columns.

Finally, note a few cautions when you add this transformation in the data flow. The Lookup transformation's lookup operation is case-sensitive—i.e., the string comparison in the lookup is case sensitive—while this may not be the case with your database if you're using a case-insensitive collation. So you need to be careful with the input data set case, *vis-à-vis* the case of the reference data set, and you also need to understand where the comparison is made. If you're using full cache mode, the comparison is made within the lookup transformation and is case sensitive; however, if you are using no-cache mode, then the lookup operation is made in the database engine and, based

on the database collation, could be case-insensitive. Similarly, you need to be clear how you are configuring your cache in the partial cache mode and where the comparison will happen. One solution may be to convert both the string data sets to either lowercase or uppercase for this transformation. Another reason for lookup failures could be the null values in data. If the lookup operation matches columns that contain null values, it will fail when you are using no or partial caching and the comparison happens in the database engine instead of in the lookup transformation. A possible solution for this could be either to use full caching mode or to remove null values by modifying the SQL statement. Last, you can join (map) columns in the lookup transformation with any data type other than DT_R4, DT_R8, DT_TEXT, DT_NTEXT, or DT_IMAGE because they are not supported data types for joins.

Hands-On: Updating PersonContact Data

You receive contact details for persons interested in your products from different sources. As a policy, you standardize data as you insert it into a PersonContact table by performing lookups for address details on the basis of postal code, which is a mandatory requirement for the data to be accepted. This data mart will be used for marketing purposes, and hence you want to make sure that all the contacts have a Salutation and there are no duplicated contacts in the table.

Method

In this exercise, you receive data in two different file formats, the Microsoft Excel file format and the flat file format, and you will load this data into a PersonContact table in the Campaign database after applying the required transformations. In the data flow of this package, you'll combine the data from these two files, format it to the required data types, generate a salutation in the data flow, perform a lookup on the postal code to get the correct city, and delete duplicate records before loading it into the table. Here is the step-by-step process:

- Add data flow sources to get the data in the data flow.
- Convert the data to match the data types outputted by both the data flow sources and then combine the data from both data streams.
- Derive a salutation.
- Enhance address details.

- Delete duplicates and insert data into the PersonContact table.
- Execute the package to see the results.

Exercise (Add Data Flow Sources)

You will create a new Integration Services project that will subsequently be used for all the packages developed in this chapter. You will add a data flow in the package and then will add data flow sources in the data flow.

1. Create a new Integration Services project with the following details:

Name	Data Flow transformations
Location	C:\SSIS\Projects

2. When the blank project is created, rename the package as **Updating PersonContact.dtsx**.
3. Drop the Data Flow task from the Toolbox onto the Control Flow Designer surface and then double-click it to go to the Data Flow tab.
4. In the Data Flow tab, drag and drop a Flat File source and an Excel source from the Toolbox onto the Designer surface.
5. Double-click the Flat File source to open the Flat File Source Editor. Click the New button opposite the Flat File Connection Manager field to create a connection manager.
6. In the Flat File Connection Manager Editor, type **PersonDetails01** in the Connection Manager Name field. Specify C:\SSIS\RawFiles\PersonDetails01.txt in the File Name field. Select the check box for the "Column names in the first data row" option. The rest of the options will be automatically filled for you with default, as shown in Figure 10-1.
7. Go to the Columns page and the columns will be listed for you as the Flat File Connection Manager selects {CR}{LF} as the row delimiter and comma {,} as column delimiter values automatically.
8. Go to the Advanced page where you can specify the column data type details. In this exercise, our final destination is the PersonContact table, so you can check the schema of this table and update column data types accordingly. The following table shows these details, which you need to apply to the column properties in this page. For example, to specify settings for the Postcode column, click this column and then in the right side pane, go to the OutputColumnWidth field and change it to 12. Make sure DataType is set to String [DT_STR]. Similarly apply

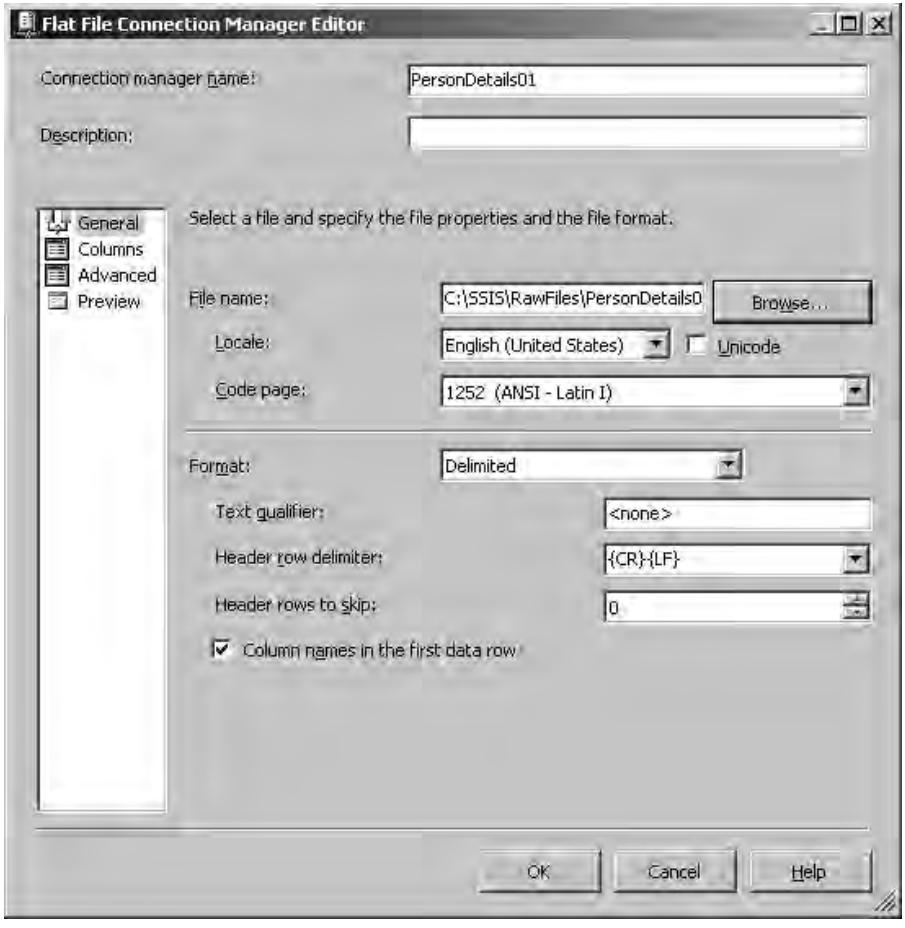


Figure 10-1 *Flat File Connection Manager settings*

the following settings to all the columns. After having applied all the settings, click OK to close the page and return to the Flat File Source Editor.

Name	OutputColumnWidth	DataType
FirstName	50	string [DT_STR]
LastName	50	string [DT_STR]
Gender	1	string [DT_STR]
Married	1	string [DT_STR]
AddressLine1	255	string [DT_STR]
AddressLine2	255	string [DT_STR]
Postcode	12	string [DT_STR]

9. Go to the Columns page and check out that all the Available External Columns have been selected. Click OK to close the editor. Right-click the Flat File source and rename it **PersonDetails01**.
10. Right-click the Excel source and rename it **PersonDetails02**. Double-click PersonDetails02 to open the Excel Source Editor. Click the New button provided opposite to the OLE DB Connection Manager field.
11. In the Excel Connection Manager dialog box, specify C:\SSIS\RawFiles\PersonDetails02.xls in the Excel File Path field. Leave Microsoft Excel 97-2003 specified in the Excel Version field, and leave the check box selected for "First row has column names." Click OK to create this connection manager.
12. In the Excel Source Editor, leave "Table or view selected" in the Data Access Mode field. Select PersonDetails02$ in the "Name of the Excel sheet" field from the drop-down list and go to the Columns page to see that all the Available External Columns have been selected. Click OK to close this editor.
13. Rename the Excel Connection Manager to **PersonDetails02**.

Exercise (Combine Two Data Streams)

In this part of the exercise, you will add a Data Conversion transformation to convert the data type of columns coming from the Excel source to match with the data types of columns coming through Flat File source and then use Union All transformation to combine these two data streams.

14. Right-click the Excel source and choose Show Advanced Editor from the context menu. Go to the Input and Output Properties tab, expand the Excel Source Output and check out the properties of columns listed in External Columns and Output Columns. Note that these columns are of the Unicode string [DT_ WSTR] data type with column width (Length) equals 255. These columns need to be converted to the data type so that you can combine them with the columns from the Flat File source. Close the Advanced Editor.
15. Drop a Data Conversion transformation from the Toolbox onto the Data Flow surface just below the Excel source. Join the Excel source with this transformation using the green arrow.
16. Double-click the Data Conversion transformation to open its editor. As you want to convert the data type of all the columns, select the check boxes provided in front of all the columns in the Available Input Columns. As you select the columns, a row for each column will be added in the lower grid section, where you can configure the changes.
17. For the FirstName Input Column, note its Output Alias, which means the converted column will be added to the transformation output as Copy of FirstName. Click in the Data Type field and change it to string [DT_STR] from the drop-down list

and change the Length to 50. A Code Page will be selected by default based on the code page of the computer; however, you can change it if you need to by selecting a different code page from the drop-down list. Similarly, change settings for all the columns as shown in the Figure 10-2.

18. Click OK to close the transformation. Rename the Data Conversion transformation **Converting PersonDetails02**.
19. Drop the Union All transformation from the Toolbox on the Data Flow surface between PersonDetails01 and Converting PersonDetails02. Join both of these components to the Union All transformations using green arrows.

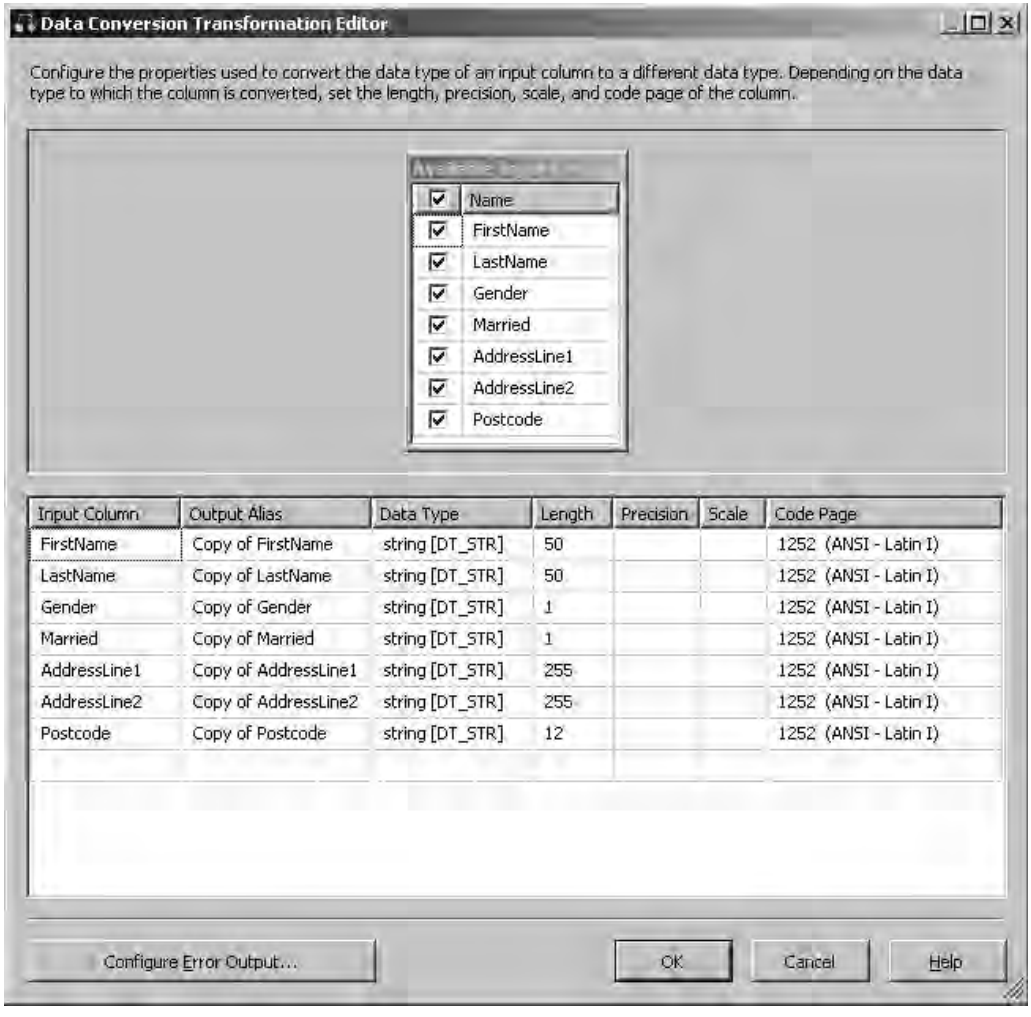


Figure 10-2 *Converting Excel source data using Data Conversion transformation*

20. Double-click the Union All transformation to open the editor. Here, you will see Output Column Name columns automatically mapped to Union All Input 1 columns, whereas you will need to map Union All Input 2 columns yourself. Click in the FirstName column under Union All Input 2 and select the converted Copy of FirstName column. If you select FirstName in this column, you will get an exception error for incompatible data type. Map all the output columns to the converted columns as you did for FirstName column (Figure 10-3).
21. Click OK to close the editor. Rename the Union All transformation **Merging PersonDetails01 and PersonDetails02**.

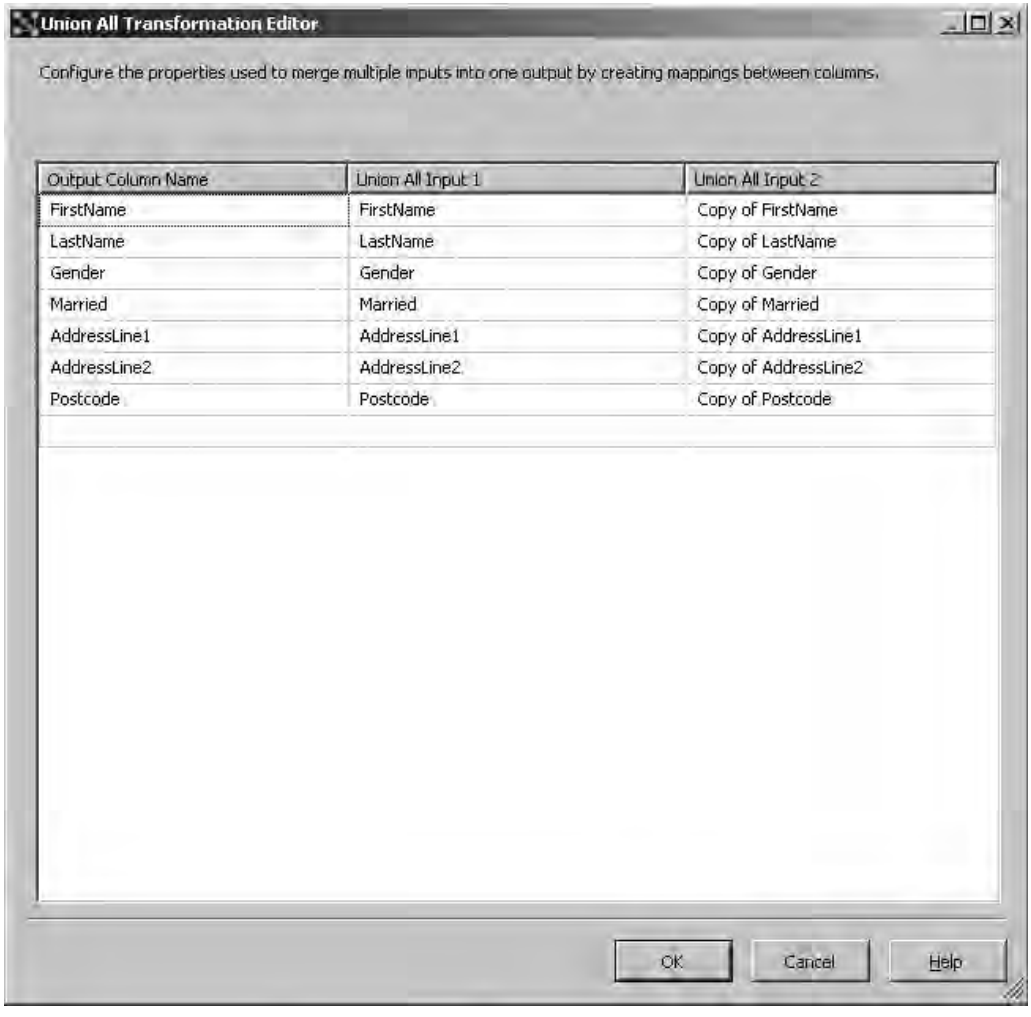


Figure 10-3 *Combining two data streams using the Union All transformation*

Exercise (Derive Salutation)

You will use a Derived Column transformation in this part to derive a salutation. You need to derive a salutation as Mr., Mrs., or Miss.

22. Drop a Derived Column transformation from the Toolbox below the Merging PersonDetails01 and PersonDetails02 and join both of them by dragging the green arrow onto the Derived Column transformation.
23. Double-click the Derived Column transformation to open the editor. This transformation has an Expression Builder interface. Click in the Derived Column Name field and type **Salutation**. By default, <add as new column> will be displayed in the Derived Column field, which you can change to replace an existing column from the drop-down list.
24. Type the following expression in the Expression field (refer to Figure 10-4):

```
Gender == "m" ? "Mr." : (Gender == "f" && Married == "y" ? "Mrs." : "Miss")
```

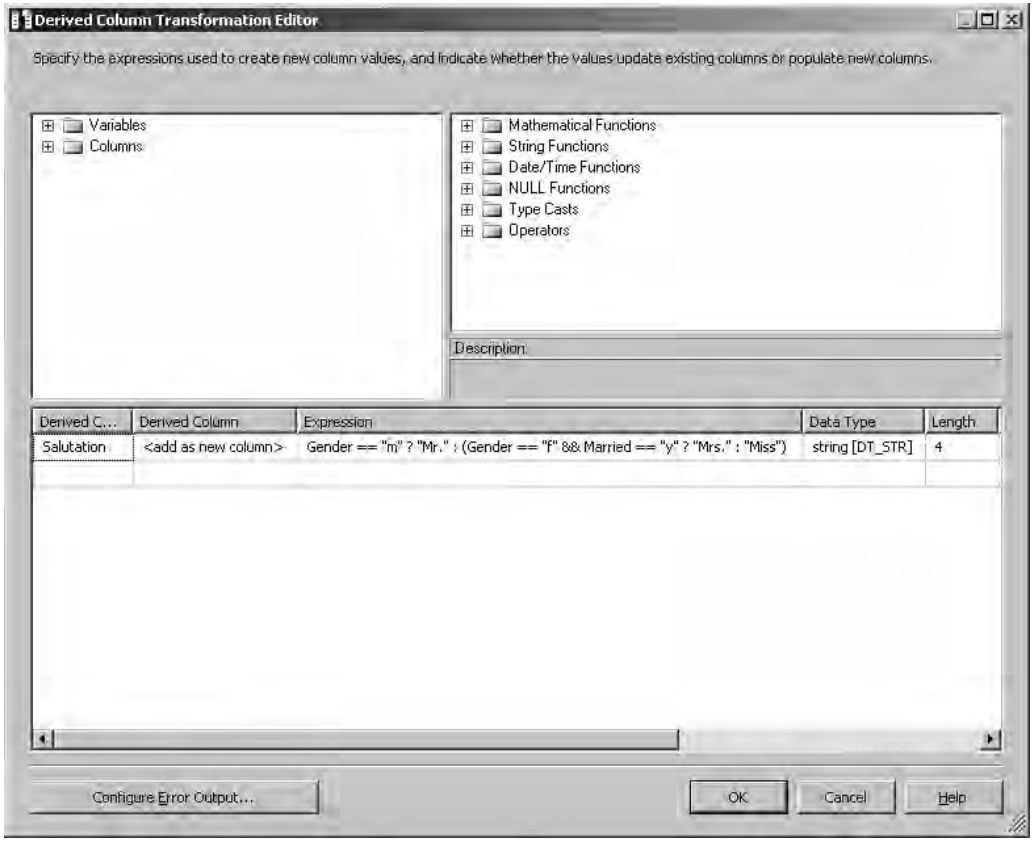


Figure 10-4 *Deriving a salutation using Derived Column transformation*

In this expression, you are using a conditional operator that returns one of the two expressions based on the evaluation of the Boolean expression.

25. Select string [DT_STR] from the drop-down list in the Data Type field and specify 4 in the Length column if not automatically specified. If you can't change the data type here, you can in the Input and Output Properties tab of the Advanced Editor. Click OK to close the editor. Rename this transformation **Deriving Salutation**.

Exercise (Enrich Address Details)

One of the requirements you will address in this exercise is to standardize the address detail by including City on the basis of the Postcode available in the data. You've a table called PostalCity in the Campaign database that contains a list of cities against postal codes, and you will perform a lookup operation to get the correct city against a postal code. As a lookup operation is a case-sensitive operation in SSIS, you will convert the Postcode column in both places—i.e., in the pipeline as well as in the reference data to UPPERCASE. Let's first use a Character Map transformation to convert the pipeline data.

26. Drop a Character Map transformation from the Toolbox on the Data Flow Designer. Join the Character Map transformation with Deriving Salutation by dragging and dropping the green arrow from the latter.
27. Double-click the Character Map transformation to open the editor. Click in the check box next to the Postcode column to select it. A row will be added in the lower grid area with Input Column selected as Postcode.
28. Check that the Destination column in the lower grid has only two possible values: New Column and In-place Change. Select the In-Place Change value in the Destination column. Note that the Output Alias is also changed to Postcode (see Figure 10-5).
29. Invoke the drop-down list of available operations in the Operation field. Select Uppercase by clicking the check box in front of it. Click OK to complete your selection.
30. Click OK to close the editor. Rename Character Map transformation **Uppercasing Postcode**.
31. Now drop a Lookup transformation on the Data Flow surface after the Uppercasing Postcode transformation. Join both of these transformations using a green data flow path.
32. Double-click the Lookup transformation to open the editor. Select the Partial Cache mode in the General page.
33. In our data stream, some of the postal codes will not find matches for cities, so they will fail the Lookup transformation. You will configure this transformation to divert these mismatching records to a flat file so that you can review them later on. Select the "Redirect rows to no match output" option under the "Specify how to handle rows with no matching entries" field.

Figure 10-5 *Configurations of Character Map transformation*

34. Go to Connection page, click the New button next to the OLE DB Connection Manager field, and choose localhost.Campaign from the Data Connections list; then click OK to add this connection manager.
35. Select the Use results of an SQL query option and type in the following SQL query.

```
SELECT [City], UPPER([Postcode]) Postcode FROM [Campaign].[dbo].[PostalCity]
```

36. Go to the Columns page and map the Postcode column in the Available Input Columns list to the Postcode in the Available Lookup columns.

37. Click in the City Column check box to add this column as a new column in the output, as shown in Figure 10-6.
38. Click OK to close the Lookup transformation. Rename this transformation as **Adding City Column**.
39. From the Data Flow destinations section of the Toolbox, drag and drop the Flat File destination below the Adding City Column transformation. Drag the green arrow from the Adding City Column and drop it on the Flat File destination. This will open the Input Output Selection dialog box. Select Lookup No Match Output in the Output field and click OK to close.

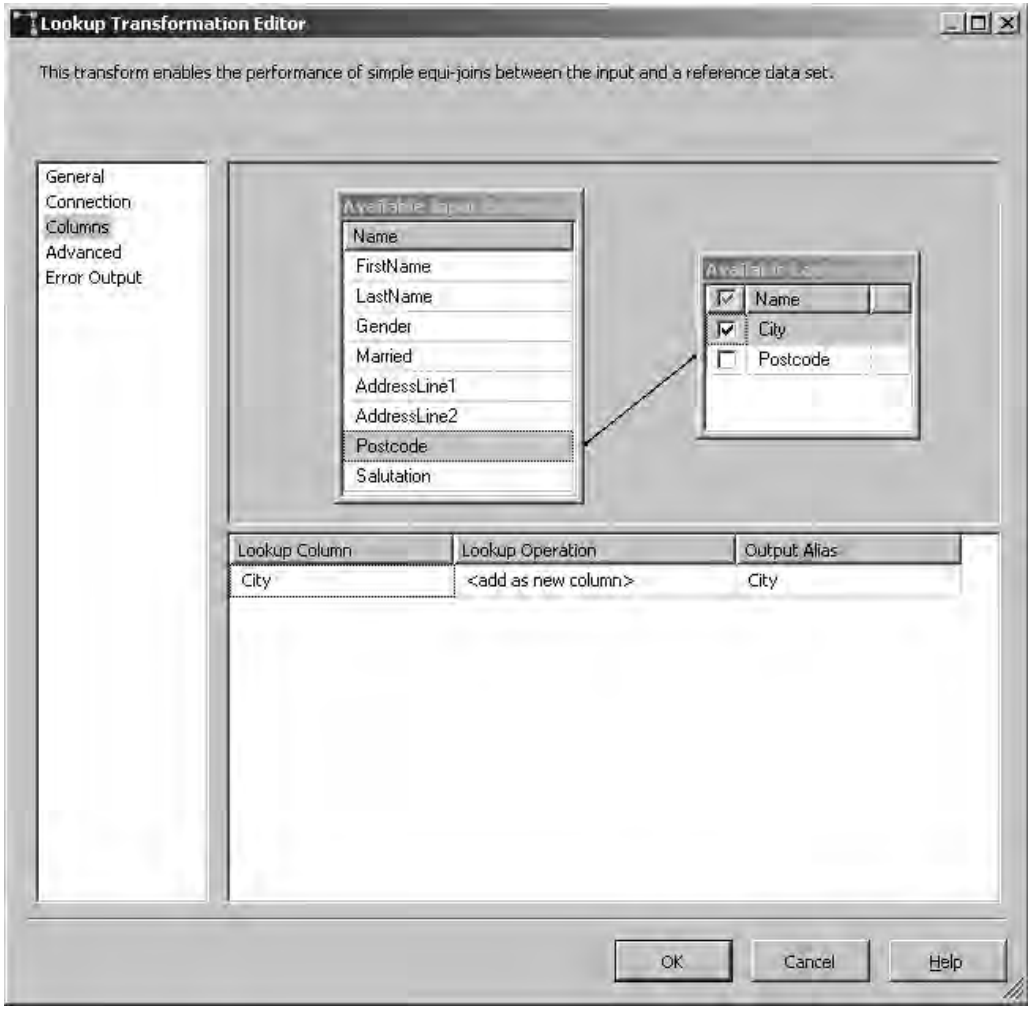


Figure 10-6 *Adding a new column in the output using the Lookup transformation*

40. Double-click the Flat File destination to open its editor and click the New button next to Flat File Connection Manager. Select the Delimited Option radio button in the pop-up dialog box asking you to select a Flat File Format, and then click OK. This will open the Flat File Connection Manager Editor.
41. Type **No Match Lookups** in the Connection Manager Name field and **C:\SSIS\RawFiles\NoMatchLookups.txt** in the File Name field. Click to check the Column names in the first data row option.
42. Go to the Columns page and check out that {CR}{LF} is selected as Row delimiter and Comma {,} is selected as Column delimiter. Go to the Advanced page and check out the properties of the columns. Note that the connection manager has correctly picked up the data types and the length of the columns. Click OK to close it and return to the Flat File Destination Editor. Leave the Overwrite Data in the file option selected.
43. Go to the Mappings page to create mappings between the Available Input Columns and Available Destination Columns and then click OK to close it. Rename Flat File destination as **No Match Lookups File**.

Exercise (Delete Duplicates and Load PersonContact)

In this part of the exercise, you will first delete the records from the PersonContact table that are to be updated and are also coming in the pipeline and will then insert all the records coming in the data flow into the PersonContact table.

44. Drop the OLE DB command from the Toolbox below the Adding City Column transformation. Connect both of these transformations using the green data flow path.
45. Double-click the OLE DB command to open the Advanced Editor for OLE DB Command. Choose localhost.Campaign in the Connection Manager field from the drop-down list in the Connection Managers tab. Go to the Component Properties tab.
46. Type **Deleting Duplicates** in the Name field. Type the following SQL statement in the SqlCommand field and click OK.

```
DELETE PersonContact WHERE FirstName = ? AND LastName = ?
```

47. Go to Column Mappings page, where you will see Param_0 and Param_1 columns in the Available Destination Columns. These represent parameters that you've used in the preceding SQL statement. Map Param_0 to FirstName and Param_1 to LastName columns in the Available Input columns (see Figure 10-7) and click Refresh. This should remove any validation error appearing at the bottom of the Advanced Editor. Click OK to close.
48. Drop an OLE DB destination on the Designer surface and join it with the green data flow path from the Deleting Duplicates transformation.

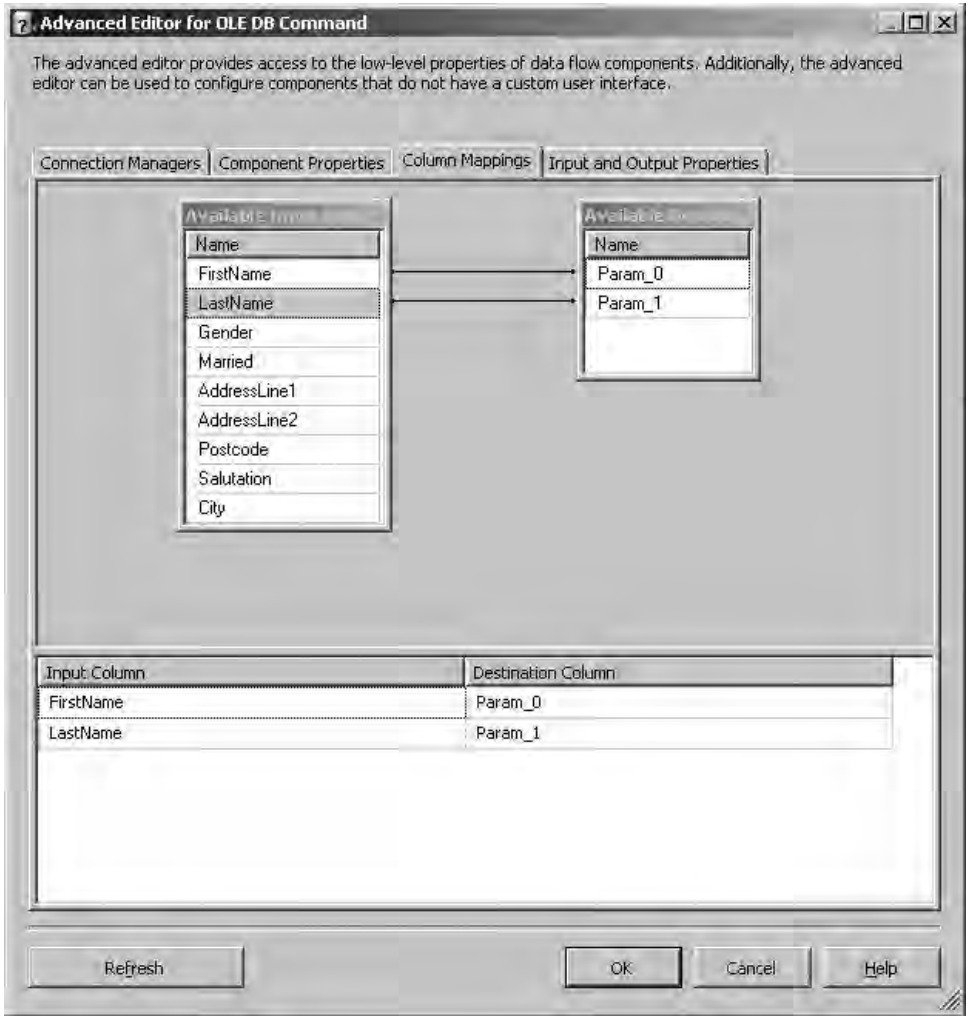


Figure 10-7 *Mapping parameters in the OLE DB Command transformation*

49. Double-click the OLE DB destination to open the editor. You will see localhost .Campaign selected in the OLE DB Connection Manager field, as this is the only OLE DB Connection Manager configured in the package. Leave "Table or view—fast load" selected in the Data access mode field. Select the [dbo].[PersonContact] table from the drop-down list in the "Name of the table or the view" field.
50. Go to the Mappings page to create the required mappings. Click OK to close this. Rename this destination **PersonContact**. After some adjustments, your package should look like one shown in Figure 10-8. Press CTRL-SHIFT-S to save all the items in the project.

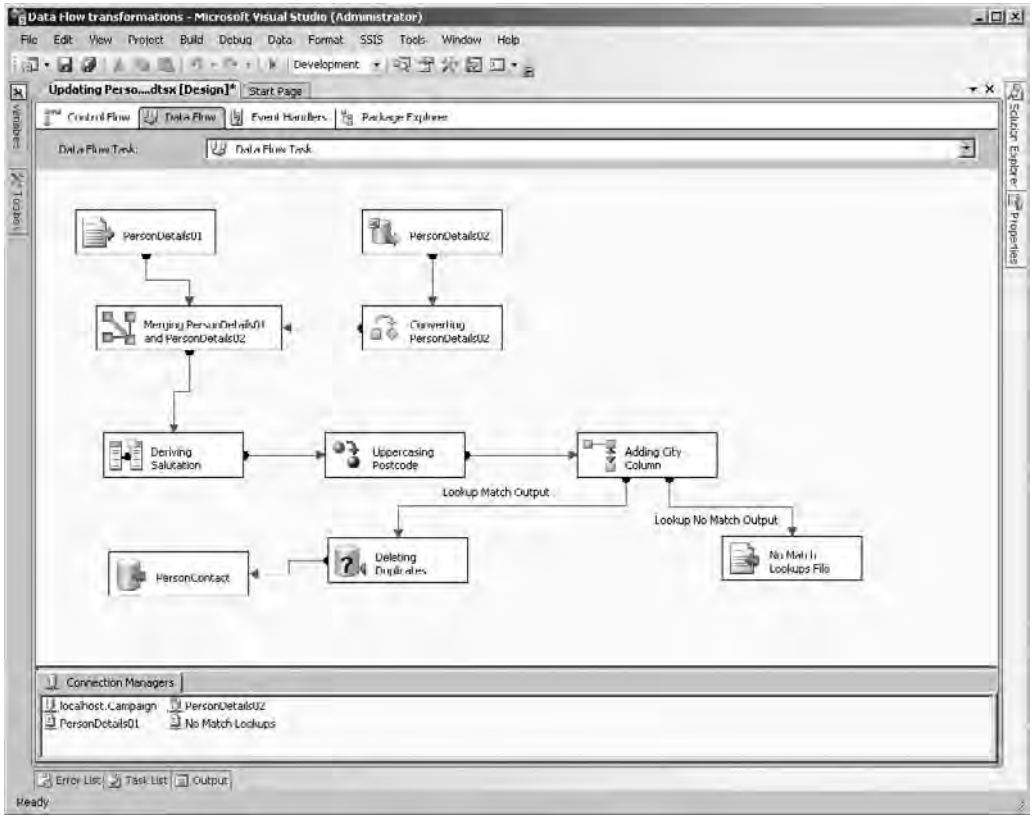


Figure 10-8 *Updating the PersonContact package*

Exercise (Execute the Package)

Here, you will add a data viewer on the Data Flow path to see the data flowing through the pipeline at run time and execute the package.

51. You can add as many data viewers to the package as you like before executing the package, but as a minimum, add a data viewer before the Deriving Salutation component and before the Deleting Duplicates component. To add a data viewer, double-click the green line joining the two components to open the Data Flow Path Editor. Go to the Data Viewers page, click Add, and then click OK in the Configure Data Viewer dialog box to return; then click OK again to add a grid type data viewer.
52. Press F5 to execute this package. As the package is executed, you will see two data viewer output windows appear. If you adjust them on the screen (without

dropping them on each other) so that you can see the background package as well, you can notice that the package execution is halted till the place where the data viewer is added in the data flow.

53. In the beginning of the package execution, note that 20 rows from flat file and 30 rows from Excel file are combined by the Union All transformation. In the first data viewer, you can see that 50 records (see in the status bar at the bottom of the data viewer window) have been combined and are flowing as a single data collection. Detach this data viewer and see the data in the second data viewer, where the Salutation has been derived and the City column has been populated with city names. This data viewer shows only 46 records, as 4 records did not find an exact match in the lookup table and so were sent out to the No Match Lookups output—i.e., No Match Lookups File. Click Detach to complete the package execution.
54. Press SHIFT-F5 to stop debugging. Close the project and exit BIDS.

Review

In this first data flow Hands-On exercise, you used several components that are the basic building blocks in a data flow. You also saw how you can use these components one after another to convert data to match the succeeding components requirements, while keeping in line with the final destination's requirements. You also captured records from the Lookup transformation for which there were no exact matches in the Lookup table. The concept of a Lookup table is extensively used to standardize data and to identify updates and inserts in the loading process. There are examples in Books Online that you can refer to understand more about configurations of lookup transformation in full cache mode or partial or no cache mode. Later in the exercise you have used an OLE DB Command transformation that uses parameters to map to input column values. If you are updating a table with a large number of columns, it will become confusing and troublesome to configure OLE DB transformation as the parameter names are not intuitive. Fortunately, this problem can be solved by using a stored procedure than using raw SQL in the SqlCommand property. For example, in the preceding case you could also create a stored procedure as follows:

```
CREATE PROCEDURE uspDeletingDuplicates
      @FirstName VARCHAR(50),
      @LastName VARCHAR(50)
AS
BEGIN
      SET NOCOUNT ON;
      DELETE PersonContact
      WHERE FirstName = @FirstName AND LastName = @LastName
END
```

And you could execute the stored procedure using SqlCommand field as

```
EXEC dbo.uspDeletingDuplicates ?, ?
```

And in Column Mappings tab, you could have mapped columns using the variable names as shown in Figure 10-9. As you can make out, it is much easier to understand and maintain.

Figure 10-9 *Updating the PersonContact package*

Rowset Transformations

The Rowset transformations work on record sets. These transformations first receive all the rows and then do the data processing as the operations they perform need all the rows upfront. Among these are the Aggregate transformation, Sort transformation, Percentage and Row Sampling transformations, and Pivot and Unpivot transformations. In this section, you will work through two Hands-On exercises built around these transformations—one built around the Aggregate transformation and the other one built around the Pivot transformation, in which you will also use the Sort transformation along with other data flow components.

Sort Transformation

The Sort transformation allows you to sort input rows in ascending or descending order by selecting one or more input columns for sort order criteria, similar to the ORDER BY clause of T-SQL. To sort the records, as you can imagine, this transformation requires collecting all the rows before applying sorting order. While sifting through the records, this transformation can also look for duplicate records with the same sort key values and can "de-dupe" them. The Sort transformation supports one input and one output to perform its operation.

The user interface of this transformation is simple and has a list of Available Input Columns in the upper half of the dialog box, along with check boxes before and after the column names. The selection of the check boxes after the column names—i.e., the Pass Through check boxes—allow the columns to be included in the sorted output; the selection of check boxes before the column names allow you to sort the input records on those columns. The lower half of the dialog has five fields to specify the columns you want to work with and the criteria to apply for sort order.

When you open the Sort Transformation Editor, you will realize that all the Pass Through check boxes have already been selected. This means by default all the columns will be sent to the Sort Transformation output columns. To configure the sort order, you select the input columns by clicking the check boxes before the Available Input Columns. Alternatively, you can select a column from the drop-down list invoked by clicking in the Input Column field. As you select an input column, notice that Output Alias, Sort Type, and Sort Order are automatically assigned to this column, though you can modify these values. The default value for the Sort Type is ascending, but it can be changed to descending. The Sort Type setting applies a sorting order to the data in the selected column and has nothing to do with the Sort Order field, which applies a sort order to the columns. The Sort Order is a numerical value assigned to a column on the basis of its position in the list of columns selected for sorting. This value starts with 1 for the first column and is increased by 1 for subsequent columns selected. The Sort Order value

determines which column is sorted first. The column with the smaller value is always sorted before a column with bigger Sort Order value—i.e., the column with a Sort Order value of 1 will be sorted before the column with a Sort Order value of 2, and so on.

The Sort Type and Sort Order are represented by a single property of the input column in the Advanced Editor. If you open the Advanced Editor for Sort Transformation, go to the Input and Output Properties tab, expand the Sort Input and then expand Input columns, you will see a list of input columns that are available to the transformation and have been selected to pass through. If you click any of the input columns and scroll to the bottom of its properties, you will see the property NewSortKeyPosition, which holds a positive or negative value. The positive value indicates that the sort type is ascending, and a negative value indicates that the sort type is descending. The numerical value—i.e., 1, 2, or 3 and so on—represents the sort order property of customer user interface. A value of 0 indicates that the column is not included in the sort criteria; in fact, these are the columns that have only Pass Through check boxes selected against them. The input columns that are not included in the sort criteria, but are selected to pass through, are copied along with the sorted columns to the output columns.

Once the Sort transformation collects all the rows, it can apply the sorting criteria to the data rows. A sort operation can be an expensive process and can cause performance issues. You need to decide what you want to achieve before using this transformation; for example, do you want to provide all the server power to the sort process or want to limit usage of resources for sort operation? In the Advanced Editor for Sort Transformation, you can specify the maximum number of threads that can be used by the Sort transformation in the MaximumThreads property in the Component Properties tab. The default value for this property is –1, which indicates an infinite number of threads available to the transformation.

Returning to the Sort Transformation Editor, where you've selected input columns and specified sort criteria, notice the Comparison Flags field. If you click in this field, you will see a list of options. The purpose of this field is to specify how the sorting should handle the data comparison. As you sort columns, the Sort transformation compares the data to allocate a proper sort order for the data rows. You can specify the options in this field that can affect the sensitivity of the comparison of the data. For example, you can choose the Ignore Case option to specify that uppercase and lowercase data is to be treated equally. Six options are available, and you can choose more than one option here:

- **Ignore Case** You choose this option to specify that while comparing data, the Sort transformation will not distinguish between uppercase and lowercase letters.
- **Ignore Kana Type** If you are using the Japanese language, selecting this option requires the Sort transformation to ignore the distinction between hiragana and katakana, the two types of Japanese kana characters. For the benefit of those who

are less acquainted with Japanese, kana is a general term used to express the two types of Japanese syllabic scripts. Hiragana is a cursive and flowing variety of kana used in most modern Japanese texts, and katakana is a relatively angular kana used for writing foreign words or official documents such as telegrams.

- **Ignore Nonspacing Characters** The data you deal with may contain diacritics, especially in the age of Internet when the data entry is left in the hands of end users across the globe. A diacritic is a mark added to a letter to indicate a special phonetic value—such as the acute accent of *resumé*. Selecting the check box for this option treats the spacing characters and diacritics alike.
- **Ignore Character Width** Using this option allows you to treat the single-byte characters (non-Unicode) and double-byte characters (Unicode characters) alike. Integration Services may automatically convert the data to Unicode data before comparing text data.
- **Ignore Symbols** Sometimes it is useful to compare the string data by eliminating the symbols and white-space characters. This may be due to poor data quality or standardization, again due to free style data entered by users. The Sort transformation ignores the symbols when you select this option and treats *HNO#* and *HNO* as identical.
- **Sort Punctuation As Symbols** You can configure the Sort transformation to remove all punctuation symbols except hyphens and apostrophes appended before the string data before comparison. Using this option treats *.NET* and *NET* as identical.

The sort criteria you specify for the input column generates a sort key value used to compare string data and sort it appropriately. If duplicate rows are included and you want to remove these, you can do so by selecting the "Remove rows with duplicate sort values" option provided on the lower-left side of the Sort Transformation Editor.

If you can extract data from the source in a sorted way, then you can avoid using Sort transform. However, some components in data flow need to know that the data is sorted or not and on what fields; for example, the Merge Join transformation will need this information before joining data from two different sources. You can do so by using the IsSorted and SortKeyPosition advanced properties on source adapters. Refer to Chapter 15, particularly Figure 15-4 and Figure 15-5 and the explanation around them, to understand how to configure them.

Percentage Sampling Transformation

When you need to give out data to call centers for telesales activities, you are generally asked to create a sample set from a data segmentation. Sometimes the requirement is defined as a percentage—for example, you may be asked to create a sample set of

a randomly selected 15 percent of the total records in the segment. One other example could be that you need to create a training data set and a test data set for your data mining models and you want to divide your data set based on a defined percentage. In such a case, you can use the Percentage Sampling transformation to create a sample data set from the input rows using the specified percentage. This transformation helps you create a representative data set much smaller in size that you can use for variety of purposes, such as testing your packages in a development environment or using the sample data set for surveys and marketing purposes.

The Percentage Sampling transformation has a simple and intuitive user interface with just four fields. The first field, Percentage Of Rows, allows you to specify a percentage of sampling. The Percentage Sampling transformation uses an algorithm to select at random the number of rows according to the specified percentage. However, the number of rows that this transformation selects does not precisely match with the percentage calculations—i.e., the output rows may be a little bit too many or a little bit too few. The next two fields allow you to specify the names for selected sample output rows and the remaining unselected output rows. As this transformation selects rows for sampling, it outputs those rows onto its first output and the remaining unselected rows are outputted on to its second output. This transformation supports one input and two outputs to support both the selected and unselected data sets and supports no error output.

The last field is a check box and a value pair, you can select the check box if you want to specify a sampling seed and type a value in the field provided as the sampling seed. If you specify a sampling seed and reuse the same sampling seed in a later run, it will produce the same sample output no matter how many times you run the package with the same data set. This is helpful in testing of packages. Alternatively, if you don't specify a sampling seed, this component will generate a random number using the tick count of the operating system. Hence, each time you run a package, a different random number is generated and a different data set is sampled.

Row Sampling Transformation

The Row Sampling transformation works quite similar to the Percentage Sampling transformation to sample a data set. However, the Row Sampling transformation outputs an exact number of rows as specified in the transformation. This random selection of a precise number of rows is sometimes very useful. An example of such a scenario can be a gift allocation to the random selection of people. Suppose you're running a campaign to introduce your new product to different segments of your customers and prospects by sending them an e-mail every week. To promote readers' interest, you decide to award gifts to a random selection of 50 persons who show interest in your product by evaluating it every week. You can easily build this package by bringing into the data flow the records for the persons who evaluated the product in

the current week and then apply a Row Sampling transformation to select 50 persons out of these records.

In the Row Sampling Transformation Editor, you specify the number of rows you want to output. This transformation supports two outputs—one for extracting the selected records and the other for unselected records. You can type in the names for both the outputs in the user interface. It is not necessary for you to configure a downstream data flow to capture the unselected records. You can simply ignore this output, and the records appearing on this output will not be included in the data flow.

You can choose to specify a random seed for selection of records by clicking in the "Use the following random seed" check box. This transformation selects random records on the basis of an algorithm that uses the random seed. If you specify the same random seed, the algorithm will select the same random records for the same input data. When you check this option, a message will pop up to tell you that using the same random seed on the same input data always generates the same sample, and specifying a random seed is recommended only during the development and testing of a package. Specifying a random seed affects the selection of records, and when you don't specify a random seed, the transformation uses the tick count of the operating system to create the random number that is obviously different each time you run the package, and hence the selected random records will be different even for the same input data.

Pivot Transformation

Relational databases are modeled to store normalized data. This normalization of data changes the data view in a way that sometimes may not be as intuitive as businesses desire. The process that is used to convert data from a normalized form to a denormalized form is called *pivoting*. To understand how you can use this transformation, you will be working through a Hands-On exercise later; but for now, let's discuss what a normalized data is by looking at the data we are going to use.

The data you will use in an exercise later is in an Excel spreadsheet that keeps sales order details in three columns: SalesOrderID, ProductName, and OrderQuantity. To keep the data in a normalized form, the table contains multiple entries or rows for the same SalesOrderID. For example, if three products have been purchased under a single SalesOrderID, the normalized data is represented by listing three rows for the same SalesOrderID with a different ProductName in each row to show the purchase quantity for that product. But the sales manager may prefer to see the sales order details with the products and the quantity for each of them listed against the SalesOrderID on the same row. This is when you need to use pivot function to denormalize the data.

Before SSIS made it available, the pivot function was available in Microsoft Excel or third-party tools, or you had to write custom code to accomplish the task. Integration Services now provides both a Pivot transformation and an UnPivot transformation to

provide different data views or forms. The Pivot transformation converts a normalized data set into a less normalized form by pivoting the input data on a column value. The role a column performs in pivoting is defined by the PivotUsage property specifying values ranging from 0 to 3. The column that is used to pivot the data around forms the *set key* for pivoting, and this column is assigned a value of 1 for the PivotUsage property. Assigning a value of 1 to the PivotUsage property of an input column indicates that it is part of the set key of a single-row or multirow set. In our example, the SalesOrderID column will act as a set key for pivoting and the multiple rows with the same SalesOrderID will be combined into one row.

When the data is pivoted, the values in a column, called *pivot key values,* are pivoted to the columns in the output. For specifying the values to be pivoted to output columns, you assign a value of 2 to the PivotUsage property on the input column. In our example, the ProductName column has 10 different values for the products. When this data is pivoted, 10 columns are created on the basis of 10 different values in the ProductName column—e.g., the value Mountain–100 of the ProductName column becomes the Mountain–100 column in the pivoted output. These newly created columns in the pivoted output get the values from the third column, which is used to provide values for newly created columns in the pivoted output by specifying a value of 3 to the PivotUsage property. All other input columns that don't participate in the pivoting process are assigned PivotUsage value of 0, and for the set of input rows that has same set key, the first input value for the column is copied to the output column.

The Pivot transformation pivots the data on the basis of set key column value. For the same value of set key, the Pivot transformation merges multiple rows into a single row and pivots the input rows into columns. This implies that if the data is not sorted to list the same set key values in one collection of rows, this transformation will output the same key values multiple times. In our example, to get only one record for a SalesOrderID, the data must be sorted on SalesOrderID. However, if the data is not sorted on SalesOrderID, this transformation will generate multiple records for same SalesOrderID, as it will pivot the rows to columns each time the value of SalesOrderID changes.

All this may appear quite complex, but it is not that complicated when it comes to configuring the Pivot transformation. Let's see how to use this transformation to pivot data from an Excel worksheet.

Hands-On: Pivoting Sales Order Records in an Excel Worksheet

The records exported from the Sales order database to an Excel worksheet are in the normalized form—i.e., one sales order number appears in multiple rows to store details for the products ordered against it. The sales manager wants to see details of all the products ordered against each sales order in a single row.

Before starting this exercise, open the C:\SSIS\RawFiles\SalesOrders.xls file to verify that the file has only one worksheet labeled *Normalized*. This exercise adds another worksheet to this file; if it already has two worksheets, delete the second worksheet and then start this exercise. Also, if you are using the provided package code, you may get a validation error, as the Excel Destination used in the package looks for the worksheets during this exercise. In this case, leave the worksheets as is.

Method

In this exercise, you will be using a Pivot transformation to transform the given data to the required format and will put the pivoted data in a new worksheet. As Pivot transformation pivots the data every time the set key column value changes, you will need to sort the data before the Pivot transformation.

Exercise (Add Connection Manager and Data Flow Task)

You will start this exercise with adding a new package to the Data Flow transformations project, and then adding an Excel Connection Manager to it.

1. Open the Data Flow transformations project in BIDS. Right-click the SSIS Packages in the Solution Explorer and choose New SSIS Package. This will add a new SSIS package called Package1.dtsx.
2. Rename the Package1.dtsx package to **Pivoting SalesOrders.dtsx**.
3. Right-click in the Connection Managers area and choose New Connection from the context menu. Select the Excel Connection Manager type from the list in the Add SSIS Connection Manager dialog box and click Add. Next, Type **C:\SSIS\RawFiles\SalesOrders.xls** in the Excel file path field in the Excel Connection Manager dialog box. Leave the Excel Version selected as Microsoft Excel 97-2003 and see that the check box for First Row has column names checked. Click OK to add the Excel Connection Manager in the Connection Managers area. Rename it as **SalesOrders Connection Manager**.
4. Drag the Data Flow Task from the Toolbox and drop it onto the Control Flow Designer surface. Rename this task **Transforming SalesOrders**. Double-click it to open the Data Flow tab and configure this task.

Exercise (Configure the Data Flow Task)

To configure a data flow for pivoting SalesOrders data, here you will add an Excel source to extract data from a normalized worksheet of the SalesOrders.xls file and then sort this data on the SalesOrderID, as a Pivot transformation requires all the rows having the same set key to be together in a sorted set for merging them to a single row. After sorting the data, you will configure the Pivot transformation to pump the pivoted data through to an Excel destination that will write the pivoted output to a new Excel worksheet.

5. From the Toolbox, drag and drop the Excel source onto the Data Flow Designer surface. Rename this adapter **Normalized Data Source**. Double-click the Normalized Data Source to open the Excel Source Editor. The SalesOrders Connection Manager will be listed in the OLE DB Connection Manager field for you. Select the Normalized$ in the "Name of the Excel sheet" field.
6. Go to the Columns page, and verify that all the three fields have been selected from the Available External Columns. Click OK to close the editor.
7. Drop the Sort transformation from the Toolbox onto the Data Flow Designer surface just below the Normalized Data Source and join both the components using green connector. Rename it as **Sort on SalesOrderID** and double-click to open the Sort Transformation Editor. Click to select the check box before the SalesOrderID column. This column will appear in the lower half of the dialog box with Sort Type as ascending and Sort Order equal to 1. Leave these setting as is and click OK to close this editor.
8. Drop the Pivot from the Toolbox onto the Designer surface just below the Sort on SalesOrderID. Connect the two transformations using the green arrow. Double-click the Pivot to open the Advanced Editor for Pivot, as Pivot doesn't have a custom UI. In the Component Properties tab, change the Name field to **Pivot on ProductName**.
9. Go to the Input Columns tab and select all three columns.
10. Move on to the Input and Output Properties tab. Expand the Input Columns under Pivot Default Input on the left side of the dialog box to reveal the three columns you have selected in the Input Columns tab. Click the SalesOrderID to list the properties of this column on the right pane of the dialog box. Scroll down in the properties to locate the PivotUsage property. Assign a value of 1 to PivotUsage, indicating that this field will be treated as a set key. Now, click the ProductName column and assign a value of 2 to its PivotUsage property to indicate that this field is a pivot key field and the distinct values in this field will create corresponding columns in the output. Next, click the OrderQuantity column and assign a value of 3 to its PivotUsage property to indicate that the values from this field will be populated in the columns generated by the pivot key column.
11. Expand Pivot Default Output and then click the Output Columns. You will notice that no output column appears in this transformation yet. Click Add Column and rename the newly added column **SalesOrderID**. This output column needs to be linked to an input column. In the properties of this column, locate the SourceColumn property. This property holds the lineage identifier of an input column and tells the Pivot transformation to populate the output column using values from the specified input column. To specify the source in this column, click the SalesOrderID column under Input Columns and note the LineageID (not the ID). Specify this value of LineageID in the SourceColumn property of SalesOrderID output field. The value *32* shown in the SourceColumn property in Figure 10-10 is the LineageID of the SalesOrderID input column.

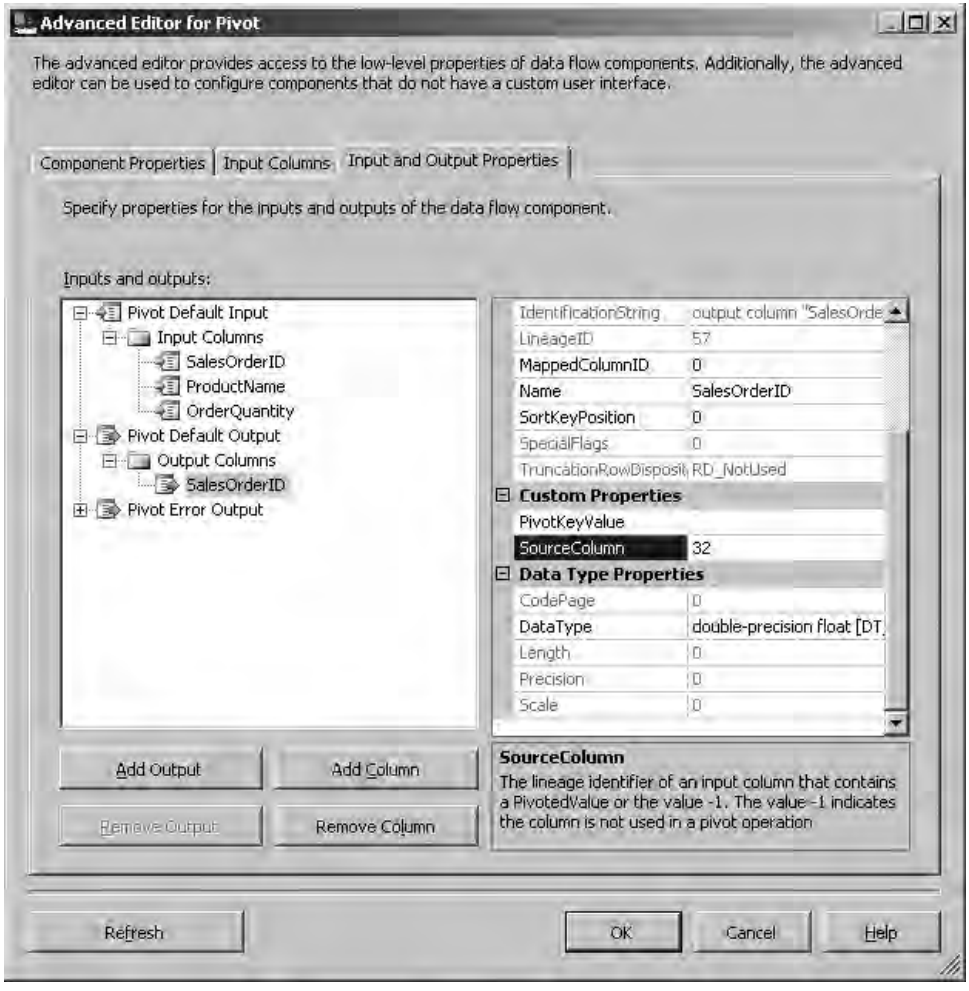


Figure 10-10 *Setting LineageID on the SourceColumn property*

12. As you did in Step 11, add 10 more columns in the Output Columns and rename them as per the following table. Also, assign values to the PivotKeyValue and SourceColumn properties of the new output columns. Remember that PivotKeyValue is one of the distinct values of pivot key column and the SourceColumn indicates from which input column the data will populate the concerned output column. The new columns added here will be populated by

OrderQuantity input column, so the LineageID of OrderQuantity will be used to populate all of them.

Output Column	Renamed to	PivotKeyValue	SourceColumn
Column	Mountain-100	Mountain-100	LineageID Value of OrderQuantity column
Column1	Mountain-200	Mountain-200	LineageID Value of OrderQuantity column
Column2	Mountain-300	Mountain-300	LineageID Value of OrderQuantity column
Column3	Road-150	Road-150	LineageID Value of OrderQuantity column
Column4	Road-250	Road-250	LineageID Value of OrderQuantity column
Column5	Road-350	Road-350	LineageID Value of OrderQuantity column
Column6	Touring-1000	Touring-1000	LineageID Value of OrderQuantity column
Column7	Touring-2000	Touring-2000	LineageID Value of OrderQuantity column
Column8	Touring-3000	Touring-3000	LineageID Value of OrderQuantity column
Column9	Sport-100	Sport-100	LineageID Value of OrderQuantity column

Refer to Figure 10-11 to see how this will look. In the figure, the value *42* in the SourceColumn is the LineageID of OrderQuantity input column. Click OK to close this transformation.

13. Drop the Excel destination from the Toolbox just below the Pivot on ProductName and connect the two transformations using the green arrow. Rename it as **Pivoted Data Destination**. Double-click the Pivoted Data destination to open the Excel Destination Editor. You will use the same connection manager and the same Excel workbook. But you will add a new worksheet called Pivoted in the Excel workbook to store the pivoted data. Click the New button opposite the "Name of the Excel sheet" field and you will see a script to create a new worksheet in the Excel workbook. Change the Pivoted Data Destination just after CREATE TABLE statement to *Pivoted* only. This will create a new worksheet named Pivoted in the SalesOrders.xls file. Click OK and select the Pivoted sheet in the "Name of the Excel sheet" field.
14. Go to the Mappings page. As you click Mappings, you should see all the mappings created for you automatically. Click OK to close the editor for this component.

Exercise (Add Data Viewers and Execute the Package)

After having configured all the data flow components, you are ready to execute the package. However, in this exercise, you will also add data viewers before and after the Pivot transformation to see how the data has been pivoted.

15. Double-click the data flow path connecting Sort on SalesOrderID and Pivot on ProductName. In the Data Flow Path Editor, click the Data Viewers page, and then click Add and add a grid type data viewer. Click OK twice to close the editor window.

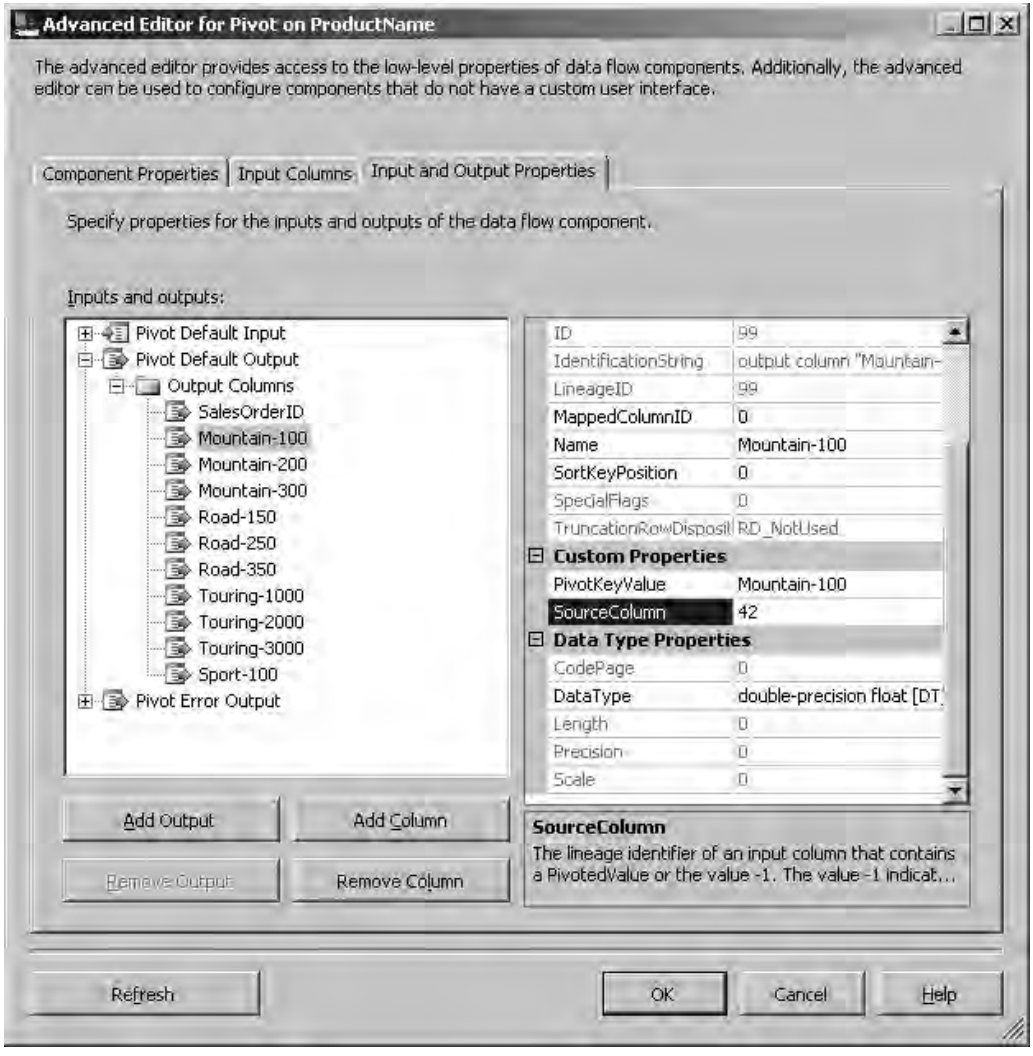


Figure 10-11 *Configuring output columns for the Pivot transformation*

16. Similarly, add another grid type data viewer on the data flow path between Pivot on ProductName and Pivoted Data Destination.
17. Press F5 to execute the package. As the Normalized Data Source extracts data and passes to the Sort on SalesOrderID, you can see Normalized Data Source turns green, indicating a successful extraction of data and a total of 2,187 records extracted. The next two components will appear in yellow, as the data viewer is holding the execution process for you to check the data. Click the Detach button on the first data

viewer to let the execution proceed. As you detach the first data viewer, you will see the second data viewer pop up with populated data and the Pivot on ProductName outputting 1,302 rows. As the data passes the Sort on SalesOrderID, it will turn green to indicate a successful sort operation (see Figure 10-12). When you are done checking the data, click the Detach button on the second data viewer and let the package complete successfully.

18. Stop debugging by pressing SHIFT-F5. Save and close the project. Open the SalesOrders.xls file and check out the Pivoted worksheet to see how the data has been pivoted.

Review

You've used a Pivot transformation to convert normalized data to a less normalized form. During this exercise, you sorted the data before sending it to the Pivot transformation

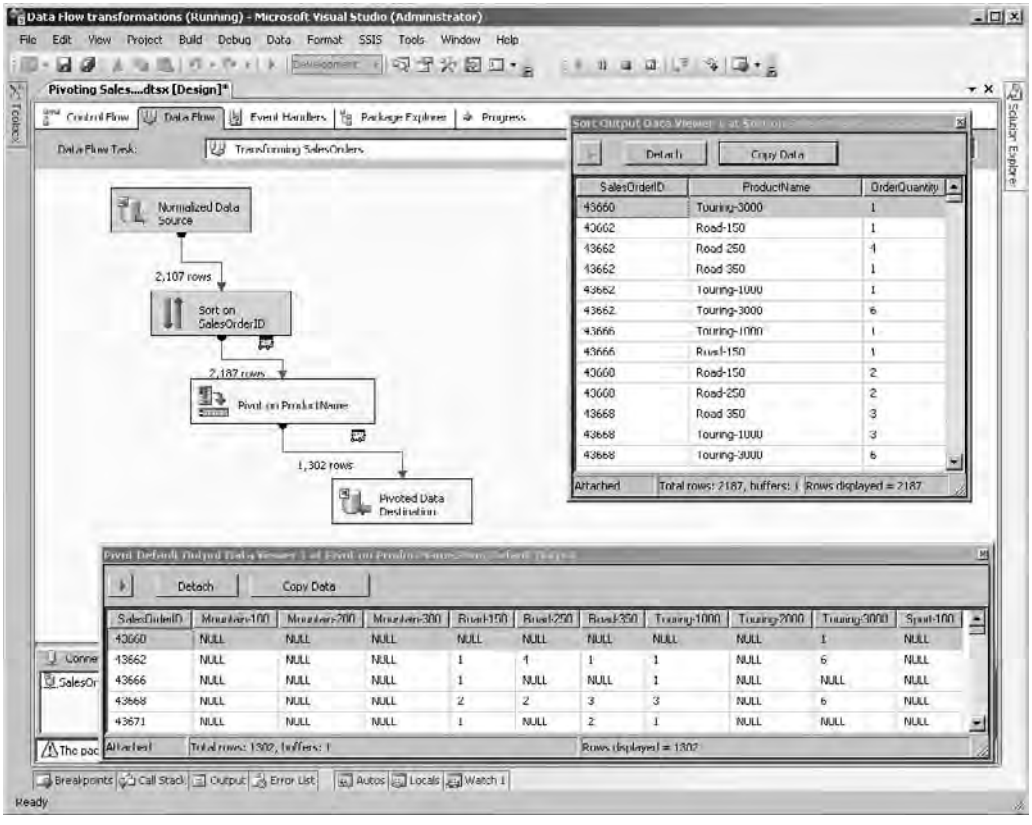


Figure 10-12 *Data viewers showing data before and after Pivot transformation*

and created output columns for the pivoted data. Finally, you created a new Excel worksheet using an Excel destination. Note that you've used only one connection manager to connect to Excel file for both extractions of data using a source adapter and loading of data using a destination adapter.

Unpivot Transformation

This transformation works in an opposite way to the Pivot transformation and converts a denormalized data set into a more normalized version—i.e., one row may be broken down into multiple atomic rows so that they can be stored in a relational database. We will use the data you derived in the last exercise, in which each row lists multiple products against a single SalesOrderID. When you run this transformation, the records will be broken up into multiple rows containing the same value for SalesOrderID, but having only one product in each row. To support its functions, this transformation uses one input, one output, and an error output.

This transformation has a custom user interface that is much simpler than that of the Pivot transformation. To get an idea of how to configure this transformation, consider the pivoted data that you created in the preceding exercise. To get the data formatted in the normalized form—i.e., from where you started in the last Hands-On—you will configure the Unpivot transformation, as shown in Figure 10-13.

Following is the step-by-step method you will use to get the pivoted data back to normalized form:

1. Use the Excel source to bring the Pivoted worksheet data in the data flow.
2. Add an Unpivot transformation and configure it. Let the set key column pass through the transformation as is, which is SalesOrderID in this case. Then select all the columns that you want to unpivot. As you select the check boxes for Available Input Columns, the Input Column and the Pivot Key Value columns will be filled in using the column name selected. After that, manually fill in the Destination Column name where you want the values of input columns to be populated, which is OrderQuantity in this example. Last, specify the column name in which you would like pivot key values to be populated. These configurations indicate that the input column names will be converted into the values specified in the Pivot Key Value, which will then be populated in pivot key value column (ProductName in this case), and input column values will be populated in a new column specified in the Destination Column, which is OrderQuantity column in this example.
3. Finally, add an Excel destination to collect the normalized data in an Excel worksheet.

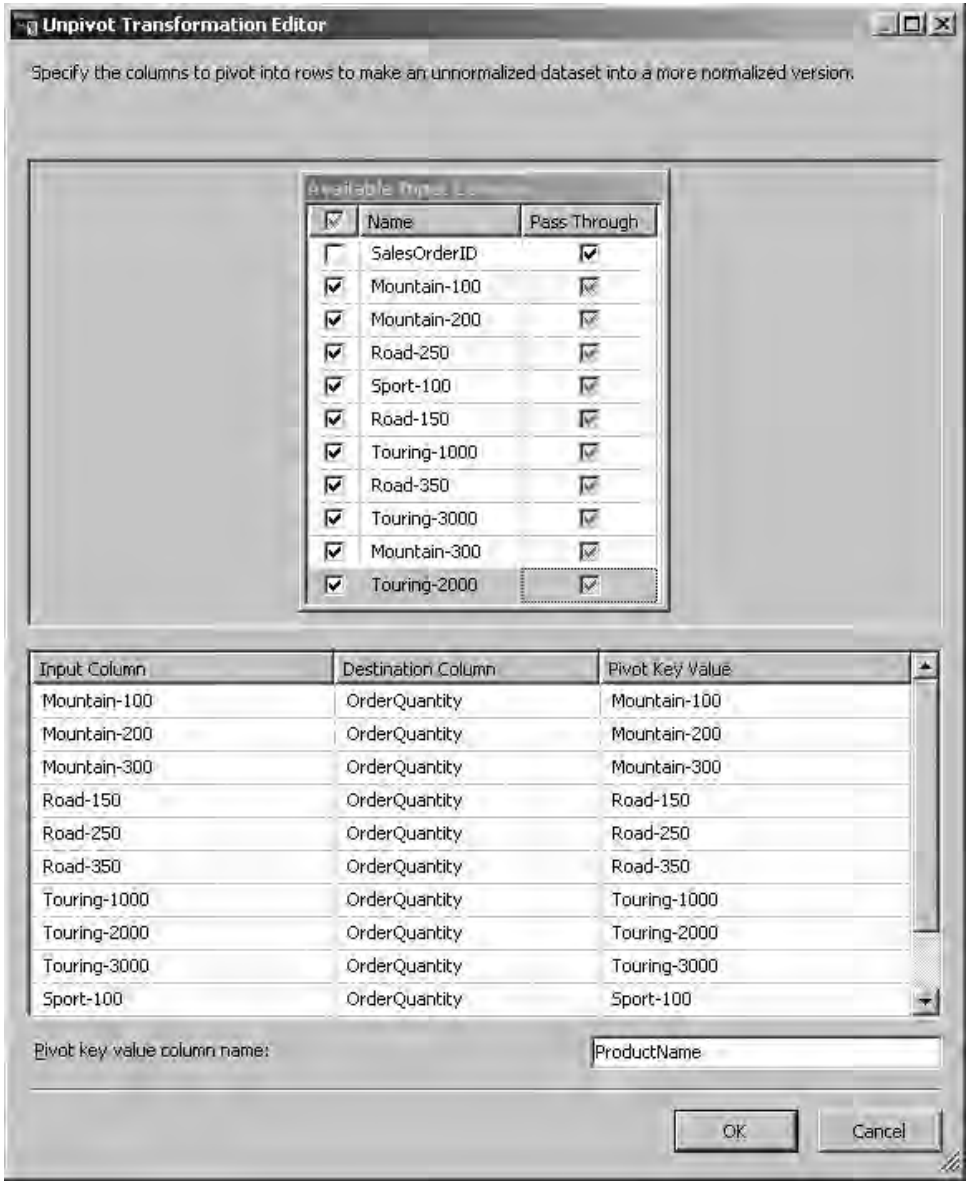


Figure 10-13 *Configurations for the Unpivot transformation*

Aggregate Transformation

The Aggregate transformation is an *asynchronous* transformation that helps you to perform aggregate operations such as SUM, AVERAGE, and COUNT. To perform these aggregate operations, you need to have a complete data set. The Aggregate transformation consumes all the rows before applying any aggregation and extracting the transformed data. Because of being an asynchronous transformation, the output data, which most likely has a new schema, is populated in the new memory buffers.

The Aggregate transformation can perform operations such as AVERAGE, COUNT, COUNT DISTINCT, GROUP BY, selecting a minimum or maximum from a group, and SUM on column values. The aggregated data is then extracted out in the new output columns. The output columns may also contain the input columns, which form part of the groupings or aggregations.

When you select a column in the Aggregate Transformation Editor and click in the Operation field, you will see a list of operations that coincide with the type of the column you selected. This makes sense as the aggregate operations require appropriate column types—for example, a SUM would work on a numeric data type column and would not work with a string data type column. Following are the operation descriptions in detail:

- **AVERAGE** This operation is available only for numeric data type columns and returns the average of the column values.
- **COUNT** Counts the number of rows for the selected column. This operation will not count the rows that have null values for the specified column. In the Aggregate Transform Editor, a special column (*) has been added that allows you to perform the COUNT ALL operation to count all the rows in a data set, including those with null values.
- **COUNT DISTINCT** Counts the number of rows containing distinct non-null values in a group.
- **GROUP BY** This operation can be performed on any data type column and returns the data set in groups of row sets.
- **MAXIMUM** This operation can be performed on numeric, date, and time data type columns and returns the maximum value in a group.
- **MINIMUM** This operation can be performed on numeric, date, and time data type columns and returns the minimum value in a group.
- **SUM** This operation is available only for numeric data type columns and returns the sum of values in a column.

The Aggregate transformation's user interface provides features and options to configure aggregations that we will cover in the following Hands-On exercise, as they will be easier to understand as you work with them. Before we dive in, consider the following:

- You can perform multiple aggregate operations on the same set of data. For example, you can perform SUM and AVERAGE operations on a column in the same transformation. As the result from these two different aggregate operations will be different, you must direct the results to different outputs. This is fully supported by the transformation, and you can add multiple outputs to this transformation.
- The null values are handled as specified in the SQL-92 standard, that is, in the same way they are handled by T-SQL. The COUNT ALL operation will count all the rows including those containing null values, whereas COUNT or COUNT DISTINCT operations for a specific column will count only rows with non-null values in the specified columns. In addition, GROUP BY operation puts null values in a separate group.
- When an output column requires special handling because it contains an oversized data value greater than four billion, or the data requires precision that is beyond a float data type, you can set the IsBig property of the output column to 1 so that the transformation uses the correct data type for storing the column value. However, columns that are involved in a GROUP BY, MINIMUM, or MAXIMUM operation cannot take advantage of this.

Hands-On: Aggregating SalesOrders

The SalesOrders.xls file has been extended with the pricing information by adding unit price and total price columns. The extracted data has been saved as SalesOrdersExtended .xls. From this extended sales orders data that has a price listed for each product against the SalesOrderID, you are required to aggregate a total price for each order and calculate the average price per product and the number of each product sold.

Before starting this exercise, open the SalesOrdersExtended.xls Excel file to verify that the file has only one worksheet named SalesOrders. This exercise adds two more worksheets in it; if the file has additional sheets, delete them before you start this exercise. Also, if you are using the provided package code, you may get a validation error, as the Excel Destination used in the package looks for the worksheets during this exercise. In this case, leave the worksheets as is.

Method

As with the Pivot transformation Hands-On exercise, you will use an Excel file to access the data from a worksheet and create two new worksheets in the same Excel file to extract the processed data. You will configure a Data Flow task and an Aggregate transformation.

Exercise (Add Connection Manager and Data Flow Task to a New Package)

Like the previous exercise, you will start this exercise by adding a new package to the Data Flow transformations project and then adding an Excel Connection Manager in to it.

1. Open the Data Flow transformations project in BIDS. Right-click the SSIS Packages in Solution Explorer and choose New SSIS Package. This will add a new SSIS package named Package1.dtsx.
2. Rename Package1.dtsx as **Aggregating SalesOrders.dtsx**.
3. Add an Excel Connection Manager to connect to an Excel file C:\SSIS\RawFiles\SalesOrdersExtended.xls.
4. Add a Data Flow task from Toolbox and rename it **Aggregating SalesOrders**. Double-click it to open the Data Flow tab.

Exercise (Configure Aggregating SalesOrders)

The main focus of this part is to learn to configure an Aggregate transformation. You will configure the Excel source and Excel destination as well complete the data flow configurations.

5. Add an Excel source to extract data from the SalesOrders$ worksheet in the SalesOrdersExtended.xls file. Rename this **Excel Source Sales Orders Data Source**.
6. Drag and drop an Aggregate transformation from the Toolbox onto the Data Flow surface just below the Excel source. Connect both the components with a data flow path.
7. Double-click the Aggregate transformation to open the Aggregate Transformation Editor that displays two tabs—Aggregations and Advanced. In the Aggregations tab, you select columns for aggregations and specify aggregation properties for them. This tab has two display types: basic and advanced. An Advanced button on the Aggregations tab converts the basic display of the Aggregations tab into an advanced display. This advanced display allows you to perform multiple groupings or GROUP BY operations. Click Advanced to see the advanced display. Selecting multiple GROUP BY operations adds rows using multiple Aggregation Names in the advanced display section. This also means that you will be generating different types of output data sets that will be sent to multiple outputs. So, when you add an Aggregation Name, you add an additional output to the transformation outputs.
8. Click the Advanced tab to see the properties you can apply at the Aggregate component level. As you can see, there are configurations to be done in more than one place in this editor. In fact, you can configure this transformation at three levels—the component level, the output level, and the column level.

The properties you define on the Advanced tab apply at a component level, the properties configured in the advanced display of the Aggregations tab apply at the output level, and the properties configured in the column list at the bottom of the Aggregations tab apply at the column level. The capability to specify properties at different levels enables you to configure it for the maximum performance benefits. The following descriptions explain the properties in the Advanced tab:

- **Key Scale** This is an optional property that helps the transformation decide the initial cache size. By default, this property is not used and can have a low, medium, or high value if selected. Using the low value, the aggregation can write approximately 500,000 keys, medium enables it to write about 5 million keys, and high enables it to write approximately 25 million keys.
- **Number Of Keys** This optional setting is used to override the value of a key scale by specifying the exact number of keys that you expect this transformation to handle. Specifying the keys upfront allows the transformation to manage cache properly and avoids reorganizing the cache at run time; thus it will enhance performance.
- **Count Distinct Scale** You can specify an approximate number of distinct values that the transformation is expected to handle. This is an optional setting and is unspecified by default. You can select low, medium, or high values. Using the low value, the aggregation can write approximately 500 thousand distinct values, medium enables it to write about 5 million distinct values, and high enables it to write approximately 25 million distinct values.
- **Count Distinct Keys** Using this property, you can override the count distinct scale value by specifying the exact number of distinct values that the transformation can write. This will avoid reorganizing cached totals at run time and will enhance performance.
- **Auto Extend Factor** Using this property, you can specify a percentage value by which this transformation can extend its memory during runtime. You can use a value between 1 and 100 percent—the default is 25 percent.

9. Go to the Aggregations tab, and select SalesOrderID and TotalPrice from the Available Input Columns. As you select them, they will be added to the columns list below as SalesOrderID with GROUP BY operation and TotalPrice with SUM operation.
10. Click Advanced to display the options for configuring aggregations for multiple outputs. You will see Aggregate Output 1 already configured using SalesOrderID as a GROUP BY column. Rename Aggregate Output 1 as **Total Per Order**. Next, click in the second row in the Aggregation Name field and type **Products Sold and Average Price** in the cell. Then select the ProductName, OrderQuantity, and UnitPrice columns from the Available Input Columns list. These columns

will appear in the columns list with default operations applied to them. Change these operations as follows: GROUP BY operation to the ProductName column, SUM operation to the OrderQuantity column, and AVERAGE operation to the UnitPrice column, as shown in the Figure 10-14.

Note that you can specify a key scale and keys in the advanced display for each output, thus enhancing performance by specifying the number of keys the output is expected to contain. Similarly, you can specify Count Distinct Scale and Count Distinct Keys values for each column in the list to specify the number of distinct values the column is expected to contain.

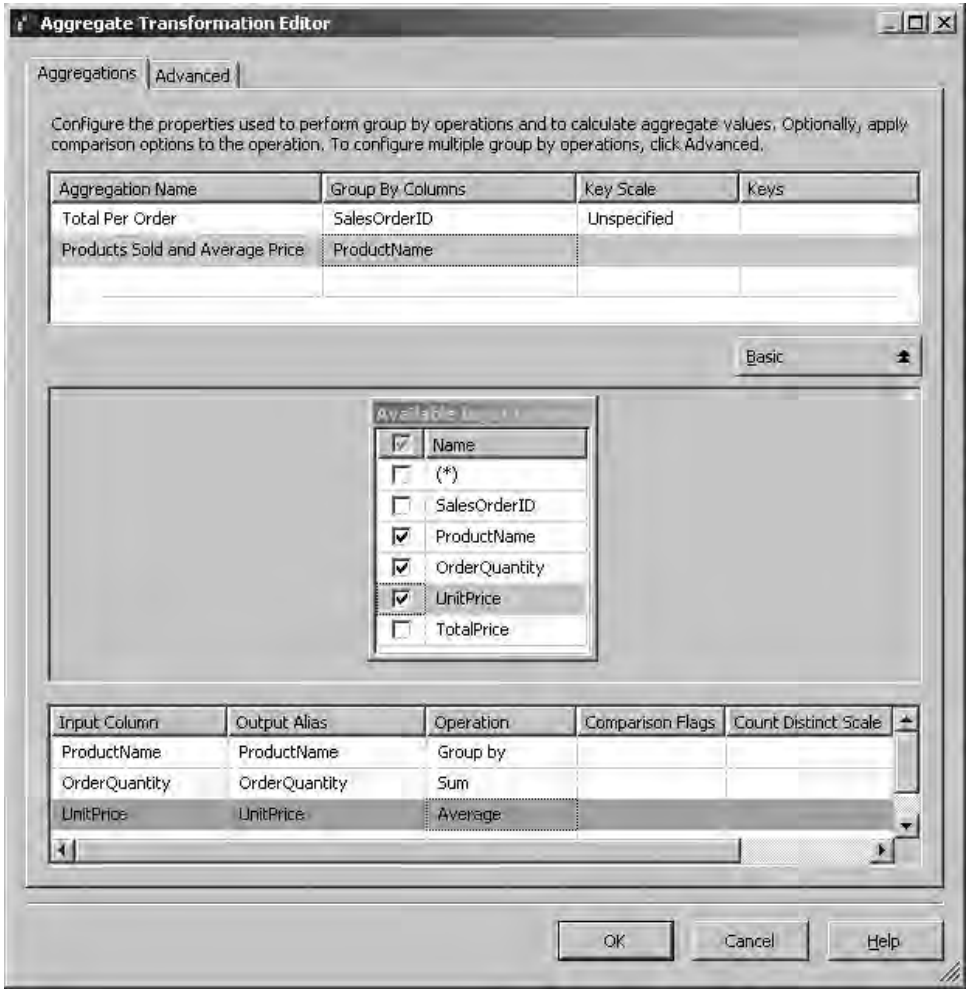


Figure 10-14 *Configuring Aggregate transformation for multiple outputs*

11. You're done with the configuration of Aggregation transformation. Click OK to close the editor. Before you start executing the package, check out one more thing. Open the Advanced Editor for Aggregate transformation and go to the Input and Output Properties tab. You'll see the two outputs you created earlier in the advanced display of Aggregations tab of the custom editor. Expand and view the different output columns. Also, note that you can specify the IsBig property in the output columns here, as shown in Figure 10-15. Click OK to return to the Designer.

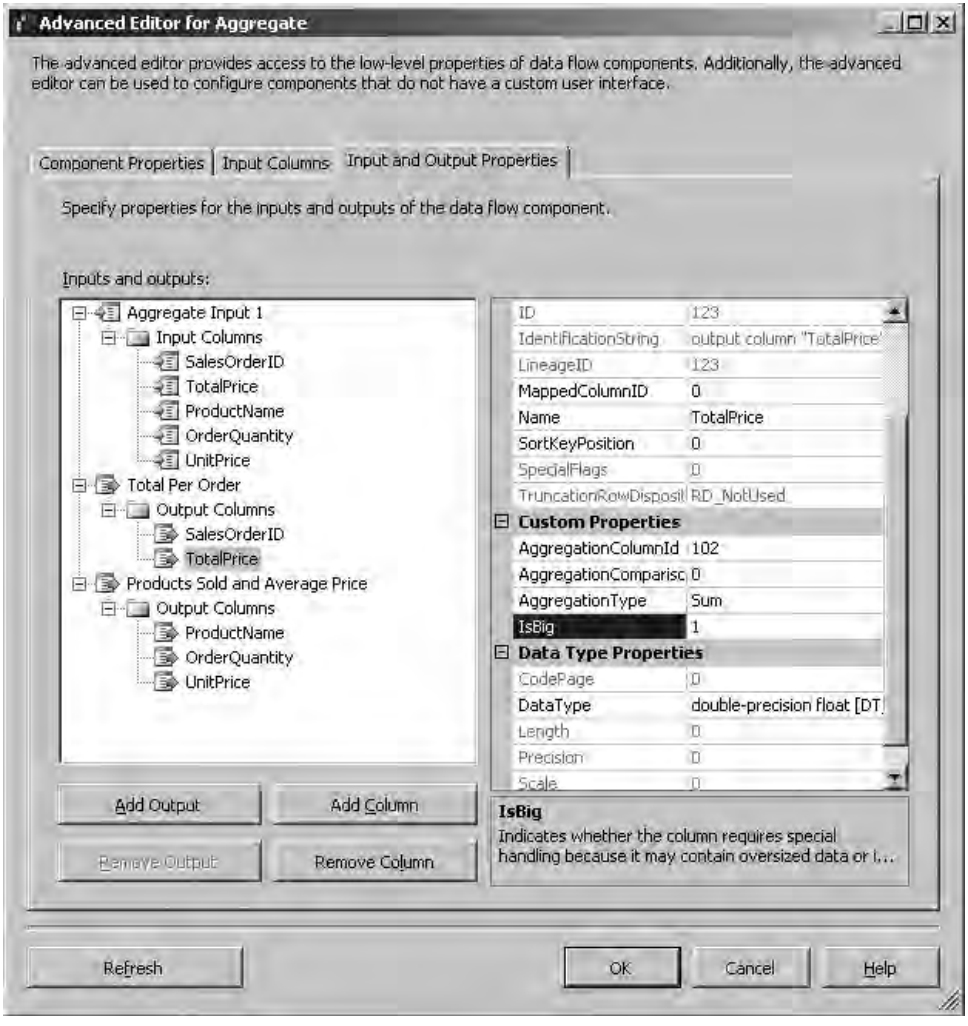


Figure 10-15 *Multiple outputs of Aggregate transformation*

12. Let's direct the outputs to different worksheets in the same Excel file. Add an Excel destination on the left, just below the Aggregate transformation, and drag the green arrow from Aggregate transformation to Excel destination. As you drop it on the Excel destination, an Input Output Selection dialog box will pop-up, asking you to specify the output you want to connect to this destination. Select Total Per Order in the Output field and click OK to add the connector.
13. Double-click the Excel Destination and click the New button opposite to the "Name of the Excel sheet" field to add a new worksheet in the Excel file. In the Create Table dialog box, change the name of the table from Excel Destination to **TotalPerOrder** and click OK to return to the Excel Destination Editor dialog box. Select the TotalPerOrder sheet in the field. Next, go to the Mappings page and the mappings between the Available Input Columns with the Available Destination Columns will be done for you by default. Click OK to close the Destination Editor. Rename this destination **Total Per Order**.
14. Much as you did in Steps 12 and 13, add another Excel destination below the Aggregate transformation on the right side (see Figure 10-16) and rename it

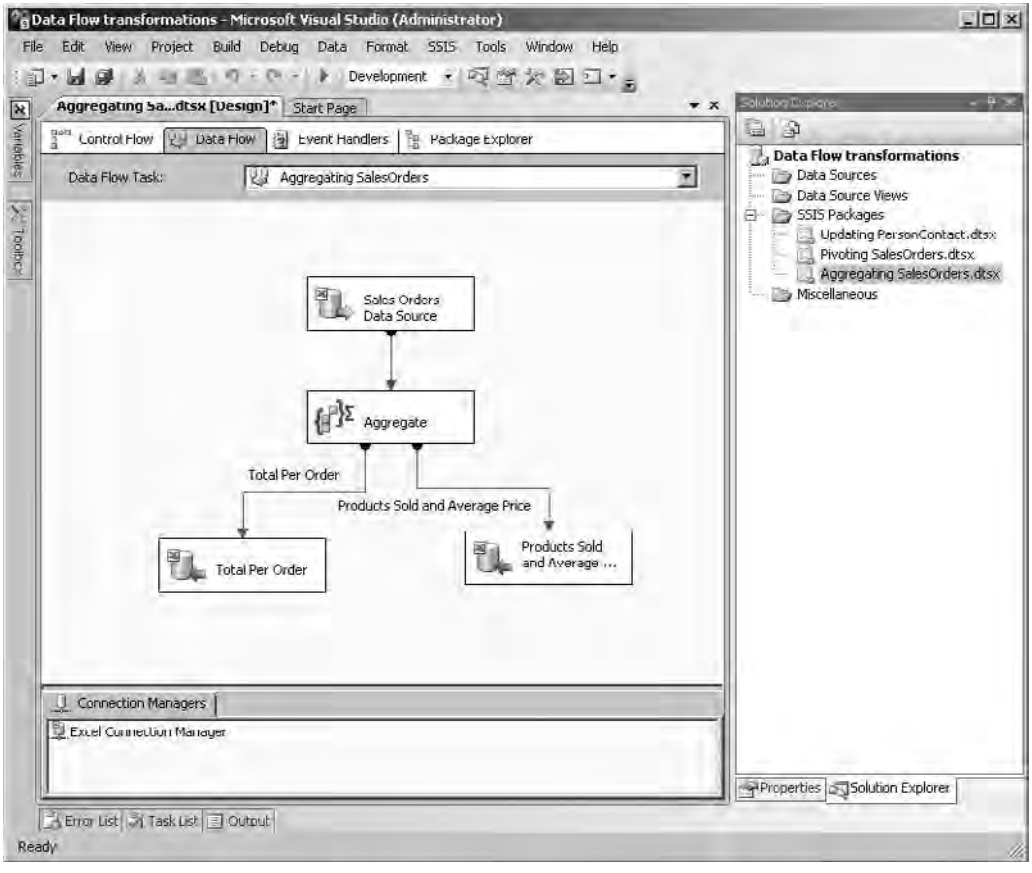


Figure 10-16 *Aggregating the SalesOrders package*

Products Sold and Average Price. Connect the Aggregate transformation to the Products Sold and Average Price destination using the second green arrow. Add a new worksheet by clicking the New button next to the "Name of the Excel sheet" field by the name of **ProductsSoldandAveragePrice**. Go to the Mappings page to create the mappings. Click OK to close the editor.

Exercise (Run the Aggregations)

In the final part of this Hands-On, you will execute the package and see the results. If you wish, you can add data viewers where you would like to see the data grid.

15. Press F5 to run the package. The package will complete execution almost immediately. Stop the debugging by pressing SHIFT-F5. Save all files and close the project.
16. Explore to the C:\SSIS\RawFiles folder and open the SalesOrdersExtended.xls file. You will see two new worksheets created and populated with data. Check out the data to validate the Aggregate transformation operations.

Review

You've done some aggregations in this exercise and have added multiple outputs to the Aggregate transformation. You've also learned that the Aggregate transformation can be configured at the component level by specifying the properties in the Advanced tab, can be configured at the output level by specifying keys for each output, and can also be configured at the column level for distinct values each column is expected to have. You can achieve high levels of performance using these configurations. However, if you find that the transformation is still suffering from memory shortage, you can use Auto Extend to extend the memory usage of this component.

Audit Transformations

Audit transformations are important as well. This category includes only two transformations in this release. The Audit transformation includes environmental data such as system data or login name into the pipeline. The Row Count transformation counts the number of rows in the pipeline and stores the data to a variable.

Audit Transformation

One of the common requirements of data maintenance and data warehousing is to timestamp the record whenever it is either added or updated. Generally in data marts, you get a nightly feed for new as well as updated records and you want to timestamp the records that are inserted or updated to maintain a history, which is also quite

helpful in data analysis. By providing the Audit transformation, Integration Services has extended this ability and allows environment variable values to be included in the data flow. With the Audit transformation, not only can you include the package execution start time but you can include much more information—for example, the name of the operator, computer, or package to indicate who has made changes to data and the source of data. This is like using Derived Column transformation in a special way. To perform the assigned functions, this transformation supports one input and one output. As it does not perform any transformation on the input columns data (rather it just adds known environment values using system variables), an error in this transformation is not expected, and hence it does not support an error output.

The Audit transformation provides access to nine system variables. Following is a brief description of each of these variables:

- **ExecutionInstanceGUID** Each execution instance of the package is allocated a GUID contained in this variable.
- **PackageID** Represents the unique identifier of the package.
- **PackageName** A package name can be added in the data flow.
- **VersionID** Holds the version identifier of the package.
- **ExecutionStartTime** Include the time when the package started to run.
- **MachineName** Provides the computer name on which the package runs.
- **UserName** Adds the login name of the person who runs the package.
- **TaskName** Holds the name of the Data Flow task to which this transformation belongs.
- **TaskID** Holds the unique identifier of the Data Flow task to which this transformation belongs.

When you open the editor for this transformation after connecting an input, you will be able to select the system variable from a drop-down list by clicking in the Audit Type column, as shown in Figure 10-17. When you select a system variable, the Output Column Name shows the default name of the variable, which you can change. This Output Column will be added to the transformation output as a new output column.

Row Count Transformation

Using the Row Count transformation, you can count the rows that are passing through the transformation and store the final count in a variable that can be used by other components, such as a script component and property expressions or can be useful

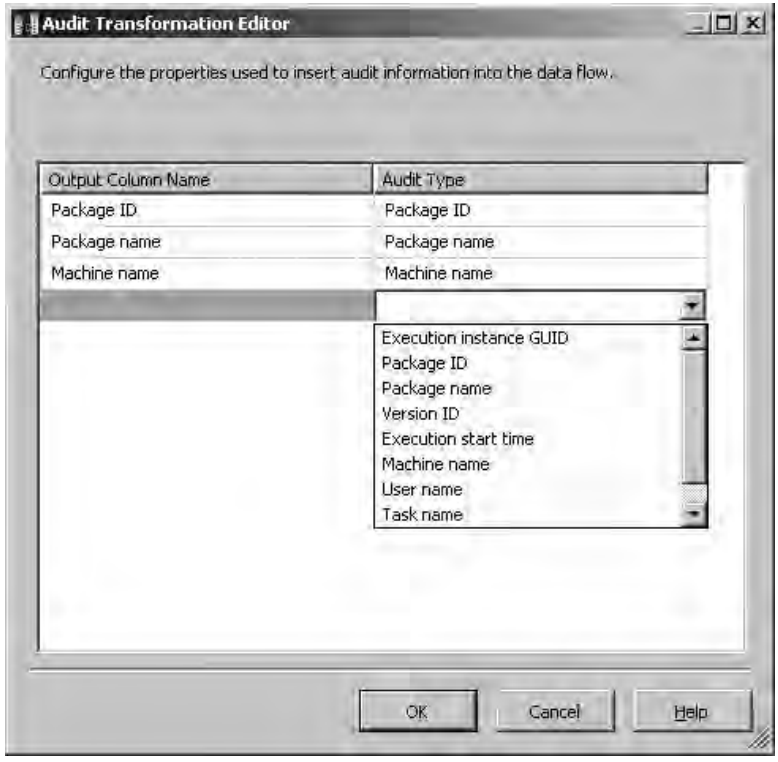


Figure 10-17 *Configuring Audit transformation*

in auditing. This transformation uses the Advanced Editor to expose its properties. To store the count of rows into a variable, you need to define a variable before you can use this task. The variable must be in the scope of the Data Flow task to which the transformation belongs. To count the rows in the data flow and update the variable value, you simply select the variable in the VariableName field of the Component Properties tab. You can actually have multiple RowCount transformations writing to the same variable, for instance, if you are updating auditing information about loading of data rows you might choose to use same variable in multiple RowCount transformations. However, you need be aware of the fact that the multiple RowCount transformations using same the variable could update variable value multiple times, and hence your auditing could get wrong values, as the variable will keep the last value that it is updated with.

Business Intelligence Transformations

The Business Intelligence transformations enable you to maintain a Slowly Changing Dimension (SCD); perform data cleaning, data standardization, and text mining operations; and run data mining prediction queries against data mining models. The Slowly Changing Dimension (SCD) transformation is slightly complex to understand, but it provides a configuration wizard that makes the task of loading a dimension table much easier. You will work through a Hands-On exercise to use an SCD transformation, and you will perform another Hands-On exercise to remove duplicates from data using Lookup, Fuzzy Lookup, and Fuzzy Grouping transformations.

Slowly Changing Dimension Transformation

SCD helps you manage slowly changing data attributes in a data warehouse. To describe this transformation clearly and for the benefit of those who are new in data warehousing, let's start with some conceptual details. When a database for a data warehouse is built using the dimensional modeling design, the data is sliced across many dimensions, such as time, location, and product, and is stored in two types of tables called the *fact table* and the *dimension table.* A dimension table stores records relevant for that particular dimension, and a fact table contains measures for the dimensions. For example, a product dimension table contains information about products such as product category, model details, and product features. A sales fact table contains measures of sales across different dimensions, such as total sales of a product, total sales in a month, and total sales at a location. The advantage of dimensional data modeling is that you can aggregate data across multiple dimensions—for example, total sales of a product at a location in a year. To learn more about dimensional modeling, refer to Chapter 12.

One of the issues you will face while maintaining a data warehouse is how to handle changes to the dimensional data across time. For example, contacts may change their addresses, products can change over time, new products may be introduced, some old products may be removed, or your company may decide to reorganize its sales regions. These changes may pose some challenges when you're tasked with maintaining the history of these changes. In order to create complex business reports that contain aggregations over a period of time, maintaining history in a data warehouse or online analytical processing (OLAP) system is much more important compared to online transaction processing (OLTP) systems, which are generally designed to represent the current state. For example, if a sales representative is allocated a new sales region, the commission for the sales she has made in the previous region should still go to her; this makes it necessary to keep a history of changes to her sales region.

To maintain a history for changing data while dealing with restrictions imposed by data complexity and storage limitations, information engineers have been adopting

different approaches to different types of data. When maintaining a history for a particular data type is not important, an existing record simply gets overwritten by a new record. This is the simplest form of handling a slowly changing dimension and is classified as Type 1. When preserving the history of changing attributes of data is critical to business, the new record is added to the dimension along with the existing original record. This is classified as a Type 2 slowly changing dimension. This is a more commonly used form that allows you to maintain accurate historical data; however, if the dimension sizes become too big and the number of rows is too high, you may have to work on storage and performance issues. In a Type 3 slowly changing dimension, the issue of additional rows with each change is addressed by adding new columns for the current value and when the current value became effective. Hence, the dimension contains the original value and the current value, irrespective of how many changes have occurred in between, thus limiting the ability to register changes accurately; this does overcome the limitations of Type 2, however. In Integration Services, the slowly changing dimension transformation doesn't support Type 3 changes.

The SCD transformation supports up to six different outputs to output records for different types of processing before loading them into the dimension table. These outputs have been assigned different names on the basis of different Change Type records they carry. The outputs are discussed in more detail later in this chapter, but here are brief descriptions for you:

- **Changing Attributes Updates Output** Used for the rows where attributes of dimension members are changing.
- **Fixed Attribute Output** Used for the rows where the attributes of dimension members are not allowed to change—i.e., the attributes are fixed.
- **Historical Attributes Inserts Output** Used for the rows where the attributes of dimension members are required to keep history.
- **Inferred Member Updates Output** Used for the rows where the attributes of the dimension members are to be updated as the row was earlier inserted as an inferred member with most of the attributes as blank.
- **New Output** Used for the new member rows and for the current member rows that are to be created as new where the existing member has been updated as expired to keep history.
- **Unchanged Output** Used for the existing members in the dimension table where no attribute has changed and no work is performed by the SCD; such rows are diverted to unchanged output and no data flow is created by default; hence, this acts as a sink for unused rows. However, if you want to capture these rows, you can collect them by creating a data flow path connected to this output.

You can use the Slowly Changing Dimension transformation to update and insert records in data warehouse dimension tables for different types of changes. This transformation contains a wizard that helps you configure various branches relevant to the change types. You set various change types on the columns and select attribute options in a wizard to configure the SCD transformation. When you finish configurations in the end, the wizard configures the SCD transformation to send the different types of records based on their change type settings to different outputs. But the wizard does not stop at simply outputting the records; it also creates downstream data flow to each used output by adding the required components to load data to the SCD table. The great thing about this is that you have complete flexibility to modify the data flow created by the SCD Wizard.

When you connect an input to this transformation in the data flow and double-click it, you invoke the Slowly Changing Dimension Wizard. The SCD Wizard allows you to select a dimension table and specify business keys to map transformation input columns to the dimension table columns. Then you can manage changes in the various columns by specifying a change type for each dimension column. The following change types are available while using the SCD Wizard:

- **Fixed Attribute** The Fixed attribute change type for a column indicates that the value in the column is not expected to change and any changes on that column should be treated as errors. At run time, the SCD transformation performs a lookup for the incoming row against the dimension table to check for a match. If SCD doesn't find any match for a particular row, it diverts that row to the New Output; however, if a match is found and the matching key contains the columns with Fixed attribute change types, the SCD transformation diverts the row to the Fixed Attribute Output. As no change is expected in case of Fixed attribute change type columns, such changes are not expected to be applied to the dimension table, hence the SCD Wizard creates no data flow for this output. However, if you want to capture these rows, you can create a data flow and connect to the Fixed Attribute Output of the SCD transformation. You can also specify to fail the transformation if changes are detected in a Fixed attribute in the wizard.
- **Changing Attribute** The columns that you select for the changing attribute change type are treated as Type 1 changes, and the changed rows will overwrite the existing rows. At run time, the SCD transformation performs a lookup for the incoming row against the dimension table and checks for the match. If a match is found and the matching key has the specified columns with changed values, the transformation directs the row to the Changing Attributes Updates Output. The SCD Wizard also adds an OLE DB Command transformation connected to this output to perform the UPDATE operation on the dimension table. If the SCD

transformation finds a matching row during the lookup operation but the row doesn't contain any change, such rows are diverted to the Unchanged Output and the SCD Wizard doesn't create any data flow for this output by default. However, if you want to capture such rows, you can create a downstream data flow and attach it to this output. Your data warehouse may contain multiple records for a business key, as it keeps the history depending on what kind of changes it has been through. So when an SCD transformation gets a Changing attribute type record, it may match against multiple records. For this, you can select an option in the wizard to indicate whether the task should change all the matching records, including outdated records, when a change is detected in a Changing attribute.

- **Historical Attribute** The Historical attribute change type on a column creates a new record and marks existing record as expired by changing the date fields or setting a flag in an indicator column. These changes are treated as Type 2 changes. At run time, when the SCD transformation finds a matching row that has changes in a Historical attribute, it diverts the row to a Historical Attribute Inserts Output. The SCD Wizard creates a data flow, which first updates the existing record and then inserts the incoming record as a new record with the same business key in the dimension table. In the data flow that the SCD Wizard creates, it adds a Derived Column transformation as the first component that is connected to the Historical Attribute Inserts Output. This component adds a new column to the incoming record as an expiry indicator (e.g., IsCurrent = False). The second component added in this data flow branch is an OLE DB Command transformation that updates the existing record in the dimension table with the newly added column to mark it as expired (Set IsCurrent = False where IsCurrent = True). After updating the existing record, it then directs all the columns sans the newly added indicator column of the incoming record to the New Output data flow path using the Union All transformation so that a new record is added to the dimension table. Refer to Figure 10-23, later in the chapter, for clarity on the data flow it creates. If you find this difficult to understand now, complete the Hands-On exercise and it will make much more sense. The New Output also gets all the new records for which SCD did not find any match within the dimension table. The Union All transformation combines new records and the records coming from the OLE DB Command component of the Historical Attribute Inserts Output data flow path for the New Output. After Union All, another Derived Column transformation is added that adds a new column and the value to indicate that the record is current (IsCurrent = True to indicate the current state of the record) before being added to the dimension table.

In addition to the outputs specified in these Change Type descriptions, SCD includes one more, called *Inferred Member Updates Output*. This output gets all the

inferred members' records. An inferred member record is created in the dimension table with the minimal data, anticipating that more details of the record will arrive when a later loading process runs. An inferred member is created because the fact table contains foreign keys for the dimension tables, and sometimes the loading of the fact table fails because fact data arrives when a key doesn't yet exist in the dimension table. To avoid the failure of a fact table loading process, you should create a record in the dimension table with minimal data so that when the fact arrives earlier than its dimension member, it can still be loaded. This inferred member record is updated later when the attribute data arrives. Support for uploading inferred members details when they arrive is inbuilt in the SCD transformation, and the SCD Wizard allows you to choose one of the following methods (see Figure 10-22, later in the chapter) to identify an inferred member while uploading:

- All columns with a change type are null.
- Use a Boolean column to indicate whether the current record is an inferred member.

You will use the first option if you have created a minimal record in the dimension table while loading the fact table, when all the columns with a change type have null values. Alternatively, you can use a Boolean column to indicate that the incoming record is an inferred member record. After you specify the inferred member identification and finish the SCD Wizard, you will see that the wizard has created a data flow with lot of data flow components connected to different outputs to load the dimension table.

Well, it is time to work through a Hands-On exercise to see how you can load a slowly changing dimension using the SCD Wizard. Later in the exercise you will see that the SCD adds all the downstream components for you that you can change if required. This flexibility takes out the complexity from the development of a package containing SCD and hence makes it quite easy.

Hands-On: Loading a Slowly Changing Dimension

You are tasked with loading the DimCustomer dimension using a SCD transformation from the Customer table. The challenge is that the customers were registered using a variety of methods, and not all customers were allocated a customer number, so the Customer table contains some records that do not have a Customer ID assigned to them. You need to identify those customers as well. The other business requirements are that changes to phone numbers are allowed and you don't need to keep a history for these changes; however, a complete history is to be retained for change of address. As an Email attribute is required for logging on to the customer's portal, this attribute is not allowed to be changed.

Method

The Campaign database contains a Customer table that you will use to load the DimCustomer dimension table. In real life, DimCustomer would be in a different database, but to keep things simple for this exercise, the DimCustomer table has been created in the Campaign database. The exercise is divided in four parts to work with Slowly Changing Dimension.

Exercise (Add Data Flow Task and OLE DB Source to a New Package)

In the first part you will create a new package and add starting components to the package.

1. Open the Data Flow transformations project in BIDS and add a New SSIS Package. Rename this new package as **Loading Slowly Changing Dimension.dtsx**.
2. Drop a Data Flow task from the Toolbox onto the Control Flow surface and rename it **Loading SCD**.
3. Double-click the Loading SCD to go to the Data Flow panel. Drop an OLE DB source from the Toolbox onto the Data Flow surface. Rename the OLE DB source **Customer**.
4. Double-click Customer to open the OLE DB Source Editor. Click the New button next to the OLE DB Connection Manager field. Choose localhost. Campaign in the Configure OLE DB Connection Manager dialog box and click OK. Leave Table Or View selected in the Data Access Mode field. Select the [dbo].[Customer] table from the drop-down list in the "Name of the table or the view" field. Go to the Columns page to map the External Columns to the Output Columns. Click OK to close the OLE DB Source Editor.

Exercise (Identify Customers that Do Not Have a Customer ID)

An SCD transformation needs to have a business key so that it can do a lookup with the dimension table using this key. As a requirement, the SCD transformation doesn't allow this key to contain null values. In this part, you will filter out all the records coming in the pipeline that have a Customer ID equal to Null.

5. From the Toolbox, drag and drop the Conditional Split transformation below the Customer source. Connect both the components by dragging the green arrow from the Customer and dropping it on the Conditional Split component. Rename the Conditional Split component **Customers Filter**.
6. Double-click the Customers Filter component to open the Conditional Split Transformation Editor. The interface of this component is basically an Expression Builder UI. Expand the NULL Functions node in the top-right section of the

dialog box and drag and drop ISNULL(<<expression>>) in the Condition column in the grid. When you click outside the condition field, the expression will turn red, indicating that there is an error with the expression syntax. This will add an output with the name *Case 1* to this component.

7. Expand the Columns node in the top-left box and drag the CustomerID on the <<expression>> part of the ISNULL function so that the Condition field contains ISNULL([CustomerID]). When you click outside the Condition field, the expression turns black, indicating that the expression syntax is correct. The ISNULL function will be true for the null values of CustomerID column and the component will divert rows that evaluate true to expression to the Case 1 output.
8. Click in the Output Name column and change Case 1 to **Customers without CustomerID**. Click in the Default Output Name field and rename it **Customers with CustomerID**, as shown in Figure 10-18. This component has been configured to output records to two outputs on the basis of whether the CustomerID is a null or a non-null value. Click OK to close the editor.
9. Drag an Excel destination onto the Data Flow surface below the Customers Filter. Click the Customers Filter and drag the green arrow onto the Excel Destination. Select Customers without CustomerID in the Output field of the Input Output Selection dialog box. Click OK to return to the Designer.
10. Double-click the Excel destination to open the editor. Click the New button next to OLE DB Connection Manager field. Type **C:\SSIS\RawFiles\Customers without CustomerID.xls** in the Excel File Path field in the Excel Connection Manager and click OK to return. The specified file will be created at run time.
11. Click the New button next to "Name of the Excel sheet" field and replace Excel Destination written after CREATE TABLE with **Customers without CustomerID** in the Create Table dialog box; then click OK twice. Select the Customers_without_CustomerID in this field. Go to the Mappings page to create mappings between the input and destination columns. Click OK to close the editor.

Exercise (Configure the Slowly Changing Dimension Transformation)

Finally, you will configure SCD transformation using the SCD Wizard in this part.

12. Drag the Slowly Changing Dimension transformation from the Toolbox onto the Data Flow surface just below the Customers Filter component. Connect Customers Filter by dragging the green arrow connector to the Slowly Changing Dimension transformation. Rename Slowly Changing Dimension to **Loading DimCustomer**.
13. Double-click the Loading DimCustomer transformation. This will start the SCD Wizard. Click Next to go to the Select A Dimension Table And Keys screen.

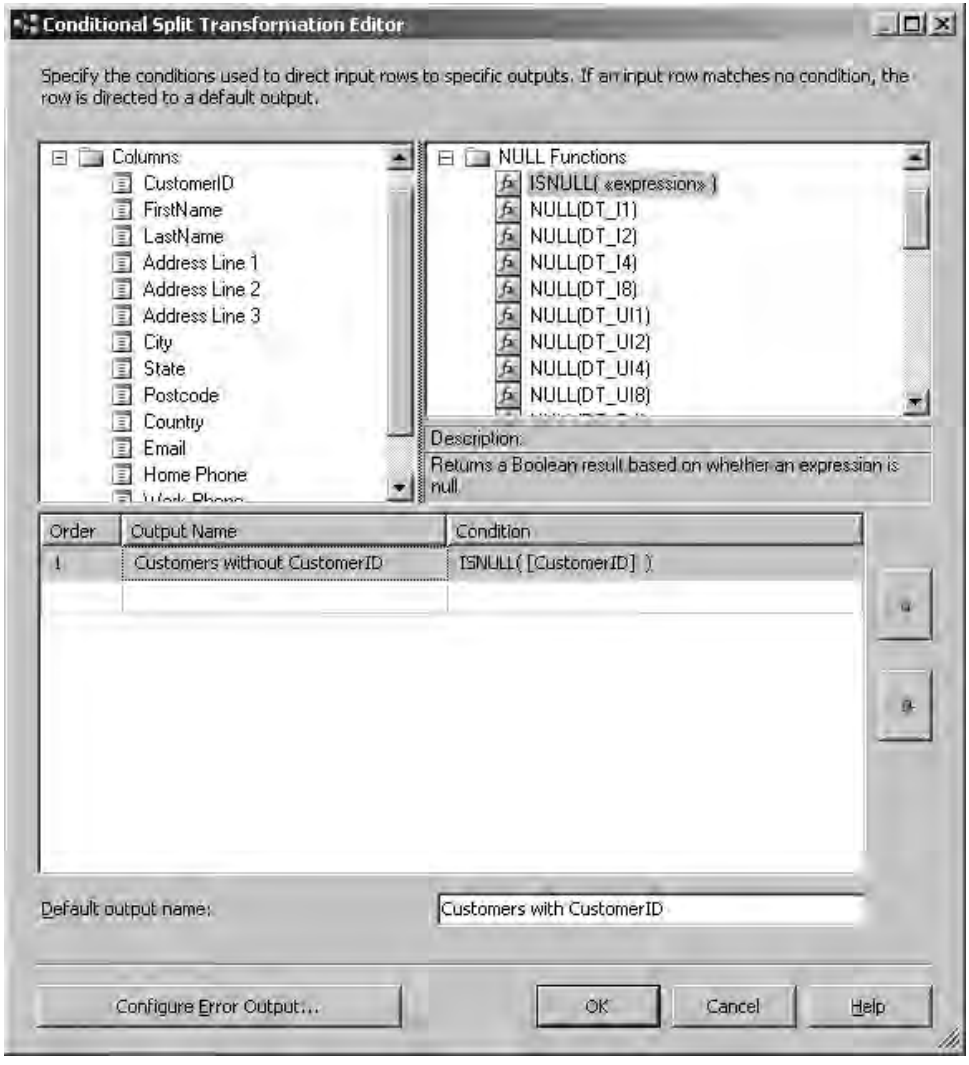


Figure 10-18 *Conditionally splitting customers on the basis of the CustomerID column*

14. Make sure that localhost.Campaign is selected in the Connection Manager field. Select [dbo].[DimCustomer] from the drop-down list in the Table Or View field. As you select the dimension table, you will see that the Input Columns are mapped to the Dimension Columns in the grid. Click in the Key Type column of the CustomerID field and select Business Key from the drop-down list, as shown in Figure 10-19. Click Next to move on.

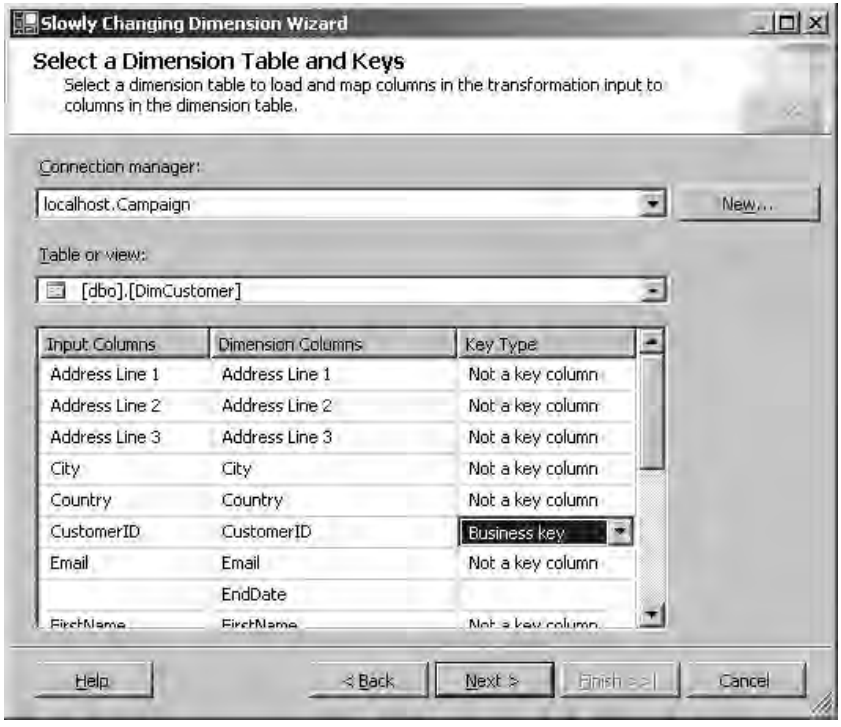


Figure 10-19 *Selecting Business Key in the SCD Wizard*

15. Specify the change types for various columns. As per the requirements set out in the initial statement of this exercise, you are to select all the address fields as the Historical attribute, e-mail as the Fixed attribute, and phone numbers as the Changing attribute. Click in the first row of the Dimension Columns and select Address Line 1 from the drop-down list; then click in the Change Type field to set it as a Historical attribute from the drop-down list. Set the different columns to different change types as per the following table.

Dimension Columns	Change Type
Address Line 1	Historical attribute
Address Line 2	Historical attribute
Address Line 3	Historical attribute
City	Historical attribute
State	Historical attribute

(*continued*)

Dimension Columns	Change Type
Country	Historical attribute
Email	Fixed attribute
Home Phone	Changing attribute
Work Phone	Changing attribute
Mobile Phone	Changing attribute

When you've configured all the columns, the wizard screen should look as shown in Figure 10-20. Click Next to move on.

16. In the Fixed and Changing Attribute Options screen, uncheck the "Fail the transformation if changes are detected in a fixed attribute" option, because we will make changes to the Email field to test how it goes. In a production environment, you need to consider both of the options in this page in light of your requirements. Click Next to open the Historical Attribute Options screen.

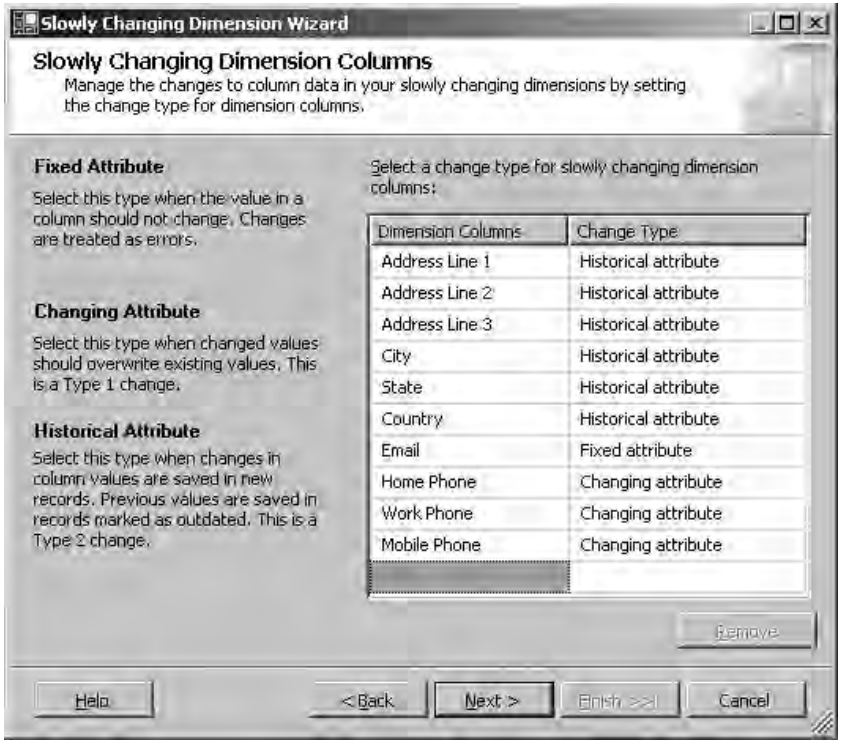


Figure 10-20 *Setting change types on the dimension columns*

17. You have two options to record a Historical attribute. If you select a single column option, you can then select a column and the value pair to indicate the Current/Expired or True/False values. Alternatively, you can select start and end dates to identify current and expired records. Select the radio button next to the "Use start and end dates to identify current and expired records" option. Select StartDate in the Start Date column and EndDate in the End Date column fields. Then select System::StartTime in the "Variable to set date values" field, as shown in Figure 10-21. This will update the values of StartDate and EndDate fields using the start time of the package provided by the specified system variable. Click Next.
18. By default, the inferred member support is enabled in the SCD transformation, and you can select from either of the two options to identify the inferred member record. In the first option, you can specify null as the column values for all columns with an inferred member change type, and in the second option, you specify an indicator column. Select the radio button for "Use a Boolean column to indicate whether the current record is an inferred member." Select IsInferred Column from the drop-down list in the Inferred Member Indicator field as shown in Figure 10-22. Click Next to review the outputs summary of the SCD Wizard.

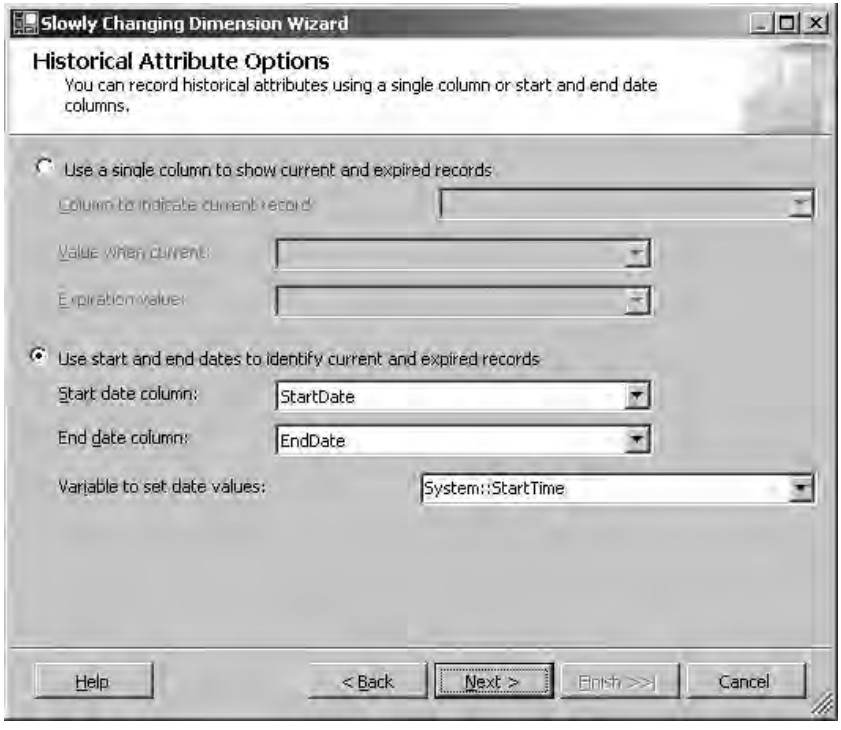


Figure 10-21 *Setting Historical attribute options in the SCD Wizard*

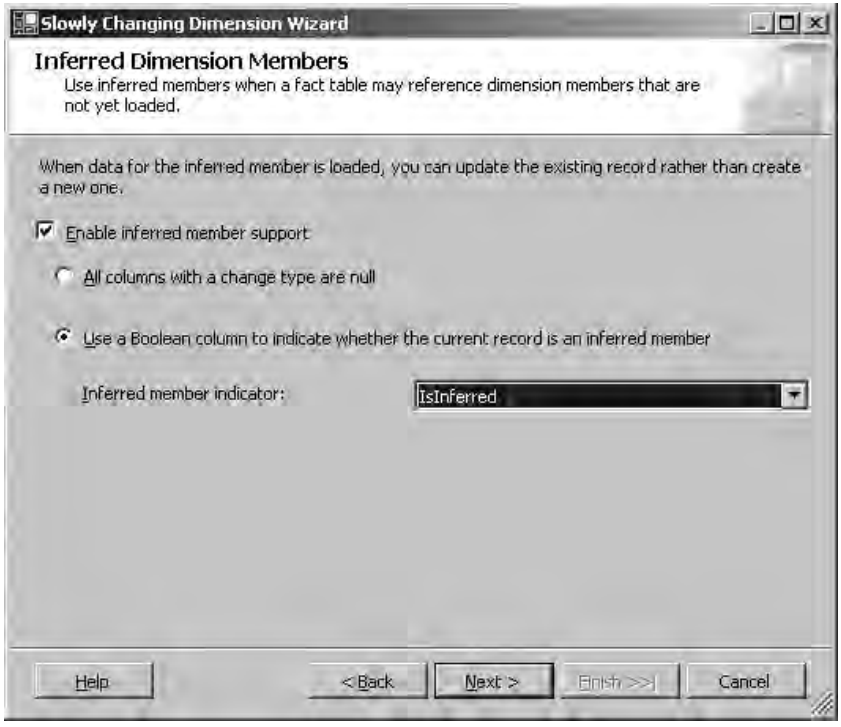


Figure 10-22 *Inferred member support in the SCD Wizard*

19. Click Finish after reviewing the summary information. The SCD Wizard will take a while to build the data flow for you, but when it's done, you will have a lot to review. After moving some of the components around, your data flow should look like the one shown in Figure 10-23.

The SCD Wizard has added a data flow to the four outputs out of six outputs available. Let's do a quick review.

- **Inferred Member Updates Output** Double-click the OLE DB Command transformation attached to this output and go to the Component Properties tab. Check the SQL Statement in the SqlCommand field to find that this statement will update the DimCustomer for the Inferred member record (IsInferred = 1) and also reset its status while updating (IsInferred = 0). Click Cancel.
- **Changing Attribute Updates Output** Double-click the OLE DB Command attached to this output and go to the Component Properties tab. Check the SQL Statement in the SqlCommand field to find that this statement will update the Changing attributes—i.e., the phone numbers for the member records. Click Cancel to exit.

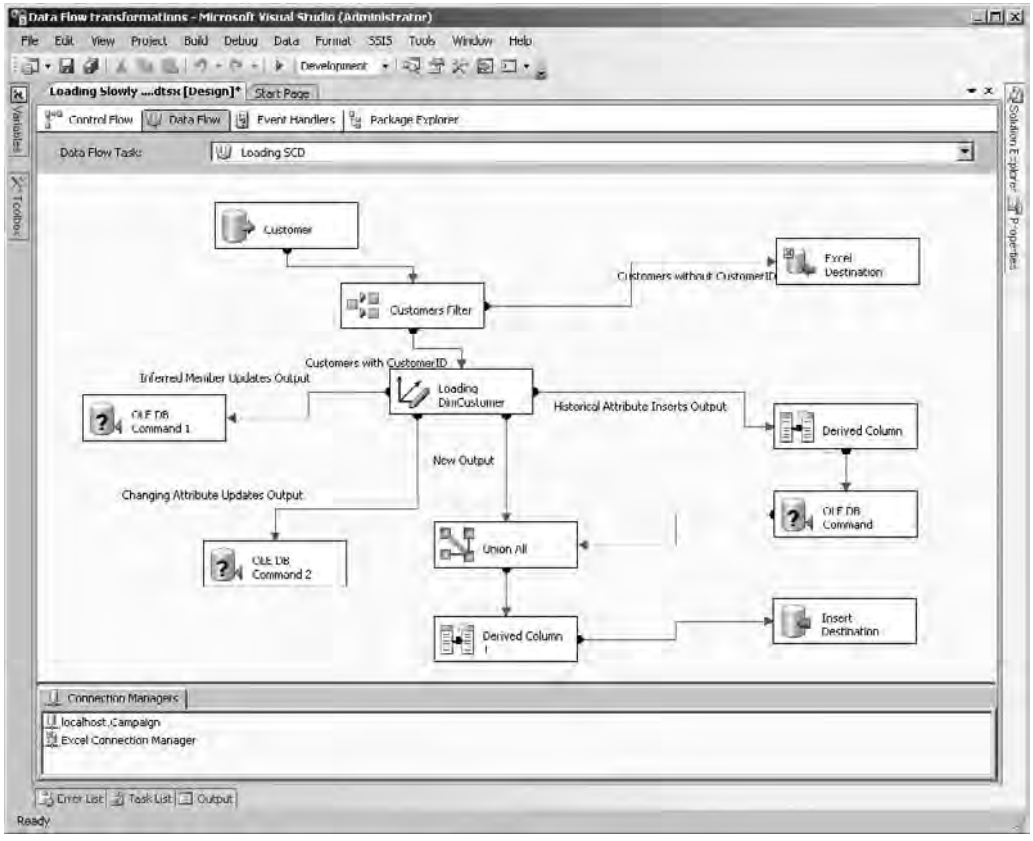


Figure 10-23 *Data flow created by the SCD Wizard*

- **Historical Attribute Inserts Output** Double-click the Derived Column component attached to this output and note that this transformation creates a new column EndDate and populates the package start time in this new column. Click Cancel. Double-click the OLE DB Command attached to the output of the Derived Column transformation and go to the Component Properties tab. Check the SQL Statement in the SqlCommand field to find that this statement will update the member record of the DimCustomer table with the EndDate derived in the previous component hence expiring the active record. Click Cancel. After the OLE DB Command, the output is then combined with the data flow of New Output using the Union All component. Double-click the Union All component to open the editor and note that the EndDate field created earlier has not been passed on to the New Output data flow. This will actually leave the EndDate column value null, indicating that the added record is the active member. So, in the Historical

Attribute Inserts Output data flow, the current member is first expired and then a new row is inserted for the same member with the new attributes as an active member.

- **New Output** This output contains new records that are combined with the "to be inserted" records from Historical Attribute Inserts output as stated in the preceding item. After that, the output is connected to Derived Column component. Double-click the Derived Column component and note that it adds a new StartDate column and populates it with a package start time using a system variable. Finally, it connects to an OLE DB Destination to insert these new records into the DimCustomer table.

Two other outputs, Fixed Attribute Output and Unchanged Output, are available at the Loading DimCustomer SCD transformation for which SCD Wizard has not created any data flow. However, if you want to capture the rows from those outputs, you can create a downstream data flow for them.

Exercise (Execute Loading Slowly Changing Dimension Package)

In this final part, you will execute the package to see how SCD deals with different types of updates.

20. Double-click the New Output Data Flow Path to open the Data Flow Path Editor. Go to Data Viewers page and then click Add. Click OK to add a grid-type data viewer in the Configure Data Viewer dialog box. Click OK again to close the editor. Similarly add data viewers to the other three outputs.
21. You are now ready to run this package. First, though, let's see the data that is going to be played and replayed to see the responses of SCD transformation to different changes. Open SQL Server Management Studio and connect to the database engine. Open a new query pane and run the following query to see records in the Customer table:

    ```
    Select * from [Campaign].[dbo].[Customer]
    ```

    Note that the Customer table contains 14 records and the CustomerID of the last four records is NULL. As per our package design, the Conditional Split transformation should filter out these records. Switch to BIDS, right-click the Loading Slowly Changing Dimension.dtsx package in the Solution Explorer, and choose Execute Package from the context menu.

    As the package executes, the data viewer that gets data first will pop up, which is the one on New Output in our case. If you move around on the screen, as shown

in Figure 10-24, you will Note that the Customers Filter component diverts four rows to the Excel destination and sends the other ten rows to Loading DimCustomer SCD transformation. Further, the SCD transformation identifies that all these records are in fact new records and diverts them to New Output. No record flows through any other output. Click the Detach button on the New Output Data Viewer to let the data flow to the destination and be inserted into DimCustomer table. Click the Detach button on other data viewers and close them. Press SHIFT-F5 to return to design mode.

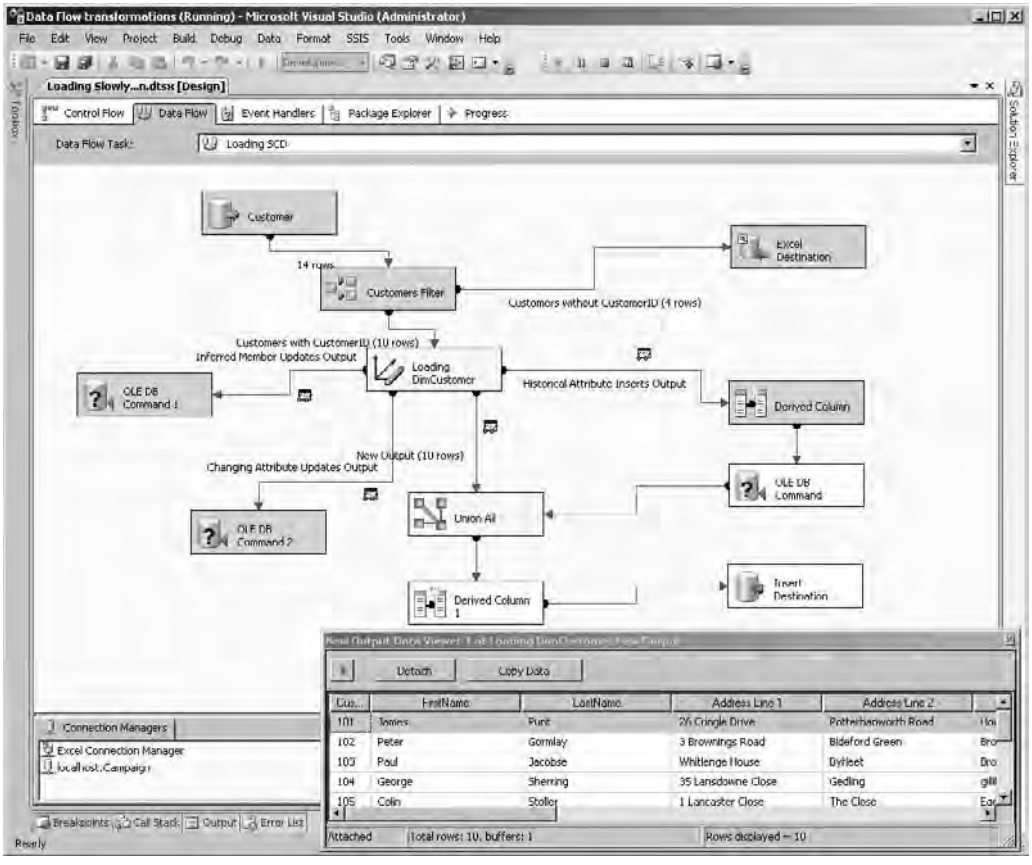


Figure 10-24 *Executing Loading Slowly Changing Dimension Package*

22. Let's make changes to Customer's data and see how this transformation works with them. Run the CustomerChanges.sql from the C:\SSIS\RawFiles folder in the query pane of SQL Server Management Studio. This script makes changes to the first three records in Customer table as follows:

CustomerID	Column Changed	New Value
101	[Home Phone]	020885711000
102	[Email]	peter.gormlay@AffordingIT.co.uk
103	[Address Line 1] [Address Line 2]	15 Abercrombie Avenue Wooburn Green

Then it adds an Inferred member record first in DimCustomer table with minimal information as shown here:

CustomerID	Columns	Value
111	[IsInferred] [StartDate]	1 getdate()

Then inserts additional information for inferred member in the Customer table, as shown next, to simulate a real-life scenario:

CustomerID	Columns	Value
111	[CustomerID]	111
	[FirstName]	Mark
	[LastName]	Morris
	[Address Line 1]	Flat 22, Crescent Flats The Ridgeway
	[Address Line 2]	Sketty
	[Address Line 3]	Chertsey
	[City]	West Midlands
	[State]	PE7 3RQ
	[Postcode]	United Kingdom mmark@AffordingIT.co.uk
	[Country]	
	[Email]	
	[Home Phone]	079576516756
	[Work Phone]	
	[Mobile Phone]	

23. After running the CustomerChanges.sql script successfully, switch to BIDS and execute the package again. As the package executes, this time three data viewers will pop up. First, you will see a record in the data viewer attached to Changing Attribute Updates output. Note that this is the record for which you changed the

home phone number, which is allowed, and no history is to be kept. Click Detach to let it process. Then look at the data viewer attached to the Historical Attribute Inserts Output will display a record with CustomerID = 103. Note that this is the record that changed the address for which a history is to be kept. This record will first update the existing record with EndDate and then will be combined with the New Output using Union All transformations so as to be inserted as a new record. Click Detach to let it process. Next look at the data viewer attached to Inferred Member Updates Output that shows a record with CustomerID = 111. This record will be updated in the DimCustomer table. Click Detach to let it process. The New Output path will also be processed and the package will complete.

24. Press SHIFT-F5 to switch back to design mode. Press CTRL-SHIFT-S to save all the files and close the project.
25. Switch to SQL Server Management Studio and check the DimCustomer table to see that it has been loaded with the expected values.

Review

This exercise has explained loading a dimension, which is a daily chore of a DBA's life. Loading a data warehouse dimension has been made easy with the Slowly Changing Dimension transformation, and configuring this transformation has been made easier still by the SCD Wizard. The SCD Wizard does so much work for you that you will probably use it every time you need to configure a SCD transformation. With the ease of use and coverage for most of the loading scenarios for a dimension, you may think perhaps SCD has nailed the problem; however, SCD has a shortcoming: its performance when it comes to loading into a large dimension. The use of OLE DB components to update each member in a dimension slows down the SCD to almost unacceptable levels of performance. You will need to do some work here to improve performance. One solution could be to create indexes on input and destination columns, while the other one could be to replace OLE DB Command with a staging OLE DB destination and an Execute SQL task to upload a data set virtually converting a row-based operation into a set-based operation. I've seen other customized solutions where developers have derived different flags, first using a script task and then loading the data set straightaway, converting the row-based decision-making logic at run time to a set-based operation and gaining performance. However, whichever method you use, testing for performance is the last, must-do step.

Data Mining Query Transformation

You can use a Data Mining Query transformation whenever you want to perform prediction queries against data mining models. This transformation has one input and one output and no error output. As you can envision, to execute your Data Mining

Extensions (DMX) query will require you to create a connection to the data mining model, so the user interface of this transformation provides two tabs—Mining Model and Query—for you to configure. In the Mining Model tab, you specify the Analysis Services Connection Manager to connect to an Analysis Services project or server. You select the mining structure from the drop-down list in the Mining Structure field. Once you've selected a mining structure, the Mining Models field lists the mining models associated with that mining structure.

In the Query tab, you can either type in your query or use the graphical query designer by clicking Build New Query. The graphical query designer helps you build DMX queries. If you type directly in the query mode and later on switch to design mode, you will get a prompt alerting you that your changes may get lost. The designer in the Query Builder allows you to select mining models and data flow input columns from two list boxes. You can drag and drop fields from these boxes onto the designer cells and build custom DMX queries for evaluating data flow input data against an existing mining model You can create more than one prediction query using multiple mining models. However, the mining models you select must belong to the same mining structure.

Term Lookup Transformation

Using the Term Lookup transformation, you can count the number of times a text term occurs in the input data row and create custom word lists and word frequency statistics. This transformation reads the terms from a lookup table to look for matches in an input column and then, by default, adds two columns named Term and Frequency to the output containing the term and the count for the term. The transformation supports one input and one output. The reference data can be in SQL Server 2000 and later or in an Access database, and this transformation can do a lookup only on a column that is either the DT_WSTR or the DT_NTEXT data type.

When you open the editor, you will see the following three tabs:

- **Reference Table** Specify an OLE DB Connection Manager and a reference table name from which the transformation can read lookup terms.
- **Term Lookup** Map an Input column to a Reference column to indicate that the lookup terms in the Reference column are to be counted in the Input column. You also select the columns that you want to pass through the data flow. Based on your selections, an advanced property—the InputColumnType of the Input column—is set that is available in Advanced Editor. Selecting a column only for pass-through sets the InputColumnType property to 0, mapping a column for lookup only sets the InputColumnType property to 1, and selecting a column for both pass-through and lookup operations sets the InputColumnType property to 2.

- **Advanced** Select to use a case-sensitive term lookup in which uppercase words are treated separate from lowercase words. However, if a word is a first word in a sentence and its first letter is capitalized, this can still match with a lowercase equivalent—so, for example, the word *travel* will match with the word *Travel* in the sentence "Travel in style and comfort."

It is relatively straightforward to configure this transformation. The mechanics are also simple. This transformation loads the terms it is to look up from the reference table in its private memory. It works in a fully precached mode only, so it loads all the values before processing lookups against the input column. However, to get accurate results you need to understand how this transformation behaves for different types of matches. The Term Lookup transformation extracts the required term from the input column by breaking the text into sentences, breaking sentences into words, and then normalizing the words. To extract the matching term, this transformation observes the following rules:

- If you specify the singular form of the word or phrase in the reference table, this transformation will match both singular and plural forms of the word or phrase.
- If you use a plural form of the noun or noun phrase in the reference table, the transformation matches it only with a plural form of the noun or noun phrase in the input data.
- If you want to do a match for nouns and noun phrases that contain special characters such as %, @, &, $, #, *, :, ;, ., , , !, ?, <, >, +, =, ^, ~, |, \, /, (,), [,], {, }, ", and ', you can do so by including these special characters in the nouns and noun phrases in the reference table.
- The Term Lookup transformation returns only one result for any lookup input column in which multiple overlapping terms are involved.
- While normalizing the input column words, the Term Lookup transformation will affect the last word in the lemmatized noun phrase for normalization.

Term Extraction Transformation

Using a Term Extraction transformation, you can extract terms from the text of an input column and can thus build a list of terms used repeatedly in the input column for text mining and data analysis. This transformation, however, is limited in that it can extract only nouns or noun phrases or a combination of both, in English text only. It is aware of linguistic information about English and comes with its own English dictionary. At run time, the Term Extraction transformation reads the specified input

column and uses its internal algorithms and statistical models in line with the options you've selected to generate the output results. The output of this transformation contains only two columns, Term and Score by default. The Term column contains the extracted term, while the Score column contains the number of times that term is found in the input column. To meet its objective of reading from an input column and writing to an output, this transformation supports one input and one output along with one error output. This transformation can extract terms from the input column that of either the DT_WSTR or DT_NTEXT data type.

Generally, you will use this transformation as the last transformation in that branch of the data flow, as it doesn't let the input columns pass through. However, it does provide an output that contains only the two resulting output columns. When you open Term Extraction Transformation Editor, you will see the following three tabs:

- **Term Extraction** In this tab, you can select an input column from the list of Available Input Columns from which you want this transformation to extract a term. You also can specify the names for the two output columns in this tab, which by default are Term and Score.
- **Exclusion** You can choose to use exclusion terms by clicking the Use Exclusion Terms check box, which tell the transformation to exclude some of the terms from extraction. While you are trying to build a meaningful list of terms that you can use for data mining purposes, you may want to exclude certain terms because they are appearing everywhere and causing you to lose focus from the key terms. You can specify the exclusion terms in a lookup table that must be in SQL Server 2000 or later editions or Microsoft Access. This lookup operation works in a fully precached mode, which means that the transformation loads the exclusion terms from the lookup table into its private memory before it starts extracting terms from input column.

  You specify an OLE DB connection manager to let it connect to the data source. Then you choose a table or view from the drop-down list, which this transformation can access using the OLE DB connection manager specified previously; finally you select the column from the drop-down list of columns that contains exclusion terms in the specified table or view.
- **Advanced** This tab has four sections. In the Term Type section, select one of the radio buttons to specify that the term is a Noun, a Noun phrase, or a combination of Noun and noun phrase. For this transformation, a noun is a single noun; a noun phrase is at least two words, of which one is a noun and the other one is a noun or an adjective. For example, *car* is a noun and *red car* is a noun phrase.

  In the Score Type section, you choose either Frequency or TFIDF (Term Frequency Inverse Document Frequency) by selecting either of the radio buttons. While the

Frequency represents the number of times the normalized term appears in the input, the TFIDF is a statistical technique used to evaluate the importance of a term in a document. This importance weight increases with the number of times the term or the word appears in the document, but is also offset by the commonality of the word in all the documents. This is an important measure used in text mining and is often used by search engines.

In the Parameters section, you specify values for the frequency threshold and the maximum length of term. Frequency Threshold is the minimum number of times a term must occur for it to be extracted and the Maximum Length Of Term is applicable for noun phrases only and specifies the maximum number of words in a noun phrase. For example, the noun phrase "top-of-the-line competition mountain bike" contains seven words.

You can select to use a case-sensitive match for extracting a term from the input column in the Options section. When you select this option, you tell the transformation to treat uppercase words different from lowercase words. In that case, *bicycle* and *Bicycle* will be treated as two separate terms. However, if *Bicycle* is the first letter in the sentence, it will still be treated as the same as *bicycle.*

As you can see, configuring this transformation for use in your package is not difficult; however, you need to work with it a few times to get the results you want to see, because this transformation behaves quite differently with different types of terms in the data, so you need to work out exactly what you are going to get for a given set of data. The term extraction process is based on its internal English language dictionary and statistical model that may not be 100 percent accurate against the data set it is working for; however, understanding the process that this transformation uses to extract terms from the input column will help you get going. Following are the steps this transformation uses in the process of term extraction process:

- **Tokenizing** The Term Extraction transformation identifies words by first breaking down the text into sentences and then separating the words from the sentences. To break the text into sentences, this transformation reads ASCII line break characters such as a carriage return (0x0d) or a line feed (0x0a). It is clever enough to recognize other characters as a sentence boundary, such as a hyphen (-) or an underscore (_), when neither the character to the left nor to the right of a hyphen or an underscore is a letter. It can recognize an acronym separated by one or more periods (.) and does not break it into multiple sentences. For example, it does not convert G.T.I. into multiple sentences. After separating the sentences, it breaks down the sentences further into separate words using spaces, tab characters, line breaks, and other word terminators but preserves the words that are connected by hyphens or underscores. This transformation takes care of

other special characters and is intelligent to extract the words properly, sometimes by separating the special characters and sometimes not. For example, (*bicycle*) is extracted as *bicycle,* whereas the term *you're* will generate only *you.* Refer to Books Online for more details on how this transformation handles tokenization.

- **Tagging** Depending upon your choice of Term type in the Advanced tab, the Term Extraction transformation will keep and tag only the words that match with your selection—i.e., if you've selected Noun Only, it will tag only the singular and plural nouns and reject all others; if you've selected Noun phrase only, it will tag only the terms that have two or more words containing at least one noun. After the words have been separated out by the Tokenizing process, they are tagged as either a singular noun, plural noun, singular or plural proper noun, adjective, comparative or superlative adjective, or number. All other words are discarded.
- **Stemming** After tagging the words, especially plural nouns that are not lemmatized, this transformation stems those words to their dictionary form by using its internal dictionary. For example, it converts *cars* to *car* and *lorries* to *lorry.*
- **Normalizing** Once the words have been separated and tagged, they are normalized so that the capitalized words and non-capitalized are treated alike. This process converts the capitalized letters in a word (for example, first letter of a word may be capital because it is the first word in the sentence) to lowercase—so, for example, *Cars* becomes *car.* However, note that the capitalized words that are not the first word in a sentence are not normalized and are marked as proper nouns, which are not included in its internal dictionary and hence not normalized.

Fuzzy Grouping Transformation

This transformation is part of the Enterprise Edition of SQL Server and is designed to help in the data cleaning process by grouping records that are likely to be duplicates and selecting a canonical record from the group to standardize the group. At run time, this transformation first groups together all the likely duplicate rows and then identifies a canonical row of data for each group. This identified canonical row and other likely duplicate rows are outputted after marking them with proper tokens in additional output columns. The duplicate rows are not deleted from the data; instead, they are outputted but marked so that you can identify and remove them from the data flow using the downstream components (such as Conditional split) if you want. This transformation uses a comparison algorithm to compare rows in the transformation input. You can customize this algorithm to be exhaustive if you want to compare every row in the input to the every other row in the input. Though this is quite an expensive method from the performance point of view, it can yield more accurate results. To compare rows

against each other in the input, this transformation creates temporary tables in an SQL Server database. To perform groupings of likely duplicate rows on the input data, this transformation supports one input and one output only.

When you open the editor of the Fuzzy Grouping transformation, you will see the following three tabs.

Connection Manager Tab

You can create a new connection manager in this tab by clicking New, or you can select already configured OLE DB Connection Manager. While specifying the connection manager here, you need to think seriously about the operations that this transformation will perform in the database using this connection. First of all, this transformation will create temporary tables and their indexes in the database, which requires that the user account you use in the connection manager setup must have necessary permissions to create tables and indexes in the database. Second, to compare rows against each other in the input data set, the algorithm that this transformation uses will create temporary tables much larger than the input data set. The sizes of the tables and indexes are proportional to the number of rows flowing through the transformation and the number of tokens you select to tokenize the data elements. This gets further aggravated if you choose to perform an exhaustive comparison. This may put quite stringent requirements of space on the database to which this transformation is connecting. You must ensure that the reference database has enough free space to perform a fuzzy comparison given the data set and your selections while configuring this transformation.

Some performance controls are provided in this transformation. If you go to Component Properties tab in the Advanced Editor, you will see four properties—Delimiters, Exhaustive, MaxMemoryUsage and MinSimilarity—under the Custom Properties section that help you fine-tune the balance between accuracy and performance. The Delimiters property lets you specify additional characters that you can use to separate strings into multiple words. Use only the required ones that generate the acceptable level of results. The Exhaustive property, discussed earlier, lets you specify whether you want to compare each input row with every other row in the input. If your data set is a few thousand rows long, you can use this option without any major impact. However, if you are dealing with data set that has millions of rows, consider using it only during the design and debugging phase or on a subset of data to fine-tune the similarity threshold requirements. You can also control memory requirements by using the MaxMemoryUsage property, for which you can specify a value in megabytes to limit its usage or specify 0 to enable dynamic memory usage based on requirements and available system memory. Finally, using MinSimilarity property, you can specify the minimum similarity threshold value between 0 and 1.

The closer the value to 1, the fewer rows will be selected as likely duplicates, which means less processing required by the transformation.

Columns Tab

In the Columns tab, you select the columns that you want to pass through as is and the columns you want to compare with other rows in the input. When you select the check box in front of a column, that column will be selected for comparison and a row will be added in the lower grid section of the editor window. In the grid, you can further specify the criteria by which you want this column to be compared against other columns. Using the Match Type field, you can either specify the column to be exactly matched or fuzzy-matched. For the columns for which you select Match Type as Exact, the Minimum Similarity is automatically set to 1, indicating that the column has to be matched 100 percent; however, for other columns that you set Match Type as Fuzzy, you can specify a Minimum Similarity value between 0 and 1. The column value closer to 1 indicates a closer match will happen. Using this, you can specify the values that match the rows that are approximately same. It requires more efforts for the transformation to identify duplicates with Minimum Similarity values closer to 1. While you specify minimum similarity values for each of the selected column in this tab, you can also specify minimum similarity thresholds at the component level in the Advanced tab. At run time, the Fuzzy Grouping transformation measures the similarity and groups together the rows on the basis of the similarity score. The rows that fall below the specified minimum similarity score are not grouped together. In real life, you may not know the minimum similarity score that works for your data. You can determine it by running the Fuzzy Grouping transformation several times using different minimum similarity threshold values against the subset or sample data that you can prepare using the Row Sampling or Percentage Sampling transformations. Once you find out the minimum similarity value, you can deploy it to production to group similar rows together.

So, in the output of the transformation, you will get all the input columns that you've selected to pass through or compare, the columns with standardized data (taken from canonical row), and a column containing the similarity score for each column that you select to participate in the Fuzzy grouping. The aliases of these columns are specified in the Similarity Output Alias fields in this tab. Finally, you can also use Comparison Flags, such as Ignore Case or Ignore Character Width, to specify how this transformation should handle the string data in the column while doing comparisons.

Advanced Tab

At run time, the transformation tokenizes each of the columns selected for comparison and then compares them against the columns of other rows. Based on the algorithm and

the settings, it then produces output, which is basically one output row for each input row with three additional columns that are specified in the Advanced tab. The first column—Input Key Column Name field—contains the _key_in value by default, which is the name of a new column it adds in the output. You can change this name if you want. This new column, _key_in, contains a string value that uniquely identifies each row. The second column—Output Key Column Name field—specifies the name assigned to a new column added in the output. By default, this name is _key_out, and it can be changed. This new column, _key_out, contains a string value that is same for all the rows that have been identified as likely duplicates and are grouped together. During run time, after having identified and grouped together the likely duplicate rows, it carries on to select a canonical row and copies its _key_in value in the _key_out column of all the rows in the group, making both values the same for the canonical row. This makes it possible for you to identify the rows in the group because they all have the same _key_out value, and the row that has _key_in value equal to _key_out value is the canonical row.

The third column added in the output is shown by the Similarity Score Column Name field, for which the default name is _score. This column holds values between 0 and 1, indicating the similarity of the input row to the canonical row. When the input row is selected as the canonical row, the value in the _score column is 1. For other rows in the group, the value of _score will vary, depending on how closely they match with the canonical row. The more the similarity, the closer the value of _score will be to 1. The exact duplicates to the canonical row will also be included in the output and will have a _score of 1.

You can specify a value for the Similarity Threshold attribute using the slider. You've used a similar attribute in the Columns tab called Minimum Similarity. That value is applied against each column, whereas the Similarity threshold is applied at the component level. The rows that have _score values smaller than the value you set for the Similarity threshold will not be considered as duplicates and hence will not be grouped together. As explained earlier, you may have to run the package containing a Fuzzy Grouping transformation several times to find the value that works with your data. Finally, you can select the token delimiters from Space, Tab, Carriage Return, and Line Feed by clicking the appropriate check boxes to tokenize data. You can also specify additional tokens in the Additional Delimiters field. This is the delimiters property in the Advanced Editor discussed earlier in this transformation.

Fuzzy Lookup Transformation

Earlier you have used the lookup transformation in a Hands-On exercise to look for exact matches of Postcodes in the database table and to add a City column in the output; and if there was no match for Postcode, the transformation extracted those rows into a flat file for your review. The Fuzzy Lookup transformation does more

for you than the lookup transformation, as it can do a fuzzy match and return one or more similar matches from the lookup table. The Lookup transformation's strength is that it can enhance data quality and standardize data by looking up matches from the reference table; however, this strength is limited by the fact that the lookup has to be an exact lookup—so, for example, Postcode has to match exactly. If you need to match First Name for *Stephen,* who may write his name as *Steve* also, you can't use an exact match. Your matching criterion has to pick up similarity in the text in a matching column to locate matching data, and this is precisely why the fuzzy lookup transformation was designed. This transformation enables you to correct, standardize, and enrich data by providing missing information using fuzzy matching technique. This transformation is available in the Enterprise Edition of SQL Server and requires a connection to an SQL Server 2005 or newer database to create temporary tables. As you would expect, the Fuzzy Lookup transformation has one input and one output to support its operation.

The Fuzzy Lookup transformation creates tokens of the data to be fuzzy matched and uses a lookup technique to fuzzy-match these tokens. To create tokenized data, this transformation needs a connection to an SQL Server where it can create a temporary table and index to store the tokenized information. As it will be matching tokenized data, there is a possibility that this transformation may return more than one match for a row. These matches carry different confidence levels for determining how close the match is. This transformation also takes into consideration the minimum similarity value before outputting that a row as a possible match.

The custom UI for a Fuzzy Lookup transformation is similar to that of the Lookup transformation and provides three tabs as described next.

Reference Table Tab

In the Reference Table tab, you specify the connection manager that this transformation uses to connect to the reference table and the match index options that this table will use to create, use, and maintain the index for fuzzy lookup matches. Once you specify a connection manager in the OLE DB Connection Manager field, you can then choose whether to use an existing index or create a new index.

The Generate New Index radio button is used to specify the creation of the new match index each time the Fuzzy Lookup transformation runs. When you select this option, you can then select the reference table from the drop-down list in the Reference Table Name field. At run time, the Fuzzy Lookup transformation connects to the reference table using the specified OLE DB Connection Manager and creates a copy of the reference table, adds an integer data type key to the copied reference table, and builds an index on the key column. Then, this transformation tokenizes the data in the columns that you want to reference and stores them in an index table called *match index.*

You can also select the Store New Index option, which allows you to save the match index for use in the subsequent processing of this transformation. On selection of this option, you can assign a name to the newly created index, which by default is FuzzyLookupMatchIndex. If you prefer to save the match index so that you can reuse it and avoid high processing costs at package run time, you may want to keep this index fresh and up to date all the times—i.e., you may want to update the match index whenever the reference table is updated with new records. For this, select the Maintain Stored Index check box. The transformation then creates triggers on the reference table to keep the match index table synchronized with the reference table. You may prefer to use the Maintain Stored Index option to keep the match index updated. However, before using this option, understand the effect of triggers on database performance and maintainability of the reference table. Refer to Microsoft SQL Server Books Online for more details on how to manage triggers when using this option with the Fuzzy Lookup transformation.

The process of creating a match index can be an expensive process, depending upon the size of the data you're dealing with. Thus this transformation provides a facility with which you can reuse an existing match index if the reference data is fairly static. When you select the Use Existing Index radio button, you can then choose a match index table from the drop-down list, which this transformation can use for repeated operations.

If you are dealing with millions of rows in the reference table, the recommended way to implement a Fuzzy Lookup transformation will be to generate and save the match index the first time by running the package containing this transformation and then reusing it using the Use Existing Index option in subsequent executions of the package.

Columns Tab

In the Available Input Columns you can select the Pass Through check boxes for the columns that you want to be passed through to the output as is. And in the Available Lookup Columns you can select the check boxes for the columns that you want to add in the output for the Fuzzy Lookup matching rows. You can create mappings in this tab between Available Input Columns and the Available Lookup Columns for which you want to perform lookup operations. The mappings you create here are visually different than those you create in the Lookup transformation because they are displayed as dotted lines in this transformation. This dotted line represents the fuzzy match. You can perform exact matches for some of the columns in the Fuzzy Lookup transformation as long as you keep at least one column using a fuzzy match. To change the match type fuzzy to exact, you have to open the Advanced Editor and change the JoinType property of the input column in the Input And Output Properties tab.

Advanced Tab

Here in the first option, you can set the "Maximum number of matches to output per lookup" by specifying an integer value. At run time, the transformation identifies matches considering similarity thresholds and can return the matches up to the number you have specified in this option. These matches may contain duplicates if you're looking for more than one output per lookup. Next, you can specify the Similarity Threshold value using the slider. This value can be a floating-point value from 0 to 1. When you specify a similarity threshold here, you apply it at the component level. You can also apply a similarity threshold at the column level—also known as the join level—using the MinSimilarity property of the input columns, which is accessible in the Input and Output Properties tab of the Advanced Editor. The closer its value is to 1 for a row or a column, the closer the row or column will be to match against the reference table and qualify as a duplicate. As mentioned, the output also contains a column for a confidence score. The combination of similarity score and confidence determines how close the input row or column is to the reference table column or row. The similarity score describes the closeness or the textual similarity between the input columns and the reference table columns, whereas the confidence describes the quality of this fuzzy match. Columns having a high similarity score and a high confidence score are the most likely candidates for duplicates; however, not all columns having a high similarity score will always have a high confidence score as well. You should understand a subtle difference between the two terms: for example, if you are looking for match on a series of cars, then the 3 series, 5 series, or 7 series returns a high similarity score, but the confidence score will be poor. Similarly, if you are looking for a PC and the only term used in reference table is *Personal Computer,* then the confidence for this will be high, whereas similarity, as you can see, is low.

At run time, the transformation creates or uses an existing match index to perform the fuzzy lookup and outputs the pass-through columns, plus the columns added from the lookup table, plus the additional columns carrying component-level similarity scores and confidence level information and the column-level similarity score column for the each column that participates in performing a fuzzy lookup.

Finally, you can select the token delimiters by clicking the check boxes provided for space, tab, carriage return, and line feed default delimiters. You can also specify Additional Delimiters in the provided field. Delimiters are the characters used to tokenize and separate fuzzy match fields into the words used for matching and scoring.

Other Considerations

Having configured all the options, you are ready to run the transformation. However, consider the following performance issues before you begin:

- As this transformation needs a connection to an SQL Server 2005 or later to create and maintain a match index table, connecting to a database server that has lots of free space is advisable. At index creation time, the reference table is locked by this transformation, so consider using another machine for the reference table if multiple users access this table. Also, it is a good idea to copy the reference table to a non-production server if the data changes regularly, especially during package execution, in which case results may be inconsistent.
- The Exhaustive property, which is available in the Custom Properties section of Component Properties tab in the Advanced Editor, is a Boolean field and can be set to True or False. This property yields more accurate results if set to True. However, setting the Exhaustive property to True should be done with care, because it will mean that each row in the input will be matched against every row in the reference table. Also, to perform this match, the entire reference table will be loaded in to the main memory, which will put high pressure on memory requirements. If your reference table is extremely large and you have little free memory available, avoid using this option. However, for a smaller reference table and with lots of free memory on the system, setting this option to True will yield better results.
- You can specify the maximum amount of memory in megabytes that this transformation is allowed using the MaxMemoryUsage option. Specifying a maximum amount of memory to match its requirements will greatly improve its performance. However, if enough free memory is not available on the system or you do not know how much memory will be required by this transformation, you can specify a value of 0, which indicates that the transformation will manage memory dynamically based on the requirements and the available free memory.
- You can manage memory on the basis of input rows as well. If you have many input rows to process, you can set WarmCaches to True to indicate that the match index and the reference table are to be loaded into memory. This can greatly enhance the performance by caching reference data and index in the main memory before the transformation starts processing input rows. Be aware that after tokenizing, only the tokenized tables are used and not the original reference data set.

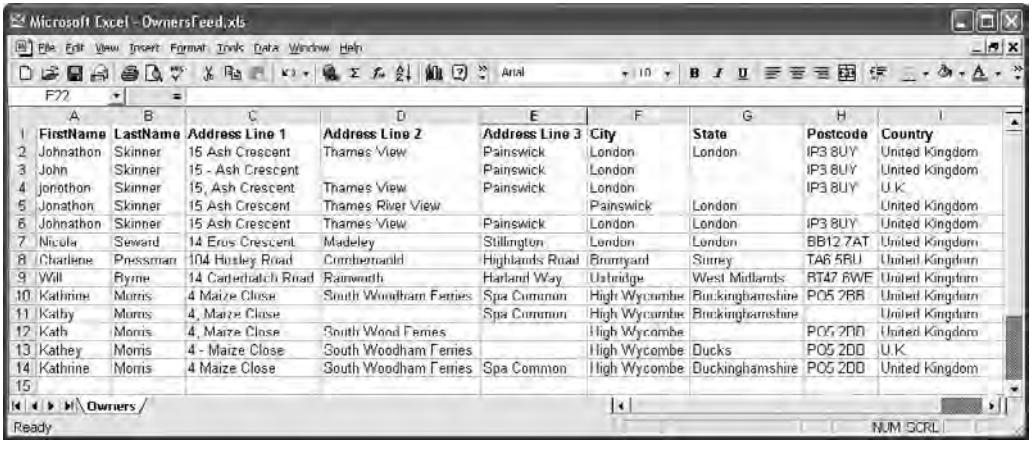
Hands-On: Removing Duplicates from Owners Data

The Fuzzy Lookup and Fuzzy Grouping transformations are the data flow components that you can use for data cleaning purposes. One of the main issues with data quality is to remove duplication in data, whether it occurs at loading time or it already exists in the data that you want to cleanse. Until now, you've worked on a couple of instances to remove duplicates in the earlier Hands-On exercises, but those instances were dealing with exact duplicates. In this exercise, you will deal with fuzzy duplication of data. You will work with these components to remove exact as well as fuzzy duplicates from the input data.

The scenario is that you are maintaining an Owner table that contains contact details for the owners of your products. You regularly receive an Owner's data feed that sometimes contains duplicate data. This duplicate data is not consistent, as users tend to provide their details differently at different occasions. You need to make sure no duplicate record is added to the Owner table.

Method

You have the Owner table in the Campaign database and receive OwnersFeed.xls files regularly that contain owner records. This Excel file can contain duplicate records for the same person. The complication, however, is that these records may not be exact duplicates, as persons provide their contact details differently at different occasions. Our sample OwnersFeed.xls file contains 13 records, of which 3 are unique records and the other 2 records have five variants each, with different name spellings and address details; one of these 2 records already exists in the Owner table. Open the OwnersFeed .xls file to have a look at the incoming data (see Figure 10-25).



	A	B	C	D	E	F	G	H	I
1	FirstName	LastName	Address Line 1	Address Line 2	Address Line 3	City	State	Postcode	Country
2	Johnathon	Skinner	15 Ash Crescent	Thames View	Painswick	London	London	IP3 8UY	United Kingdom
3	John	Skinner	15 - Ash Crescent		Painswick	London		IP3 8UY	United Kingdom
4	jonothon	Skinner	15, Ash Crescent	Thames View	Painswick	London		IP3 8UY	U.K.
5	Jonathon	Skinner	15 Ash Crescent	Thames River View		Painswick	London		United Kingdom
6	Johnathon	Skinner	15 Ash Crescent	Thames View	Painswick	London	London	IP3 8UY	United Kingdom
7	Nicola	Seward	14 Eros Crescent	Madeley	Stillington	London	London	BB12 7AT	United Kingdom
8	Charlene	Pressman	104 Hosley Road	Cumbernauld	Highlands Road	Bromyard	Surrey	TA6 5BU	United Kingdom
9	Will	Byrne	14 Carterhatch Road	Rainworth	Harland Way	Uxbridge	West Midlands	BT47 6WE	United Kingdom
10	Kathrine	Morris	4 Maize Close	South Woodham Ferries	Spa Common	High Wycombe	Buckinghamshire	PO5 2BB	United Kingdom
11	Kathy	Morris	4, Maize Close		Spa Common	High Wycombe	Buckinghamshire		United Kingdom
12	Kath	Morris	4, Maize Close	South Wood Ferries		High Wycombe		PO5 2DD	United Kingdom
13	Kathey	Morris	4 - Maize Close	South Woodham Ferries		High Wycombe	Bucks	PO5 2DD	U.K.
14	Kathrine	Morris	4 Maize Close	South Woodham Ferries	Spa Common	High Wycombe	Buckinghamshire	PO5 2DD	United Kingdom
15									

Figure 10-25 *Incoming data contains variants of duplicate records*

The owner with first name Johnathon already exists in the table and has five different variants of contact details. The other owner with first name Kathrine is a new owner, but has five different variants of contact details in the feed. You want to load only four records into the Owner table consisting of three unique records plus an owner record for Kathrine, while all the duplicate variants and the owner records for Johnathon should be removed. You will remove the exact duplicates from the OwnersFeed file using the Sort and Lookup transformation. After removing exact and matched duplicates, you will use the Fuzzy Lookup transformation to remove fuzzy matched duplicates. At this stage only the duplicate rows that exist in variant forms in the pipeline would be left. You will remove these duplicate variants using the Fuzzy Grouping transformation.

Exercise (Create Removing Duplicate Owners Package)

Start by creating a new package, adding the Data Flow task into this package, and then adding an Excel source to get the data in the data flow.

1. Open the Data Flow transformations project in BIDS. Add a New SSIS Package in the project and then rename it as **Removing Duplicates.dtsx**.
2. Drop a Data Flow task from the Toolbox onto the Control Flow surface and rename it **Removing Duplicates from OwnersFeed**.
3. Double-click the Removing Duplicates from OwnersFeed task to go to the Data Flow panel. Drop an Excel source from the Toolbox onto the Data Flow surface. Rename the Excel source **OwnersFeed**.
4. Double-click OwnersFeed to open the Excel Source Editor. Click the New button next to the OLE DB Connection Manager field. Type **C:\SSIS\RawFiles\OwnersFeed.xls** in the Excel File Path field. Leave the "First row has column names" option checked. Click OK to add an Excel Connection Manager and return to the Excel Source Editor. Leave Table Or View selected in the Data Access Mode field. Click in the Name of the Excel sheet field and select Owners$ from the drop-down list.
5. Go to the Columns page and verify that all the columns have been selected. Click OK to close the Excel Source Editor.
6. Take an opportunity to rename the Excel Connection Manager as **OwnersFeed Connection**.
7. Right-click anywhere on a blank surface in the Connection Managers area and choose New Connection from the context menu. Select Excel from the Connection Manager Type list and click Add. This will open the Excel Connection Manager dialog box. Type **C:\SSIS\RawFiles\DuplicateOwners.xls** in the Excel File Path field. Leave the First Row Has Column Names check box selected and click OK to add this connection manager. Once it is added, rename this connection manager **DuplicateOwners Connection**.

Exercise (Remove Exact Duplicates)

If you look at the data in the Figure 10-25, you will see that Johnathon Skinner and Kathrine Morris have duplicate records in the OwnersFeed. You will use a Sort transformation to remove these duplicates. Then you will check the OwnersFeed against the Owner table in the database to see whether any of the record exists in the Owner table. Johnathon Skinner also exists in the Owner table and will be removed from the input data flow using a Lookup transformation. The other variants of these owners will remain unaffected and will have to be dealt with separately.

8. Drop the Sort transformation onto the Data Flow surface and join OwnersFeed with it using the green path.
9. Double-click the Sort transformation to open the editor. Select all the Available Input Columns starting with FirstName, as shown in Figure 10-26. Select the check box for "Remove rows with duplicate sort values" and click OK to close the editor. Rename the Sort transformation **Removing Exact Duplicates by Sort Key**.
10. Add a Lookup transformation to the data flow and join the Sort transformation by dragging the green arrow onto it.
11. Double-click the Lookup transformation to open the Lookup Transformation Editor. Select "Redirect rows to no match output" option in the "Specify how to handle rows with no matching entries" field. Click the Connection page link in the pane.
12. Add the localhost.Campaign connection manager using the New button next to the OLE DB connection manager field. Then select [dbo].[owner] table from the drop-down list in the "Use a table or a view" field.
13. Go to the Columns tab, and map all the Available Input Columns to the Available Lookup Columns for the matching names as shown in Figure 10-27. Click to select the OwnerID column in the Available Lookup Columns to add it as a new column in the output. In this transformation, you are matching input records with the records in the Owner table, and for the matching record, you are adding OwnerID in the output to capture the ID of the record with which a match has been found. Click OK to close this editor. Rename the Lookup transformation **Removing Exact Duplicates by Lookup**.
14. Drop an Excel destination just below Removing Exact Duplicates By Lookup and join it to the Excel Destination. You will be asked to select an Output in the Input Output Selection dialog box. Select Lookup Match Output to connect to Excel Destination. Rename this Excel Destination **Exact Duplicates**.
15. Double-click the Excel Destination to configure it. Choose DuplicateOwners Connection in the OLE DB Connection Manager field if it's not already selected.

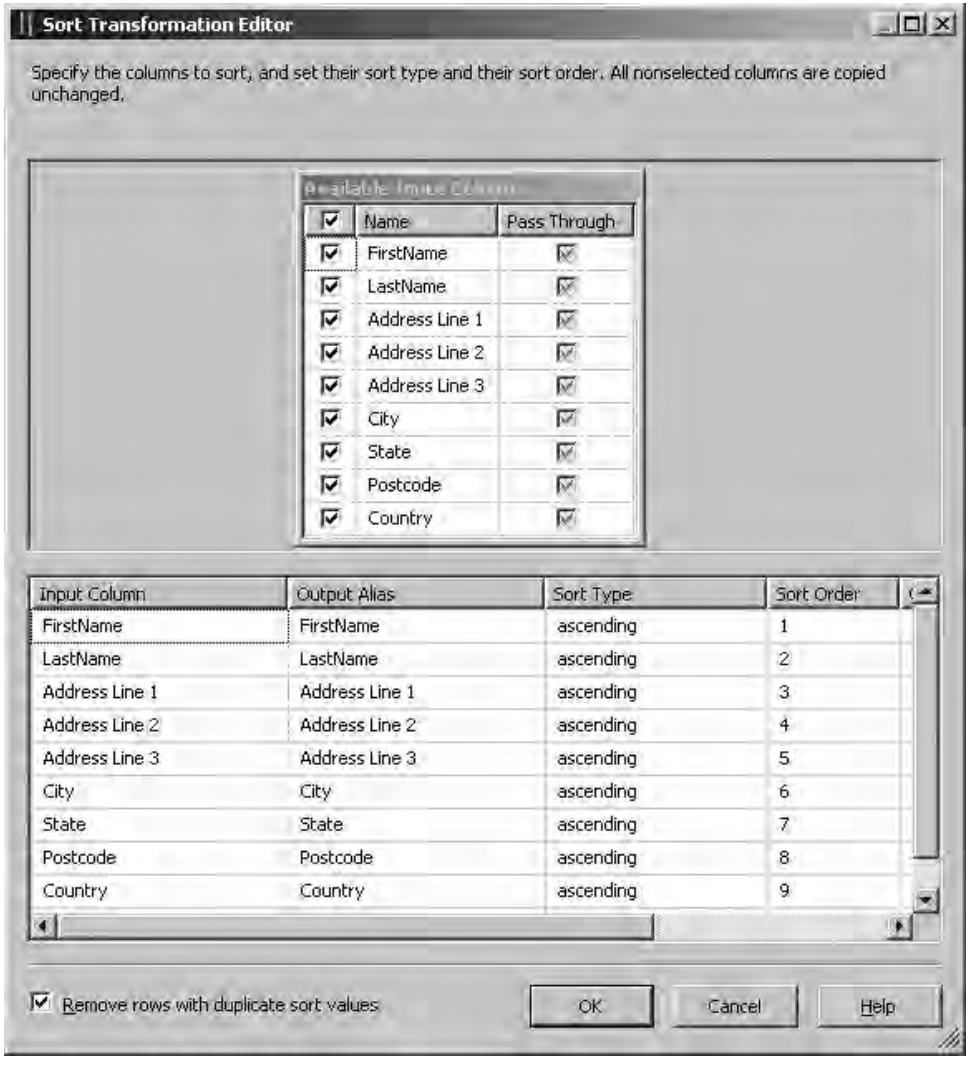


Figure 10-26 *Removing exact duplicates using a Sort transformation*

16. Click the New button next to the Name of the Excel sheet field and verify the CREATE TABLE statement that it is creating an Exact Duplicates table—i.e., a worksheet in Excel. Click OK to accept. Select Exact_Duplicates in this field. Go to the Mappings page to create the necessary column mappings that happen automatically. Click OK to close this editor.

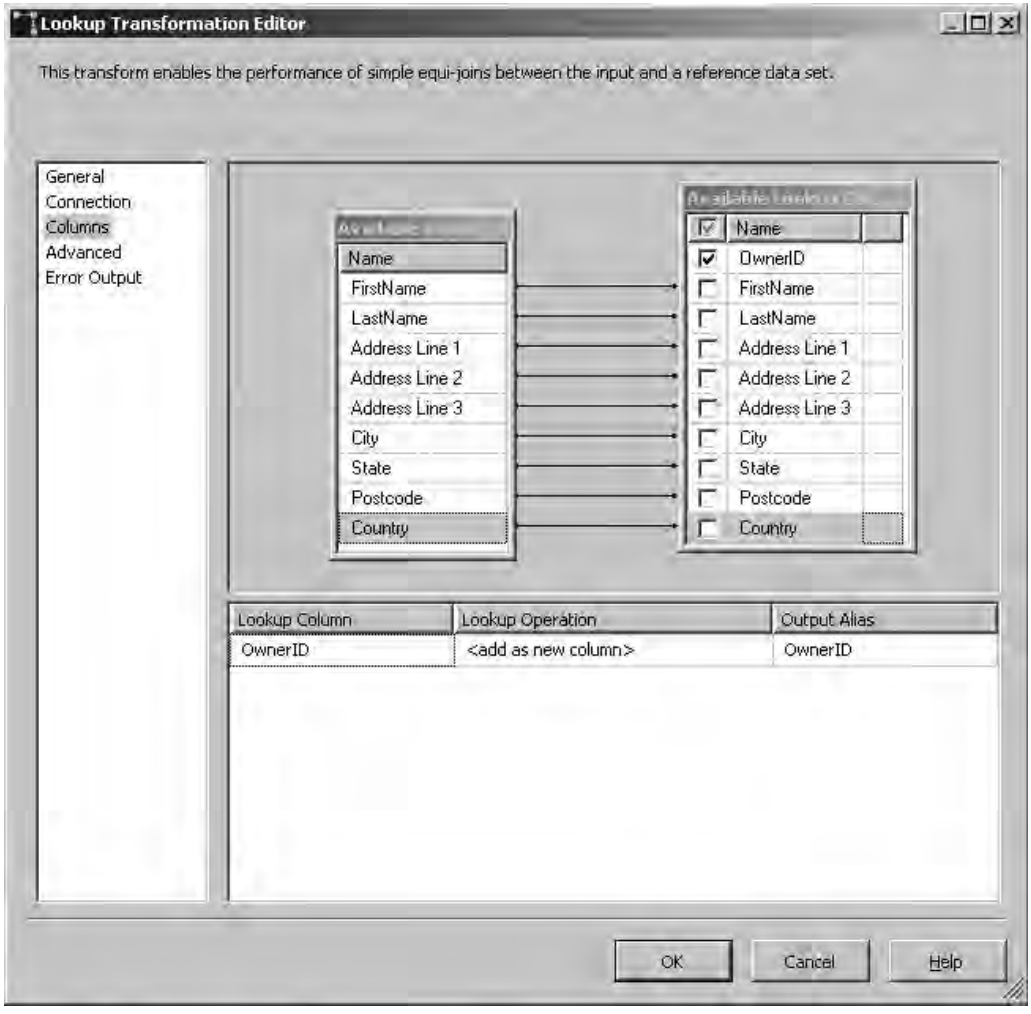


Figure 10-27 *Removing exact duplicates using the Lookup transformation*

Exercise (Remove Fuzzy Duplicates)

You've extracted exact duplicates from the Lookup transformation to an Excel destination. Now you will redirect the remaining rows to a Fuzzy Lookup and extract the duplicates using fuzzy match logic. You will be removing all the variants of Johnathon Skinner.

17. Drop a Fuzzy Lookup transformation onto the Data Flow surface to the right of the Removing Exact Duplicates by Lookup component. Click Removing Exact Duplicates by Lookup and drag and drop the second green arrow, which is

actually the Lookup No Match Output, onto the Fuzzy Lookup transformation. You have redirected all the remaining rows that did not match in the exact lookup operation to the Fuzzy Lookup transformation.

18. Double-click the Fuzzy Lookup transformation and make sure the Generate New Index radio button is selected. Select the [dbo].[Owner] table from the drop-down list in the Reference Table Name field. Go to the Columns tab.
19. Select the check box of the OwnerID column in the Available Lookup Columns to add it in the fuzzy matched records, as shown in Figure 10-28.
20. Go to the Advanced page and set the Maximum number of matches to output per lookup equal to 3. Slide the Similarity Threshold slider to 0.30. Click OK

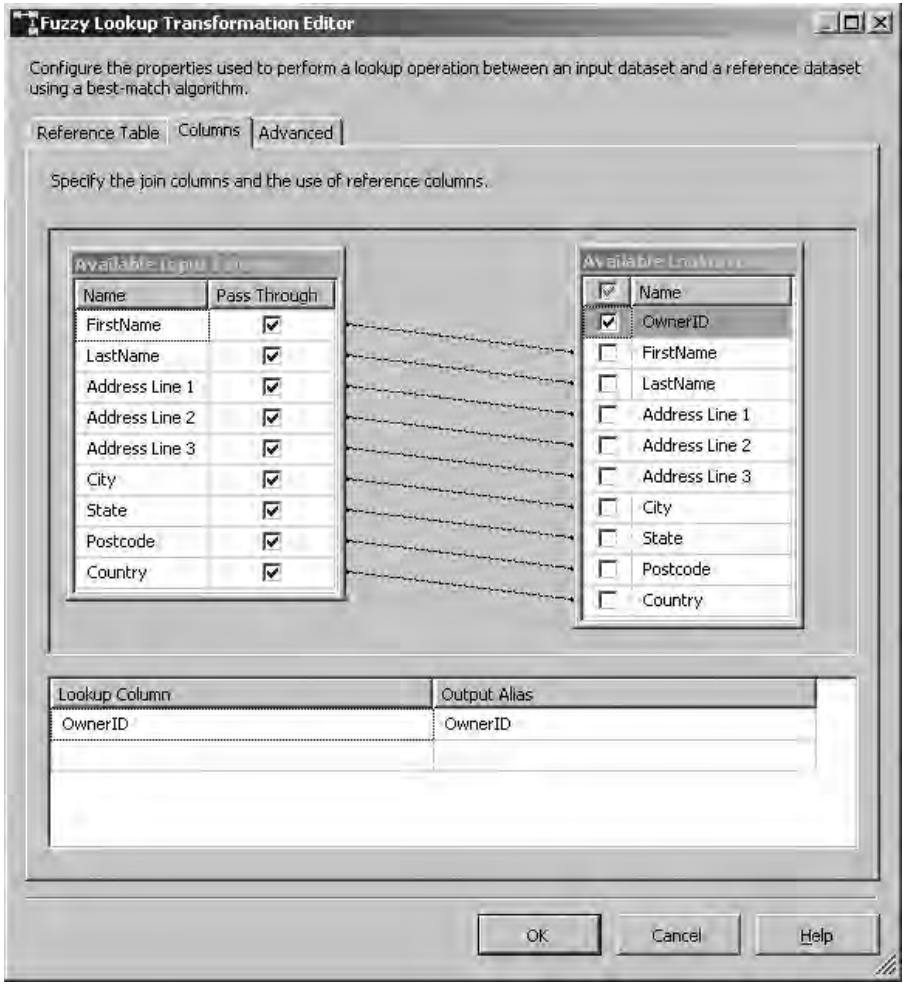


Figure 10-28 *Identifying fuzzy matched records with Fuzzy Lookup*

to close the Transformation Editor. Rename this transformation as **Identifying Duplicates by Fuzzy Lookup**.

21. Drop the Conditional Split transformation just below the Identifying Duplicates by Fuzzy Lookup transformation and connect both the components using a green data flow path. Double-click the Conditional Split to edit it. You need to identify the duplicates using _Similarity and _Confidence columns added by the Fuzzy Lookup transformation in the pipeline. You may have to run your package several times before you can get workable values for this identification. Here, I've already done the work for this data to identify these values; add the following in the Condition field:

    ```
    _Similarity >= 0.60 && _Confidence >= 0.50
    ```

    Type **Fuzzy Lookup Matches** in the Output Name field and **Remaining Owners** in the Default Output Name field, as shown in Figure 10-29. Click OK to close the editor. Rename this transformation **Splitting Fuzzy Matched Duplicates**.
22. Drop an Excel Destination below the Conditional Split transformation and join both of these components using a green arrow. Select Fuzzy Lookup Matches in the Output field of the Input Output Selection dialog box. Rename this Excel Destination **Fuzzy Matched Duplicates**.
23. Double-click the Excel Destination to configure it. Select DuplicateOwners Connection in the OLE DB Connection Manager field if it is not already selected.
24. Click the New button next to the Name of the Excel sheet field, verify in the CREATE TABLE statement that is creating the Fuzzy Matched Duplicates table—i.e., the worksheet in Excel—click OK to accept, and then select the Fuzzy_Matched_Duplicates in this field.
25. Go to the Mappings page to create necessary column mappings automatically. Note that in addition to the _Similarity and _Confidence columns, the Fuzzy Lookup transformation has added one _Similarity_ColumnName column for each column that participated in the fuzzy match. Click OK to close this editor.

Exercise (Remove Duplicates by Fuzzy Grouping)

Here you will remove duplicate variants of the Kathrine Morris. These records do not exist in the reference table but appear multiple times in the input file in variant forms. You will capture the Remaining Owners from Split transformation and use a fuzzy grouping to identify canonical row and the likely duplicates.

26. Drop Fuzzy Grouping in the pipeline to the right of the Splitting Fuzzy Duplicates component. Click Splitting Fuzzy Duplicates and then drag the available green arrow from the Splitting Fuzzy Duplicates and drop it on the Fuzzy Grouping transformation.

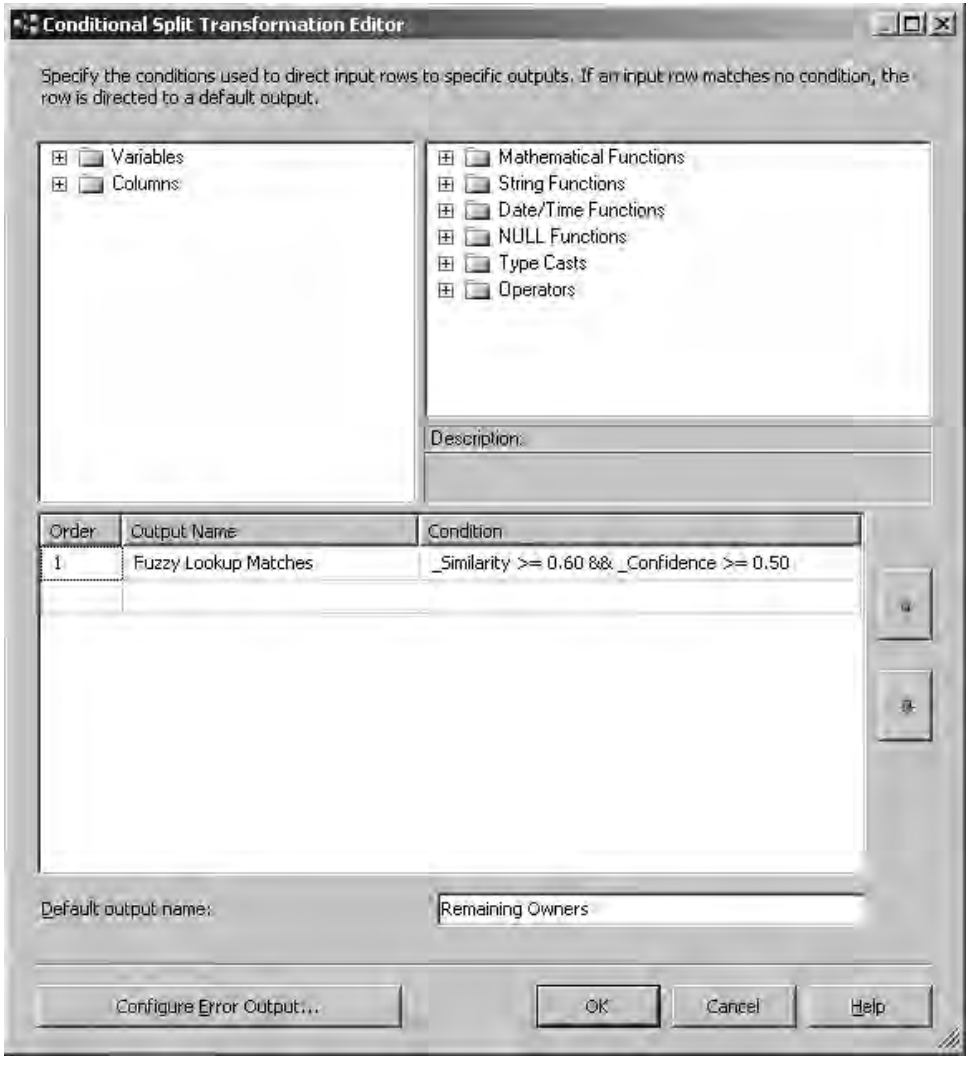


Figure 10-29 *Splitting fuzzy matched records from the Data Flow.*

27. Double-click the Fuzzy Grouping transformation and verify that the localhost. Campaign is selected in the OLE DB Connection Manager field. Go to the Columns tab. Note that all the Available Input Columns have Pass Through check boxes selected. Uncheck all of them. Select only the columns that are coming from OwnersFeed—i.e., from FirstName to Country, as shown in Figure 10-30. As you select these columns, a line corresponding to each selected column will be added in the grid. Type **0.25** in the Minimum Similarity column for the FirstName and LastName rows.

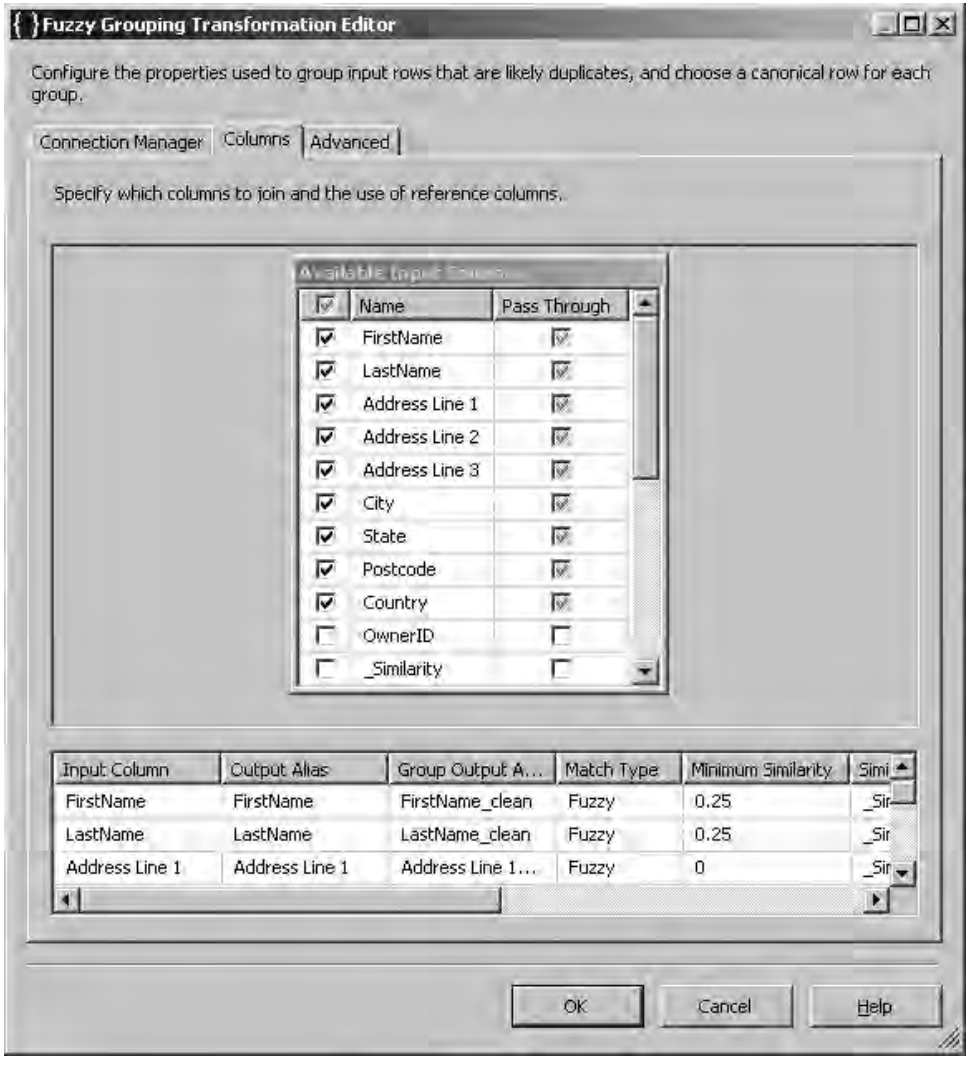


Figure 10-30 *Identifying and grouping duplicates using the Fuzzy Grouping transformation*

28. Go to the Advanced tab and set the Similarity threshold to 0.50 using the slider. Click OK to close this editor. Rename the Fuzzy Grouping transformation **Identifying Duplicates by Fuzzy Grouping**.
29. Drop the Conditional Split transformation just below the Identifying Duplicates by Fuzzy Grouping transformation and connect both the components using a green data flow path. Double-click the Conditional Split to edit it. You need to split the duplicates using _key_in and _key_out columns added by the Fuzzy

Grouping transformation in the pipeline. Add the two rows in the grid by adding the following in the Condition field:

```
_key_in == _key_out
_key_in != _key_out
```

Type **Canonical Row** in the first row and **Fuzzy Grouped Matches** in the second row of Output Name field, as shown in Figure 10-31. Click OK to close the editor. Rename this transformation **Splitting Fuzzy Grouped Duplicates**.

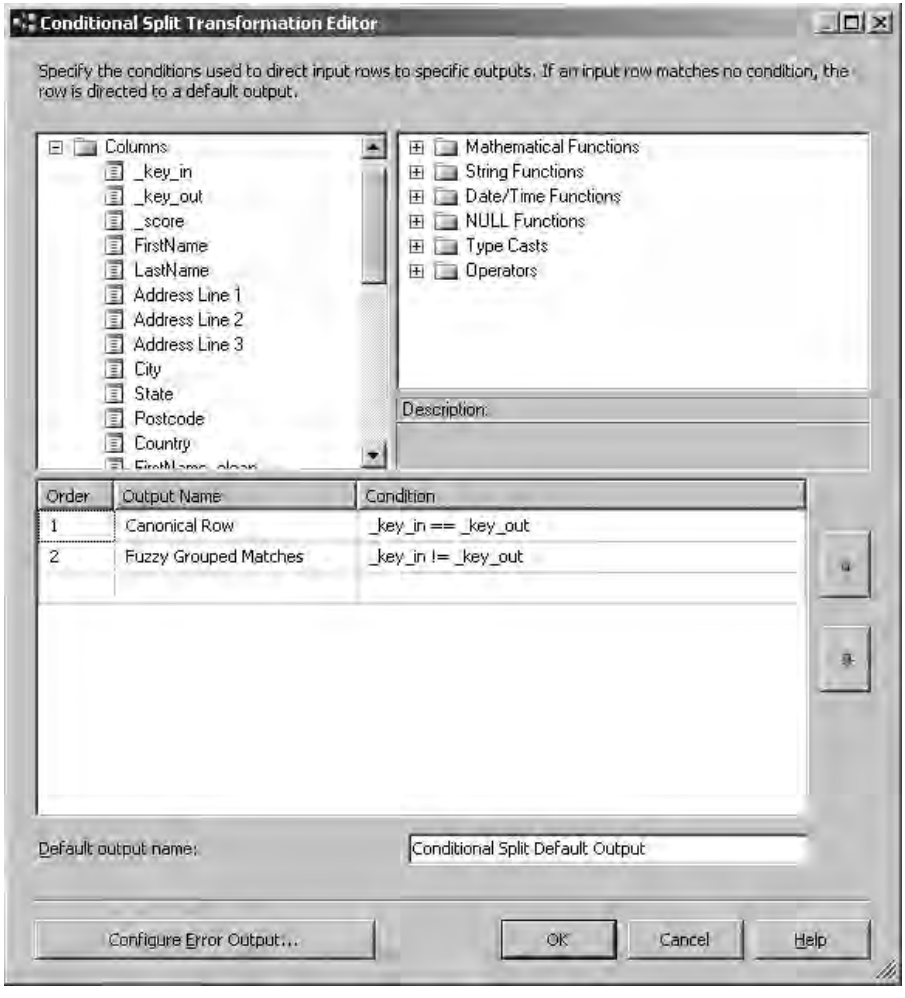


Figure 10-31 *Splitting fuzzy grouped and unique records in the data flow*

30. Drop an Excel Destination just below the Splitting Fuzzy Grouped Duplicates component and join both of these components using the green arrow. Select Fuzzy Grouped Matches in the Output field of the Input Output Selection dialog box and click OK. Rename this Excel destination **Fuzzy Grouped Duplicates**.
31. Double-click the Excel destination to configure it. Select DuplicateOwners Connection in the OLE DB Connection Manager field if it's not already selected.
32. Click the New button next to the "Name of the Excel sheet" field and verify in the CREATE TABLE statement that it is creating the Fuzzy Grouped Duplicates table—i.e., worksheet in Excel—then click OK to accept. Select Fuzzy_Grouped_ Duplicates in this field.
33. Go to the Mappings page to create the necessary column mappings automatically. Note that in addition to _key_in, _key_out, and _score columns, the Fuzzy Lookup transformation has added one _Similarity_ColumnName column for each column that participated in the fuzzy grouping. Click OK to close this editor.
34. Drop an OLE DB destination on the Data Flow surface below the Splitting Fuzzy Grouped Duplicates component and join the component to the OLE DB destination using the available green arrow. Select Canonical Row in the Output field of Input Output Selection dialog box and click OK.
35. Double-click the OLE DB destination and select [dbo].[Owner] table in the Name of the table or the view field. Go to the Mappings page and verify that the necessary mappings have been automatically created. Click OK to close this editor. Rename the OLE DB Destination **Owner**. Press CTRL-SHIFT-S to save all the files in the project.

Exercise (Execute Removing Duplicates Package)

Finally, execute the package to see how the various transformations remove the duplicate data.

36. You can add the data viewers after each transformation to see the records that have been removed from the pipeline. Later, I've explained how you can re-run the package to see the workings of various transformations over and over. As a first go, just run it without any data viewer.
37. Press F5 to execute the package. When the package completes execution, note the number of records after each transformation (see Figure 10-32).

Following is an explanation of the execution results:

- The Excel source brings 13 rows into the data flow.
- The Sort transformation removes two exact duplicate records—one for Johnathon Skinner and one for Kathrine Morris.

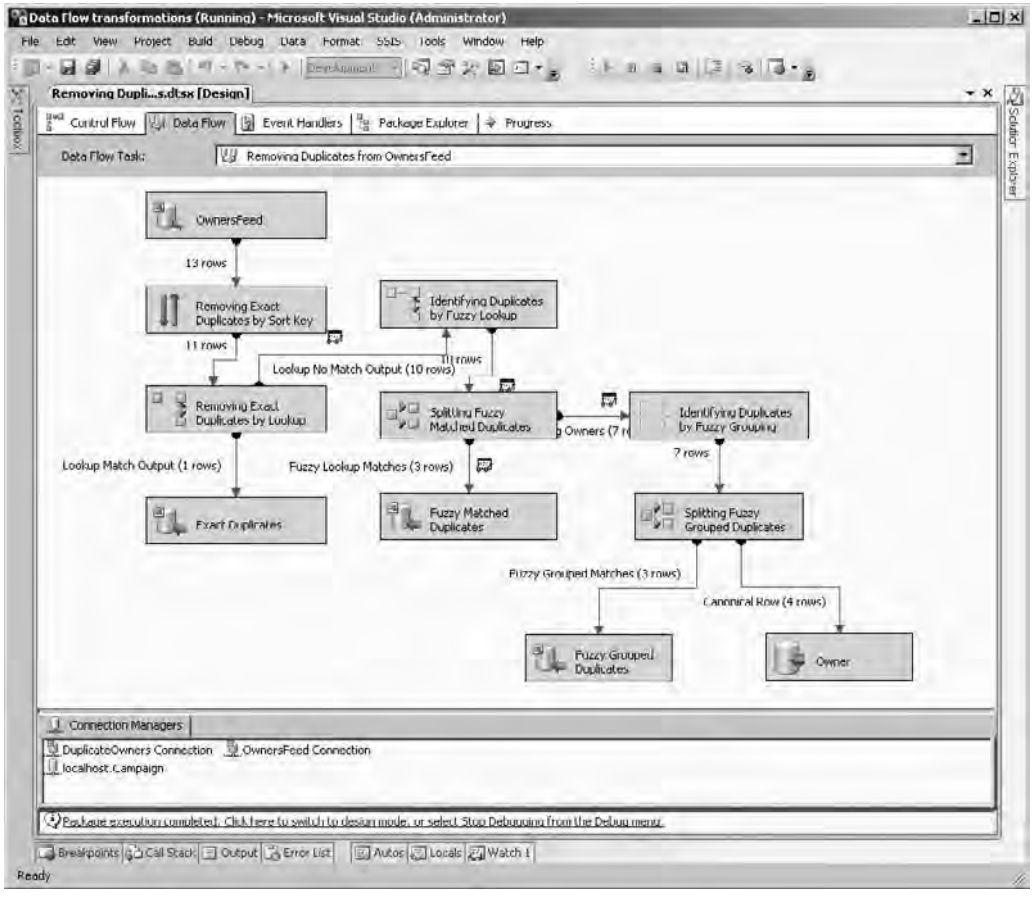


Figure 10-32 *Executing Removing Duplicates package*

- The Lookup transformation then matches Johnathon Skinner in the reference table and diverts it to the Excel destination; the remaining ten rows are then passed on to the Fuzzy Lookup transformation.
- The Fuzzy Lookup transformation fuzzy matches John, Jonothon, and Jonathon with the Johnathon in the reference table and marks all three records with a high similarity and confidence score. This high similarity and confidence score is then used by the Conditional Split transformation to filter out these three records to the Excel destination.

- The remaining seven records are then passed on to Fuzzy Grouping transformation, which looks for records that are likely duplicates in the data flow and groups them together using _key_in, _key_out, and _score column values. These values are then used by the conditional Split transformation to filter out records for Kathy, Kath, and Kathey that are fuzzy-grouped together. The remaining four unique records are then sent to the OLE DB destination for loading into the Owner table.

Review

You've seen various types of duplicates and the methods for removing them in this package. This package will give you a kick start into real-life de-duplication problems. However, bear in mind that whenever you use Fuzzy Lookup and Fuzzy Group transformations, you need to find the similarity threshold values by running the package on sample data that correctly represents the main data. The more effort you put into finding the value of the similarity threshold that works with your data, the more accurate your results will be.

If you want to re-run the package, you need to execute the following SQL statement against the Campaign database in the SQL Server Management Studio. This will refresh the Owner table for you to run the package with the same results.

```
USE Campaign;
DROP TABLE [dbo].[Owner];
SELECT * INTO [dbo].[Owner] FROM Owner_original;
```

Summary

Having used lots of Data Flow transformations, you must by now feel a lot more confident and ready for real-life challenges. You have used various transformations to perform functions such as pivoting; sorting; performing an exact lookup for de-duplication of data; and standardizing data, fuzzy lookup, and fuzzy groups to eliminate duplicates in a pipeline, aggregate data, and load a slowly changing dimension table. You've also studied several other preconfigured transformations that are straightforward to use in your packages. Having come so far, you can now create control flow and data flow in your packages to perform workflow and transformation functions, store and manage your SSIS packages, and secure them as well. In the next chapter, you will study how to deploy your packages in an enterprise environment.

Chapter 11

Programming Integration Services

In This Chapter

- The Two Engines of Integration Services
- Programming Options
- Extending Packages with Scripting
- The Legacy Scripting Task: ActiveX Script Task
- Script Task
- Script Component
- Script Task vs. Script Component
- Summary

By now you have worked with almost all the preconfigured tasks and components that Integration Services provides and can appreciate the power and ease it provides to developers for building enterprise-wide solutions. However, businesses are doing so many different things that it is sometimes not easy or even possible to build a solution for every scenario that can exist in an enterprise using the preconfigured tasks and components. SSIS does provide a way to cover even those complex scenarios. You can extend SSIS by writing your own custom code.

Not only can you extend SSIS using custom code, Microsoft has tried to make it easier to extend your custom solution and, hence, provides you various programming levels. SSIS provides a much enhanced object model that can be easily programmed with different options to choose from, based on the problem you're trying to solve. You can choose to extend your packages using scripting, you can develop custom components that can be deployed into SSIS and used as preconfigured components, or you can program your packages all over from scratch. In this chapter you will learn more about these options and when to choose the one that is more appropriate for your particular scenario. However, at this point I want to clarify that as programming Integration Services is a vast subject, it will be very difficult to cover it completely or even do some justice to it in just one chapter. This subject probably requires a complete book in itself. So, in this chapter scripting SSIS has been covered in detail, as I think it is the one area that will be of interest to most readers, and other options have been covered only up to an introduction level so that you can choose the best method to extend SSIS. Refer to the Books Online for other programming options.

The Two Engines of Integration Services

As you know, all of your packages contain Control Flow tasks and most of them also contain a Data Flow task, which is a special task and has its own components such as sources, destinations, and transformations. You also understand that the work flow and the management of the tasks are designed in Control Flow pane while the data movement and the transformations are designed in Data Flow pane. If you refer back to Figure 1-1, you can notice that top half of the figure—the object model of the Integration Services run time—includes connection managers, event handlers, and log providers, along with tasks and containers. This represents the Integration Services run-time engine and, as you can see, provides the necessary infrastructure for package execution and management support such as execution order, logging, event handling, connections, breakpoints, and transactions. The second engine of Integration Services, shown in the lower half of the Architecture diagram, manages the data movement and the transformations. The Data Flow task that performs the actual work of data movement and transformation runs under the management of the Data Flow

engine, also popularly called *the pipeline*. When you drop the first Data Flow task on the Control Flow Designer surface, you invoke the data flow engine. Though you'll have only one Control Flow within a package, you can include multiple Data Flow tasks in the package, with each Data Flow task able to support multiple data sources, transformations, and destinations.

These two engines of Integration Services provide complete control over the execution of a package and the flexibility to deal with buffer-oriented data movement and transformations in a very efficient way. You may also notice in the architecture diagram that both engines provide scope for building custom objects such as custom tasks, custom log providers, and custom connection managers for the run-time engine, and custom data flow components such as custom sources, custom transformations, and custom destinations for the pipeline. In fact, whenever you extend Integration Services programmatically, you will be working with these engines using different classes, methods, and properties exposed by the engines.

Some of the tasks or components provided in Integration Services are written in managed code, where the run-time engine and the data flow engine have been written in the native code for providing enhanced performance, but they are exposed for development through the managed object model of Integration Services and providing ease of extension. The run-time engine represents the Microsoft.SqlServer.Dts.Runtime namespace that contains the classes and interfaces to create packages, custom tasks, and other package control flow elements. And the data flow engine represents the Microsoft.SqlServer.Dts.Pipeline namespace that contains the classes used to develop managed data flow components.

Programming Options

The object model of Integration Services allows you to program any aspect of SSIS; for instance, you can extend the prebuilt functionality and manage interaction of SSIS with other applications by building interfaces or SSIS packages created programmatically by your custom-built application. This is possible because Integration Services fully supports the Microsoft .NET Framework and allows you to choose any of the .NET-compliant languages. The SSIS development team has done an excellent job by making it easier to program SSIS by extending the packages, yet more powerful by enabling the development of custom objects that can be developed outside the package, but can be included in the package just like prebuilt objects once deployed into the Integration Services object model. You can use Microsoft Visual Studio or any other development environment with your preferred .NET-compliant language to write custom code. So, you can choose from the options of simple one-off scripting or custom develop SSIS objects or in fact build complete packages depending upon your requirements and ability to program. Let's explore these options in detail.

Scripting

As mentioned earlier, when you have a need that can't be met using prebuilt tasks or components, you'll need to create the required functionality. Now, if your need is one off—i.e., the particular functionality is not going to be required in other packages and you are looking for the least development effort—scripting SSIS should be your choice. The code developed using scripting is generally reused within the development team working on the same project. The developers who have worked with SQL Server Data Transformation Services (versions 7.0 and 2000) might have used the scripting option already. The ActiveX Script task is the only method to extend Data Transformation Services (DTS), so many developers have used it extensively and some have actually built quite complex scripting solutions that were deployed throughout the enterprise. Later it has been realized that the cost of maintaining such solutions is quite high, as the ActiveX Script task was not designed to create enterprise solutions. Integration Services has overcome this limitation and includes many flexible scripting solutions along with the possibility of developing reusable custom objects that are easy to maintain. Though the ActiveX Script task has been provided in Integration Services, you should refrain from using it for new development work. This is provided only for backward compatibility, as an interim support while migrating your packages to Integration Services.

The two scripting objects, the Script task and the Script component, replace the Dts scripting functionality with a much better and more powerful programming environment, Microsoft Visual Studio Tools for Applications (VSTA). This embedded scripting environment allows you to choose Microsoft Visual Basic 2008 or Microsoft Visual C# 2008 as your preferred language for scripting. You can create a custom task for use in the Control Flow with the Script task or a custom component such as a source or a transformation, or else a destination for use in the Data Flow task with the Script component. When you use the VSTA environment to write scripts for either of these script objects, the scripting environment creates lots of infrastructure code for you and leaves you to focus on writing the code for the required functionality. This makes writing scripts much easier using VSTA. There are several other benefits to use of this powerful IDE, such as extensive debugging and testing of the written code. Due to .NET Framework support, you can use the .NET namespaces, take advantage of the class libraries, and also reference external .NET assemblies quite easily in your scripts. This is a very powerful feature that can save your many man-days of redevelopment effort for the assemblies that already exist. You can simply reference existing assemblies and use already-developed business rules or functionality within your package.

All this power and ease of use doesn't come free, though the cost of these benefits is very minimal in this case. The code that you write in the Script task or Script component resides in the package and is not available to other packages. If you want to

reuse the code, you will have to copy the script to other packages. To explain it further, when you deploy a package that has been developed using only the prebuilt objects to a different server, the code for prebuilt objects is not sent along with the code for your package. The prebuilt objects are available as a compiled binary library to the package within Integration Services, while the custom scripts obviously have not been published and hence are not available for use with other packages. You can copy the code quite easily to script objects in other packages if you need to; however, that will increase the maintenance cost. Think about the code that needs to be modified and has been used in hundreds of packages all over in the enterprise. It won't be a welcome thing to do. The facility to script yourself out of a requirement should be used carefully and should be used where you know that the requirement is unique and will not be used in many packages. If that's not the case, you'll be better off with the custom-built extensions that have been developed from scratch by deriving from the base classes provided by the Integration Services object model.

Developing Custom Objects from Scratch

If you do not want to use scripting to extend your packages because the custom code might be used in multiple packages and you don't want to undertake the hassle of fixing several packages later on, you can build custom extensions in the managed code from scratch. Using the managed object model of Integration Services, you can develop extensions such as control flow tasks, connection managers, log providers, enumerators, data flow sources, data flow transformations, and data flow destinations. To develop a custom object, you will inherit from the appropriate base class as provided for the functionality and build on that. For example, to develop a control flow task, you will inherit from the Task base class, and for a data flow component you will inherit from the PipelineComponent base class. The provision of a base class as a starting point makes it much easier to develop custom extensions.

Once the object development has been completed, you will then build and deploy the object assembly into the appropriate Integration Services and global assembly cache (GAC). The object then can be added into the Toolbox within Visual Studio and can be used as any other prebuilt object. You will need to deploy the custom extensions on all the servers wherever you want to use them. For example, suppose a developer builds a package using a custom component that has been installed on his computer and wants to share this package with another team member. This new team member cannot use the package until he or she installs the custom component on his or her computer, as the component will be referenced locally on the computer. The availability of the custom extension in the SSIS designer makes it very easy to reuse it. As mentioned earlier, the code for the prebuilt components does not get copied into the package; rather, the package references the objects only. This also applies to the custom-built extensions,

and you do not need to worry about the deployment of the custom objects within your packages. The custom-built extensions are not deployed with your packages; rather, they are handled separately and keep your package deployments simple. This means that if you need to make an enhancement or a change into a custom extension, you do not need to modify all your packages, but will need to make change only in the custom extension and the packages will automatically pick up this changed object at their next run.

Building Packages Programmatically

When you want to work with your packages programmatically, the object model allows you to create, configure, load, and execute packages. You can create packages dynamically and define the sources, transformations, and metadata of the selected columns and destinations. Just to explain, think of a CRM application that you may want to extend with ETL capabilities so that you can create a reporting data mart. This CRM application is configured with different metadata for different clients, so you can create SSIS packages programmatically reading metadata of the deployed application from your application interfaces and avoid configuring SSIS packages manually for each client. Such extended applications can save you and the customer a lot of time and effort. Considering your requirements, you can create a grand application by creating packages from scratch, including package objects, you can simplify your solution by loading a template package and configuring it for the relevant changes, or else you can simply load and run an existing package.

Extending Packages with Scripting

Now that you've an overview of programming options, let's explore them in detail and try some Hands-On exercises along the way as we proceed. I would like to clarify some of the concepts about my approach in this chapter before we get deep into the exercises. The focus will be to demonstrate on how you can implement your code in Integration Services rather than on how to write the code, and to keep things simple and within reason, only Visual Basic 2008 code will be listed.

The Legacy Scripting Task: ActiveX Script Task

If you have used DTS 2000, you might have used the ActiveX Script task to extend your Dts packages. This powerful task was provided in DTS 2000 and helped database developers to develop packages that otherwise wouldn't be possible. Many database developers and information analysts have exploited this task to customize data transformation; apply business logic in the Dts package; manage files and folders; dynamically set properties on tasks, connections, or global variables; and perform

complex computations on the data. To help smooth migration from DTS 2000 to SSIS, Microsoft provided the ActiveX Script task in SSIS to run those custom-build scripts until such time when the scripts can be upgraded to a more advanced scripting task, simply called the Script task in SSIS.

The ActiveX Script task provided in SSIS is quite different than the one provided in DTS 2000 in look and feel. The basic purpose of the ActiveX Script task in SSIS is to allow you to run existing scripts, not develop new scripts. In fact, this task will be removed from future releases of Integration Services. Better not to use this task to develop new scripts, and opt instead for use of the more advanced and efficient Script task for new development work.

Here are some of the benefits of using the Script task over the ActiveX Script task:

- The Script task uses a much powerful development environment—Visual Studio Tools for Applications—which provides an integrated development environment (IDE) rich in features such as IntelliSense, color-coded syntax highlighting, line-by-line debugging support, and online help.
- It is easier to develop scripts in the Script task using either Visual Basic 2008 or Visual C# 2008, both of which are fully capable of referring external .NET assemblies in addition to .NET Framework classes and libraries.
- All the scripts developed in the Script task (and in Script component) are precompiled and hence, yield enhanced performance due to fast execution at run time.

If you have to use this task to use an existing ActiveX script, follow these steps:

1. Drop the ActiveX Script task on the Designer surface and double-click it to configure it.
2. Specify a Name and a Description for the task in the General page.
3. In the Script page Language field drop-down list, choose a scripting language that was used to write the ActiveX script. The default choices available are the VB Script Language and the JScript Language, though the ActiveX Script task can support other scripting languages, depending on the scripting engines installed on the local computer.
4. The Script field provides a simple interface where you can paste or type in your ActiveX script. If you have an ActiveX script saved into a file, you can click Browse and select the file, and your script will be read in by the task and shown in the Script field. Click Save to save the contents of the Script field to a file, and click Parse to parse the script.
5. The EntryMethod specifies the name of the method that is called from the ActiveX Script task at run time.

Script Task

The Script task lets you write a piece of code to extend the package functionality and get the job done. The ability to use your own code is provided by the Script task in the control flow and by the Script component in the data flow. To provide an IDE, the Script task and the Script component use Visual Studio Tools for Applications (VSTA) in Integration Services 2008. The previous version, Integration Services 2005, used Visual Studio for Applications (VSA) as its IDE, so if you have some scripts that are written in SSIS 2005, you will need to upgrade them for the new environment. Refer to Chapter 14 for details on migration issues.

Using VSTA, you can write scripts with Microsoft Visual Basic 2008 or Microsoft Visual C# 2008. If your business rules are already coded in different .NET-compliant languages or you prefer to write code in a different language, you can compile it into a custom assembly and call it within the Script task, as it can call the external .NET assemblies quite easily. The Script task allows you to leverage the powerful .NET libraries also. The code written in VSTA is completely integrated with Integration Services—for example, the breakpoints in VSTA work seamlessly with breakpoints in Integration Services. Before you run a package containing a Script Task, you do need to make sure that the VSTA engine is installed on the computer.

So, whether you want to achieve extra functionality or use existing code, the Script task provides enough facilities to allow you to accomplish your goals.

Hands-On: Scripting the Handshake Functionality

In a classical data warehousing scenario, it is quite common to use control or handshake files to let the processes know whether the operations upstream or downstream have completed. In our test scenario, we have a mainframe process that copies the files into a folder and stamps the handshake file with different strings based on the current status of the process. As the files could be big, depending upon what has been extracted—i.e., daily, weekly, monthly, or yearly data—we do not want to start loading the file that is still being written by this upstream process. Before loading the data file, the mainframe process stamps the handshake file with the LOADING string, and on completion of data loading into the data file, it stamps the file with the OK string. When the SSIS package starts, we want to check the string in the handshake file first; if it is OK, the package should stamp the handshake file with the PROCESSING string and should start processing. And after completion of processing, it should stamp the file with the UPDATED string so that the mainframe process knows that the data of the previous day has been processed.

Method

As you can make out, it is not easy to implement the preceding requirements in SSIS using prebuilt components, but on the other hand, they can be implemented quite easily using the Script task. The logic required is shown in the Figure 11-1. In our scenario, we will use HandshakeFile.txt file as the control file that has been saved in the C:\SSIS\RawFiles folder and the RawDataTxt.csv file as the data file saved in the same folder. You've already imported this data file in Chapter 2. The logic of the branches shown in the figure will be implemented with the help of a variable HandshakeMessage that will be created in our package. In this exercise, we will focus more on the Script task; the rest of the items you should be able to implement yourself by now.

Exercise (Working with Script Task GUI)

We will add a Script task in the package to read the handshake file.

1. Create a new project in BIDS with the following details:

Template	Integration Services Project
Name	Programming SSIS
Location	C:\SSIS\Projects

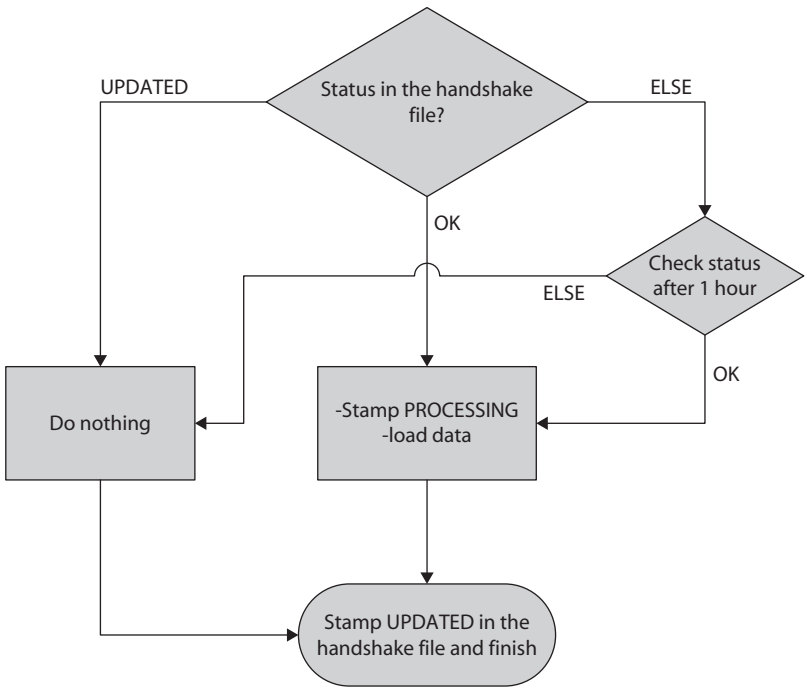


Figure 11-1 *Work flow for handshake exercise*

2. When the blank solution is created, rename the package **Extending SSIS with Script Task.dtsx** and click OK in the confirmation dialog box.
3. Create a variable called **HandshakeMessage** at the package scope.
4. Drop a Script task from the Toolbox onto the Control Flow pane and double-click to open the Script Task Editor.
5. Set the Name and the Description as follows in the General page:

Name	Determine workflow using Handshake
Description	This task sets HandshakeMessage variable that is used to determine the package control flow.

6. You can set the preferred programming language in the ScriptLanguage field. Change it to Microsoft Visual Basic 2008 as shown in Figure 11-2.

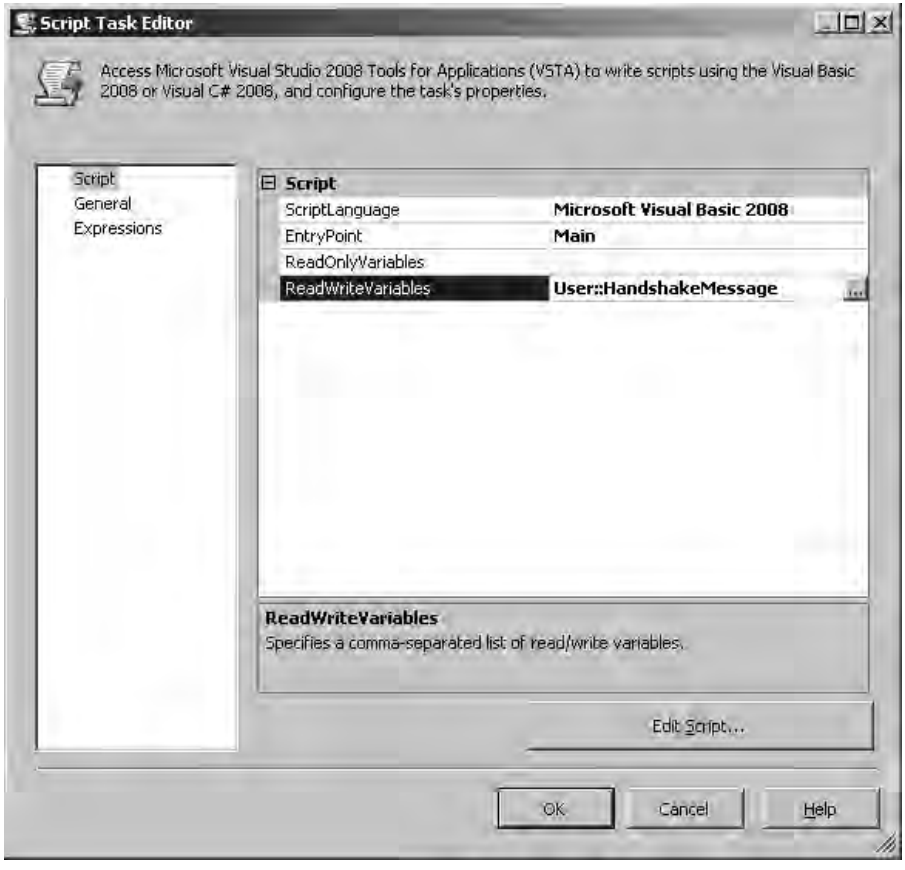


Figure 11-2 *Script Task Editor*

7. Entry point is the method that is called when the Script task runs. Specify the entry point name in the EntryPoint field. When you click Edit Script, the VSTA development environment is launched and a script project is created from the script templates based on the language you've specified in the ScriptLanguage field. This auto-generated script generates the ScriptMain class as the default class, which further contains a public subroutine or method called Main that acts as an entry point for the script. Make sure that the name of the public subroutine or method Main is the same as the value specified in the EntryPoint field at all times, in case you choose to change one later on.
8. The next two fields allow you to list variables that exist in the package and you want to access in your script. You can specify multiple variables in either of these fields by using a comma-separated list. As the names indicate, you can have either read-only access or read/write access to the variables, depending upon which field you choose to list them in. This is a cool method to exchange values between the package and your script. Though the script is a part of the package and gets saved within the package, yet it has its own object model: ScriptObjectModel, represented by the Dts object, that allows interaction with the package objects outside the Script task. We will study the Dts object in a bit more detail later on when we work on the script, but for the time being just remember that the Dts object makes the package objects such as variables accessible to you while working in the script. The variables listed in the ReadOnlyVariables and ReadWriteVariables fields are referred using the Variables property of the Dts object, and the Script task locks the variables for read or read/write access transparently. The code example in this case would be as follows:

```
Dim strScriptVariable As String = Dts.Variables("varPkgVariable").
Value.ToString
```

There is another way to access variables from within the script. You can use the VariableDispenser property of the Dts object within the scripts. In this method, you don't use ReadOnlyVariables and ReadWriteVariables fields in the Script task GUI; rather, you lock the variables for read or read/write access using your code. While using the VariableDispenser property is a standard way of accessing variables within scripts, it requires a bit more code to access a variable than the earlier method. Look at the following code example using the Dts.VariableDispenser method that shows much more code than the Dts.Variables method.

```
Dim vars as variables
Dts.VariableDispenser.LockOneForRead("varPkgVariable", vars)
strScriptVariable = vars("varPkgVariable").Value.ToString()
vars.Unlock()
```

You may prefer to use the earlier method, as that is more convenient, though the Dts.VariableDispenser method is the recommended method to use. First, in the earlier Dts.Variables method you specify variables in the ReadOnlyVariables

and ReadWriteVariables fields in the GUI; however, if the variables do not exist at design time and gets created only at the run time, you can't use this method. In such cases, you must use the Dts.VariableDispenser method, as this method locks the variables only when the code needs them and it doesn't need to know beforehand whether the variables existed or not. Second, as you can control release of locks in the second method and can release them more quickly for access to the variables by other concurrent processes, this can be the more efficient method to work with. The first method—i.e., Dts.Variables—does not release locks until the task has completed, and that obviously, in some cases, will be blocking other processes.

We will use the Dts.Variables method in our script to keep things simple, so let's specify the variable in Script Task GUI as a first step. Also, as we will need to update the variable value, so we will use the ReadWriteVariables field. To specify a variable, click in the ReadWriteVariables field and then on the ellipsis button and select the check box next to the User:HandshakeMessage variable. Click OK to come back and find the variable listed in the ReadWriteVariables field as shown in Figure 11-2.

Exercise (Understanding the Auto-Generated Script)

In this part of the exercise, you will open the scripting environment and understand the various parts of the auto-generated code.

9. Click Edit Script. This will invoke the VSTA-based Script task development environment. The Script task creates a blank scripting project within the VSTA environment (Figure 11-3) using the language you specified in the GUI. This VSTA project gets saved as a ScriptProject item inside the package XML code. Once this project gets created, you can't go back and change the language. Now if you close the VSTA environment, you will see that the ScriptLanguage field has been grayed out and you can't change the language.
10. Look in the Project Explorer. By default the project creates a ScriptMain item (ScriptMain.vb if you've selected the Microsoft Visual Basic 2008 language or ScriptMain.cs for Microsoft Visual C # 2008). This does not limit you in any way. You can create more items in your project such as classes, modules, and code files, and if you need to reference other managed assemblies in your code, you can also do that by adding a reference in your project. You can add a reference to an external assembly by right-clicking the project in the Project Explorer or from the Project menu bar item. Click the Project menu bar item to see what other items you can add in the scripting project. The scripting code that is auto-generated and the code that you add to create the required functionality get attached to the Script task in which it resides. All the items that you add in the scripting project get persisted inside the package, and you can organize them in the folders as you would normally do in other development projects.

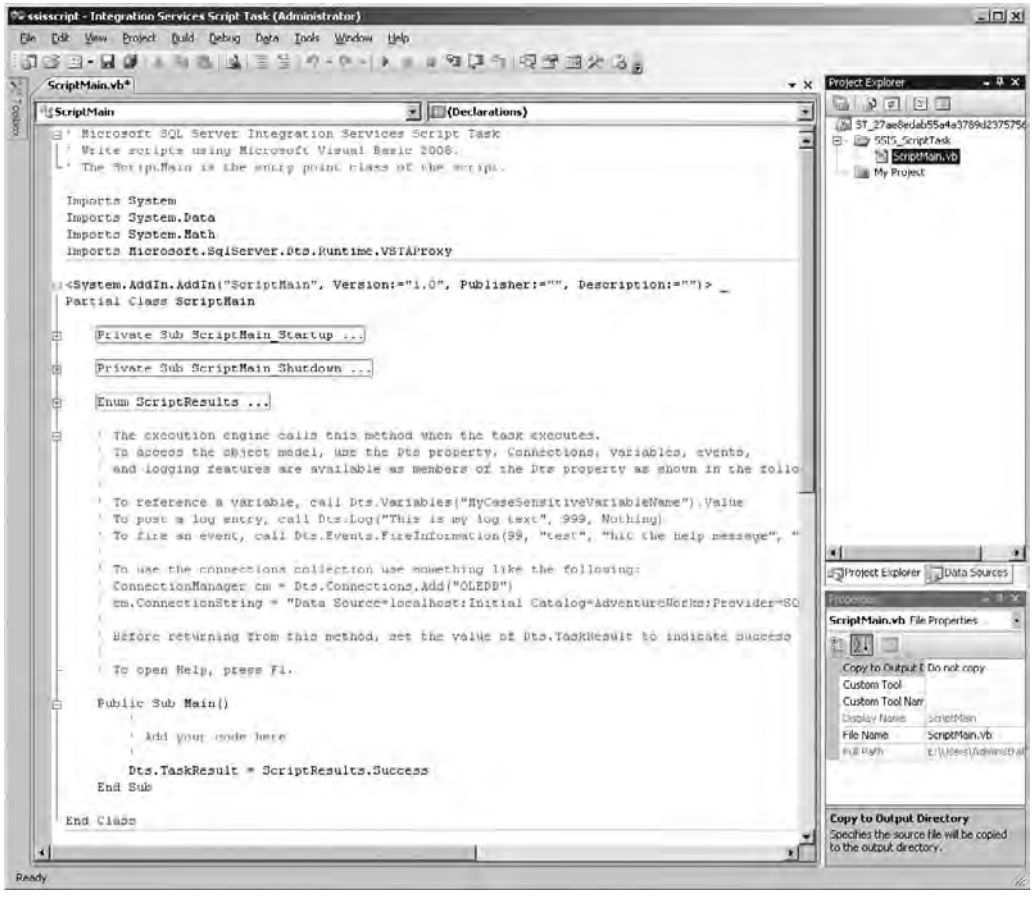


Figure 11-3 *Default Script task blank project*

Note that a lot of default code has been auto-generated for you. It starts with the comment about what the code is about and tells you that the ScriptMain is the entry point class of the script. It will be a good practice to add comments at various levels within your script. Replace the first line of the comment—Microsoft SQL Server Integration Services Script Task—with the project heading—**Script for Handshake** functionality.

After the comment lines, Imports statements have been added. These statements represent the .NET Framework system libraries and make it easier to call the functions in those libraries. Here you can add more system libraries as required in

your code. Listed next are some of the more frequently used .NET Framework classes.

- **System.Data** Used to work with the ADO.NET architecture
- **System.IO** Used to work with the file system and streams
- **System.Windows.Forms** Used for form creation
- **System.Text.RegularExpressions** Used to work with regular expressions
- **System.Environment** Used to retrieve information about the local computer, the current user, and environmental settings
- **System.Net** Used to provide network communications
- **System.DirectoryServices** Used to work with Active Directory
- **System.Threading** Used to write multithreaded programs

The code contains a class that is named as ScriptMain by default. If you scroll to the end of the code, you can notice that a public subroutine is also created called Main. So, at run time, Integration Services calls the ScriptMain.Main subroutine to execute the Script task. Two more subroutines have been added in the ScriptMain class—ScriptMain_Startup and ScriptMain_Shutdown—along with an enumerator ScriptResults that is used to enumerate the execution results. The ScriptMain_Startup subroutine doesn't contain much code, while the ScriptMain_Shutdown subroutine unlocks the variables that have been locked by the Dts.Variables collection. You can add more code to these subroutines in case you want to perform some logic at the startup or shutdown of the Script task.

Finally, note that the subroutine Main is the place where you would add your code as indicated by a comment line in the auto-generated code. The only line contained by the Main subroutine sets the TaskResult. It is important to explicitly set the execution result, as in some instances you may want to force the failure based on an outcome in the script if you're building a slightly more complex work flow within a package. The Script task uses the TaskResult property to return status information to the run-time engine, which in turn can use the return status to determine the path of the workflow. The Dts.TaskResult gets the result value from the ScriptResults Enumerator that has also been added in the script.

Exercise (Adding Your Code in the Script Project)

We have used the Dts object quite a few times in the previous sections, but it is time to learn about it a bit more before we add code into the script. Integration Services uses a class—Microsoft.SqlServer.Dts.Tasks.ScriptTask.ScriptObjectModel—called the ScriptObjectModel class to access the package objects within the code written for a Script task. Developers use properties and methods of the ScriptObjectModel class

to access the objects such as connections, variables, and events defined elsewhere in the package. The global Dts object is nothing but an instance of the ScriptObjectModel class and hence is used to interact with the package and the Integration Services run-time engine. Just be clear that the Dts object is available only in the Script task and not in the Script component, as it is inherited from the Script task's namespace. It has seven properties and one method as shown in Figure 11-4 and explained next.

- **Connections** Accesses connection managers defined in the package. The connection manager accessed in this way stores the information such as user name and password to connect to the data source, and you don't have to provide these details in your script. They can also be utilized to access data directly in the script to perform any data-related operation.
- **Events** Lets the Script task fire errors, warnings, and informational messages.
- **ExecutionValue** Returns additional information such as a value from your Script task to the run-time engine. This value in turn can be used to determine the control flow path.
- **Log** This is the only method of Dts object that helps to log information to any of the enabled log providers.
- **TaskResult** Returns the success or failure of the Script task to the run-time engine and determines the control flow path within your package. As explained earlier, setting the TaskResult property is the main way to control the package workflow, as you can force the value based on the outcome in the script and not just on the execution status of the script.
- **Transaction** Provides the transaction within which the task's container is running.
- **Variables** Provides access to the variables listed in the ReadOnlyVariables and ReadWriteVariables task properties for use within the script.

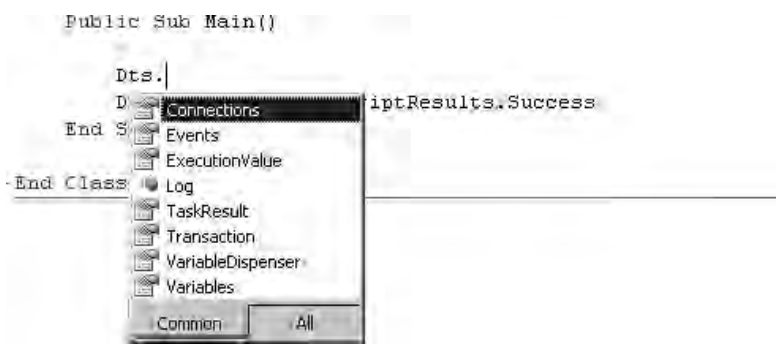


Figure 11-4 *The properties and method of the Dts object*

- **VariableDispenser** Provides an alternative way to access package variables within the script. Both the Variables and the VariableDispenser properties have been discussed in detail earlier in the exercise.

Now that you've learned all you need to write a script that can interact with the package, let's create a logic that is needed to fulfill our requirement of this Hands-On exercise—i.e., creating a different workflow based on the message in the handshake file for our package.

11. Add the following code below the comment "Add your code here."

```
Dim strFileReader As String
strFileReader = My.Computer.FileSystem.ReadAllText("C:\SSIS\
RawFiles\HandshakeFile.txt")
Dts.Variables("HandshakeMessage").Value = strFileReader
'If the mainframe process is still loading or the previous day's
processing is still running
'Change the handshake message to ELSE for special handling in the
right most branch of the logic.
If (strFileReader = "PROCESSING" Or strFileReader = "LOADING") Then
Dts.Variables("HandshakeMessage").Value = "ELSE"
ElseIf strFileReader = "OK" Then
My.Computer.FileSystem.WriteAllText("C:\SSIS\RawFiles\HandshakeFile
.txt", "PROCESSING", False)
End If
```

This code reads the HandshakeFile.txt that resides in the RawFiles folder. In real-life coding, you will be using a connection manager, probably with a variable that provides flexibility as to where the file has been placed, to connect to the file. The variable could be populated at run time by some other process before the script runs, giving you flexibility and control over the process. To keep things simple, I have used a direct connection to the file here. At this stage if you refer back to Figure 11-1 to remind yourself of the logic we need to develop, you will understand that this code better. The code reads the HandshakeFile.txt file and passes the value to the HandshakeMessage variable, and if the value is OK, then it writes PROCESSING in the HandshakeFile.txt file to implement the middle branch logic. For the scenario where the process has been invoked while the mainframe is still loading or the previous processing has still not finished, it will set the value of the HandshakeMessage variable to ELSE to implement the rightmost branch of logic.

12. Add a Data Flow task in the package and from the Script task, drag the precedence constraint and drop it onto the Data Flow task. Double-click the precedence constraint to open the editor. Select Expression in the Evaluation Operation field and type the following in the Expression field as shown in Figure 11-5.

```
@HandshakeMessage == "OK"
```

There is no space between the two equal (=) signs. Click Test to test the expression. Close the success message and the Precedence Constraint Editor by clicking OK twice.

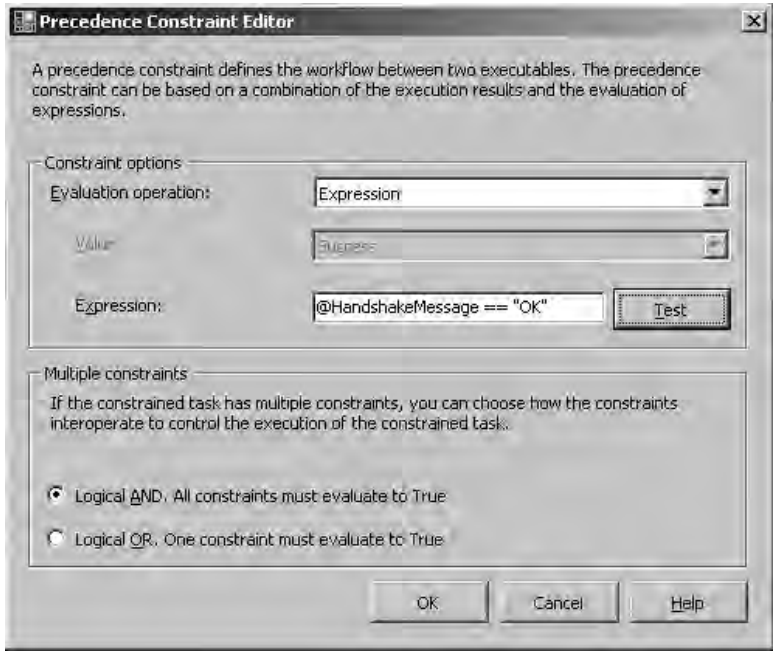


Figure 11-5 *Setting the Expression in the Precedence Constraint Editor*

13. Configure the Data Flow task to import the RawDataTxt.csv file into the Campaign database. The main items you will set up in this task are described next. If you've difficulty in setting up this task, refer to the code provided for this book, as the complete code for this exercise been included.

Source	Flat File Source to read RawDataTxt.csv file.
Destination	OLE DB Destination to import into Campaign.dbo.RawDataTxt table

14. Drop another Script task below Data Flow task and connect both of them with a success precedence constraint. This Script task will write the UPDATED message in the HandshakeFile.txt file if the HandshakeMessage variable has a value of OK and will complete the middle branch of our required work flow.
15. Configure the Script task as follows:

Name	Update HandshakeFile
ScriptLanguage	Microsoft Visual Basic 2008
EntryPoint	Main

And add the following script into the Main subroutine:

```
'Update the HandshakeFile.txt with UPDATED message if the processing has
started in this run
'Otherwise i.e. for any other status such as UPDATED or ELSE do nothing.
If Dts.Variables("HandshakeMessage").Value = "OK" Then
My.Computer.FileSystem.WriteAllText("C:\SSIS\RawFiles\HandshakeFile.txt",
"UPDATED", False)
End If
```

16. After completing the middle branch, let's complete the leftmost branch, which is very simple, as there is very little logic to implement in this branch other than that it has to be attached to the final Script task. Click the Determine Workflow Using Handshake Script task and drag the green arrow on to the final Script task labeled as Update HandshakeFile. This will set a constraint on the final Script task to be executed only when the two preceding tasks—the first Script task and the Data Flow task—get executed. We do not want this, as the workflow is mutually exclusive and both tasks will never be executed in the same run. We need an OR constraint; that is, the final Script task should execute when either of the tasks complete. Double-click the newly added green line and change the multiple constraints condition from Logical AND to Logical OR as shown in the Figure 11-6. Also, select Expression in the Evalutation operation field and set

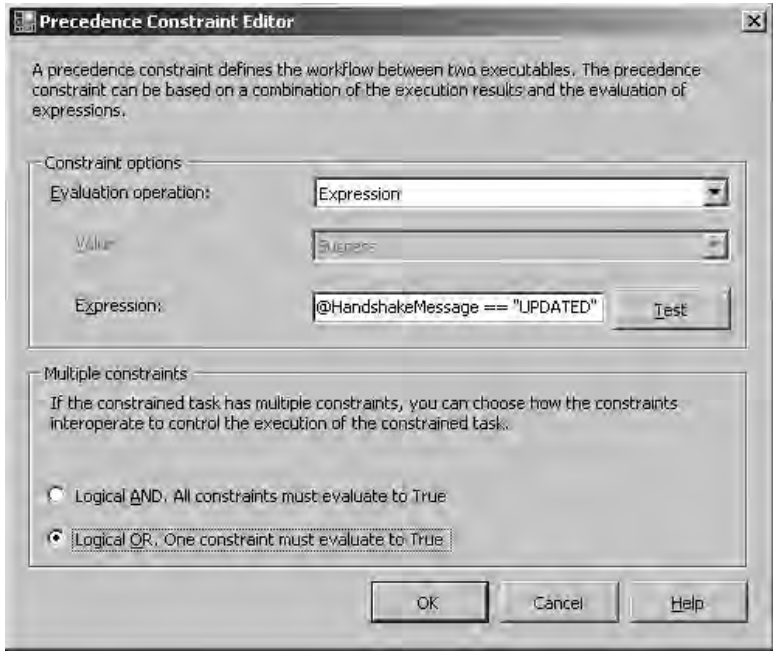


Figure 11-6 *Setting the Logical OR precedence constraint*

Expression as follows as we want the workflow to run via this branch only when the HandshakeMessage value is UPDATED. Close the Precedence Constraint Editor Dialog box.

```
@HandshakeMessage = = "UPDATED"
```

17. Now let's complete the rightmost branch of the work flow. Add another Script task in the package control flow on the right side and above the Data Flow task and join the first Script task and the new Script task. Open the precedence constraint, select Expression in the Evaluation Operation field, and set Expression as follows. Then click OK to close the editor.

```
@HandshakeMessage = = "ELSE"
```

18. Configure this Script task as follows:

Name	Check after delay
ScriptLanguage	Microsoft Visual Basic 2008
EntryPoint	Main
ReadWriteVariables	User::HandshakeMessage

And add the following script into the Main subroutine as shown in Figure 11-7:

```
Dim strFileReader As String
'MsgBox("Started _ " & TimeOfDay())
System.Threading.Thread.Sleep(5000)
'MsgBox("Finished _ " & TimeOfDay())
strFileReader = My.Computer.FileSystem.ReadAllText("C:\SSIS\RawFiles\
HandshakeFile.txt")
Dts.Variables("HandshakeMessage").Value = strFileReader
If strFileReader = "OK" Then
My.Computer.FileSystem.WriteAllText("C:\SSIS\RawFiles\HandshakeFile.txt",
"PROCESSING", False)
'Set the HandshakeMessage variable as ELSE if the loading or processing
doesn't finish in one hour.
'The message in the HandshakeFile.txt stays the same as it existed in the
file.
ElseIf (strFileReader = "LOADING" Or strFileReader = "PROCESSING") Then
Dts.Variables("HandshakeMessage").Value = "ELSE"
End If
```

In this script, first of all, a delay has been added for five seconds (though the logic required is for one hour, but for our test, I'm sure you wouldn't want to wait for that long!). The delay is using another subroutine, Sleep, that has been declared in a separate statement. After the delay, the handshakeFile.txt file is read and the HandshakeMessage variable is updated with the new value. In case the new value is OK, that is, the running process has completed during the time Script task was waiting, the PROCESSING string is written in the handshakeFile.txt; otherwise, the HandshakeMessage variable value is changed to ELSE.

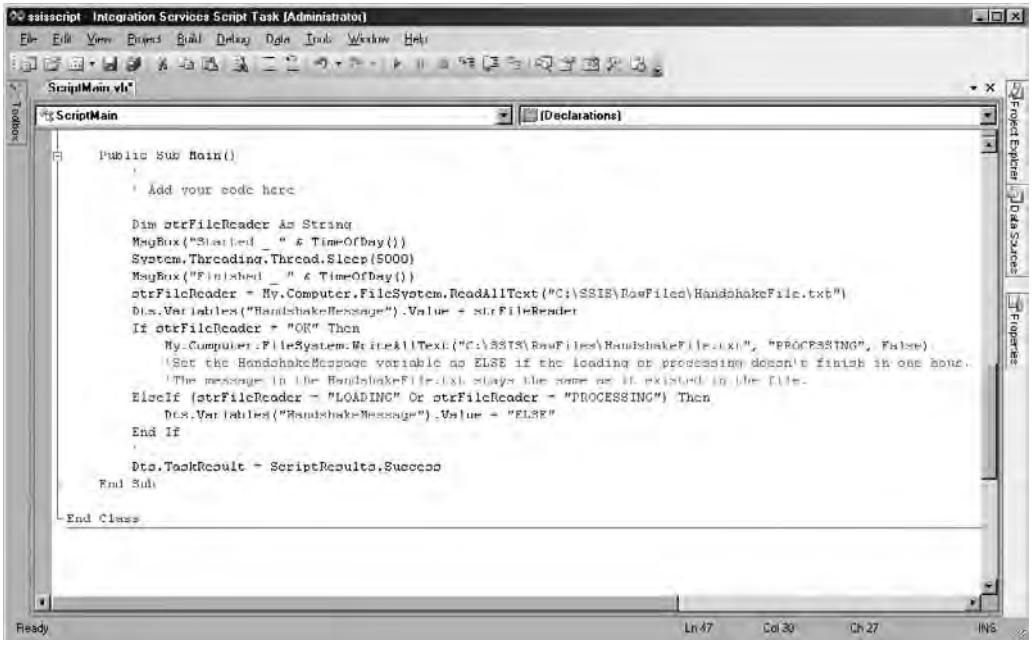


Figure 11-7 *Script for the Check After Delay Script task*

19. As per the logic, we need to add two branches: one to connect to the Import RawDataTxt Data Flow task and the other to the Update HandshakeFile Script task. Refer to Figure 11-8 to see the final layout of the control flow in the package. Double-click the precedence constraint that joins the Check After Delay Script task and the Import RawDataTxt Data Flow task to open the editor. Change the constraint option to Expression with the following value in the Expression field:

    ```
    @HandshakeMessage = = "OK"
    ```

    Also, change the multiple constraints option to Logical OR, as the Data Flow task now has two mutually exclusive constraints defined on it.

    Open the editor for second constraint from this Script task to the Update HandshakeFile Script task. This expression should already have Logical OR multiple constraints defined. Change the constraint option to Expression with the following value in the Expression field:

    ```
    @HandshakeMessage = = "ELSE"
    ```

Now that the logic has been implemented for all the three branches, it's time to run the package and test for various scenarios.

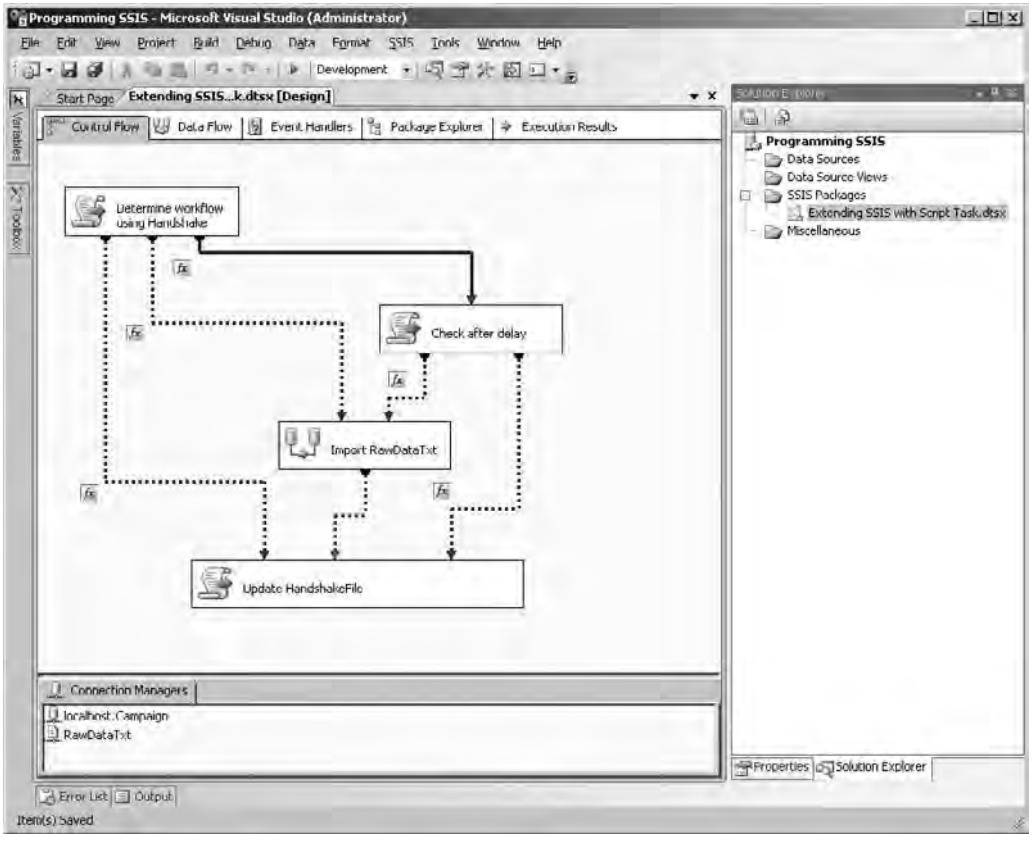


Figure 11-8 *Control flow for the Extending SSIS with Script task package*

Exercise (Debugging the Script Task)

The Script task provides breakpoints that can be applied in the script and allow you to step into the code to see how the values of variables change. We will set a breakpoint in the first Script task to see how the values change.

20. Double-click the Determine Workflow Using Handshake Script task to open the editor and then click Edit Script to open the VSTA IDE environment. Scroll through the code to the section where you've added your own code. Locate the third line that sets the HandshakeMessage variable value to strFileReader and click the left side gray bar. This will set a breakpoint on the line and will highlight the line in red as shown in Figure 11-9.

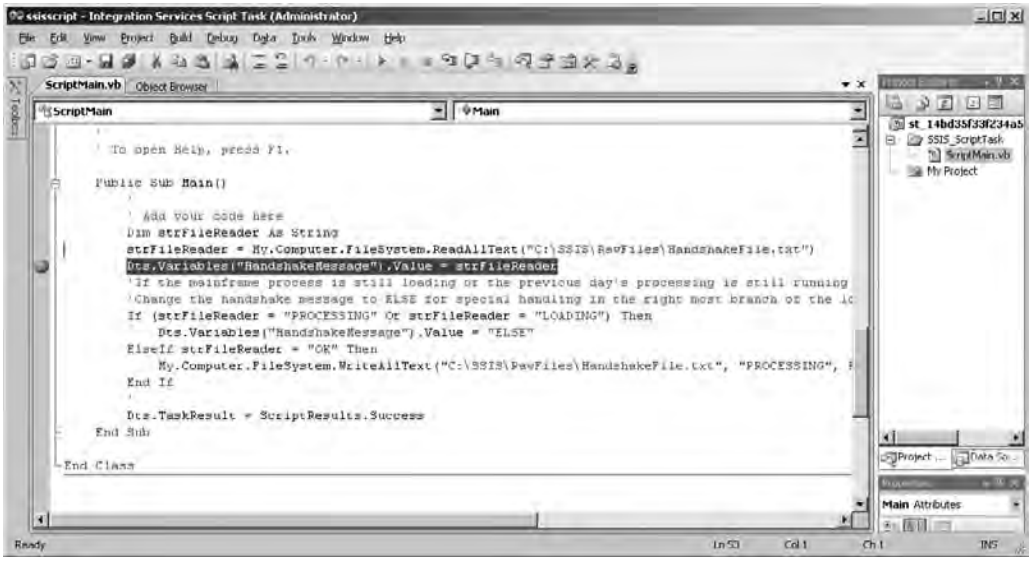


Figure 11-9 *Setting a breakpoint within the Script task script*

21. Close the VSTA IDE and the Script task by clicking OK. Notice that a breakpoint has been set up on the task. Right-click the Script task and note that the breakpoint you set earlier has been shown in this dialog box and the line and character number has also been specified. Click OK to close.
22. Make sure that the HandshakeFile.txt file in the C:\SSIS\RawFiles folder has an OK string before we start debugging. Also, be aware that the strings read from the file are case sensitive. The code has been built using all uppercase strings. Click the Start Debugging button in the BIDS to run the package. As the package execution starts and the first Script task changes to yellow color, VSTA IDE will open in couple of seconds to show you where the script breaks as it highlights the row in yellow. If the Locals window does not open, you can open it from the Debug | Windows | Locals command and press the autohide push pin on the top-right corner of the window to keep it open as you walk through the script.
23. There are a few ways to see the value of a variable—the Locals window, a watch or quick watch window, or directly in the code. Note that the strFileReader variable in the Locals window shows a value of OK. Now hover the mouse over the strFileReader in the breakpoint line; you will be shown the value as a pop-up hint. Sometimes the value could be a long string, in which case you can invoke the Text Visualizer by clicking the zooming lens symbol shown next to the value in the pop-up hint or from the Locals window. You can also open the QuickWatch window from the Debug menu or by pressing SHIFT-F9 and evaluating any expression or a variable. If you type **strFileReader** in the Expression field and click Reevaluate, you will be shown the current value.

24. Press F8 to step into the code and move forward line by line. Press F8 for three more times to stop at the End If line. Switch to Windows Explorer and open the HandshakeFile.txt. You will see the PROCESSING string in the file that has been written by the code. Close the file and switch back to the VSTA IDE. Complete the processing of the script and the package. Once the package execution has been completed, you can check the HandshakeFile.txt again to see the UPDATED message written in the file.
25. You can rerun the package using different strings in the HandshakeFile.txt file and applying breakpoints in the different Script tasks.

Exercise (Raising Events and Logging in the Script Task)

You can raise events within your script to report information, warnings, and errors to the package so that the package event handlers can respond to these events. The Events property of the Dts object is used to raise events. The Dts.Events property exposes the following methods to fire the events.

Event Method	Description
FireCustomEvent	Fires a user-defined custom event—e.g., you may want to capture certain data-related attributes.
FireError	Fires an error event.
FireInformation	Fires an informational message—e.g., information for auditing purposes.
FireProgress	Fires the on-progress event.
FireQueryCancel	Fires an event to indicate that the task should be canceled immediately.
FireWarning	Fires a warning that is less critical than an error—e.g., a warning that a varchar(50) column is being mapped to varchar(10), which will work as long as the data doesn't exceed ten characters.

Similar methods are exposed by the Script component in the Data Flow task that you will study later in this chapter; however, keep in mind that the usability for these methods is different in both the script objects in SSIS. The events raised using any of the previously mentioned methods can be logged to any log provider that has been enabled on the package. To learn more about SSIS logging or log providers, refer back to Chapter 8. The Script task, though, can also log information directly without using the user-defined events. The Dts.Log method is used to log the information to all the log providers enabled on the package. When you add a Script task to your package, a custom log entry ScriptTaskLogEntry is made available in the log events to log information when the code is run inside the Script task. To configure logging in a package for a Script task, you have to select this log entry in the Configure SSIS Logs dialog box.

26. Add the following line of code before the End If line to raise an error and fail the package. For this code to be run, change the value in the HandshakeFile.txt to any other string, say ABC.

```
Else : Dts.Events.FireError(0, "Determine Workflow using Handshake", "UNKNOWN
STATUS IN THE HANDSHAKEFILE: " + strFileReader, String.Empty, 0)
```

Now if you run this package and step through the code, you will see that this line of code is executed and Script task turns red, failing the package. You can check out the following message displayed in the Output window or the Progress tab.

```
[Determine Workflow using Handshake] Error: UNKNOWN STATUS IN THE
HANDSHAKEFILE: ABC
```

Review

You have used the Script task three times with minor variations in this exercise and understand how easily you can develop a custom functionality with this task. You've used the Dts object, which provides almost all the interactivity to the Script task, and you've used its two properties, Variables and TaskResult, in this example. Don't forget there are other important properties, such as Connections, ExecutionValue, and Transaction, that can be quite helpful for building custom functionalities, while the properties like Events and Log can be used to raise errors, warnings, and informational messages and log them to any of the log providers that have been enabled in the package. I would encourage you to take time to go through the examples available in the Books Online and on the web for the Script task. At this time, it will be worthwhile to highlight some of the best practices and the ideas that you can use while scripting:

- Always try to use your code within TRY/CATCH block. The previous examples have not used TRY/CATCH purposely to keep them simple and understandable. However, it is not difficult to implement that in real life.
- Try to use package connection managers in the script using the Dts.Connections property instead of creating connections afresh, as the package connection managers come with the benefits of providing the connection information and the flexibility to change the connections using package configurations.
- Remember that the TaskResult property can be used to force the return status to the Integration Services run time. Using this, you can better control the workflow in your package. And if you need, you can return a value or information using the ExecutionValue property.
- And finally and probably a more powerful feature, don't forget the ability of the Script task to use the Microsoft.VisualBasic namespace (and the Microsoft.CSharp namespace in case you're using C#), the .NET Framework class library, and the

custom-managed assemblies, to implement custom functionality. You can also enable your script to access objects and methods from a web service when you use the Add Web Reference command in the VSTA. To add a reference to a managed assembly, you need to perform actions for design-time access and run-time access separately. To allow access at design time, save the managed assembly on your local computer and use the Add Reference command in VSTA (right-click the project or use the Project menu), and in the Add Reference dialog box, go to the Browse tab and there locate and add the managed assembly. For run-time access, you need to sign the managed assembly with a strong name and install the assembly in the GAC (global assembly cache) on the computer on which the package will be run.

Script Component

The Script component has been mentioned briefly in the previous two chapters, where you learned that this component can be used as a data source, a transformation, or a destination. The basic functionality of this component is to enable you to write custom code and include that custom code as a component inside the package. This custom code will be executed for every row of data, as the Script component is part of the row transformations. The Script component makes it easy to extend the data flow with custom components by writing lot of infrastructure code for you. Just like the Script task, this component also allows you to write custom code using either Microsoft Visual Basic 2008 or Microsoft Visual C # 2008. Using the Script component as a transformation, you can write custom code to call functions that are not available in SSIS—for instance, you can call the .NET assembly to use working code that is available outside SSIS; build transformations that are not available in SSIS; or apply multiple transformations with custom code to enhance the performance of your package. You can, in fact, write custom code to do all the transformations within the Script component. When the Script component is configured to work as a transformation, it provides an input and multiple outputs, but no error output. However, you can direct your error rows to one of the outputs using the ExclusionGroup property and the Row.DirectRowTo<Output Name> method and simulate it as an error output. You will learn about this in detail later in this chapter.

You may be wondering how the Script component is different than the Script task and when you should choose one over the other. So, before we go into exploring the Script component more, let's understand the differences between the two scripting objects provided in the Integration Services.

Script Task vs. Script Component

While both the objects provide scripting environments using the VSTA engine, yet there are inherent differences in their purpose and functions. This is due to their design architecture and the placement in development environment: the Script task is made available in the Control Flow tab, while the Script component works in the Pipeline or the Data Flow tab. You can use either object to extend your package, but they extend different aspects of the package and shouldn't be used to perform each other's function. While the Script task extends the workflow, the Script component is used to create custom functionality within the pipeline or the data flow. You may create some data flow functionality in the Script task, but this is not what it is designed for. You will find it much easier to develop and extend your data flow custom script with the Script component than with the Script task. For example, you can create a source or a transformation or a destination with a Script component quite easily and focus on the code required to develop that particular component.

Depending on what you choose to build with a Script component—i.e., a source, a transformation, or a destination—you will be configuring different metadata and properties in the editor; for instance, for a source you will be configuring outputs only, as there are no inputs for a source.

The choice of function and the metadata and properties you configure in the editor results in creation of different members of the base classes in the auto-generated code. The Script task does not provide any such facility, as the code generated is always the same.

One of the main differences you should know is the way the script is executed by both the objects. The Script task executes the script only once for each execution instance of the task, while the Script component generally executes the script for each row, as it applies the custom transformation on each of the data row it reads. This difference alone makes the Script component a much desirable and powerful option for data-related operations.

The abilities of both objects to generate and work with different base classes also differentiate them in the way you can write scripts. While working with the Script task, you realized that most of the interaction of the task with the package is provided by the global Dts property of the ScriptMain class. The Dts property works with only the Script task and is not available in the Script component, which uses typed accessor properties to access certain package features such as variables and connection managers. We will go through various code examples when we work with the Script component, but here just keep in mind that the programming object model for the Script component is different than that of the Script task.

Some other minor differences to note are that the Script component does not support breakpoints, and you use alternate methods such as a MessageBox to interrupt the execution of the script or raise events that you can examine in the Progress tab or log events to examine later when the execution has completed. I'll just mention here that some developers prefer to use trace listeners by using the Trace.Writeline method of the System.Diagnostics namespace. Trace Listeners are objects that capture the tracing information and route it outside your application to the place where you want to store it, for instance, in a text file or an event viewer. While doing that, you have an opportunity to watch the trace results using available utilities. If you want to have more visibility to your debug information, you can use this method.

Finally, the Script component does not report the execution results as reported by the Script task using the TaskResult or ExecutionValue properties. Once added in a data flow, the Script component becomes an integral part of the pipeline and the success or failure depends on the execution status of Data Flow task.

Hands-On: Extending Data Flow with the Script Component

A Script component can be configured as a data source, a transformation, or a destination within the Data Flow task. We will now do one Hands-On exercise to import a nonstandard text file for sales, derive a bonus based on employee title and the sales the employee has achieved, and finally will write those values into a comma-delimited and nonstandard text file. Sometimes even though you can perform some operation using preconfigured components, you may still prefer to use a Script component if you find that it is easier to keep the logic in one place or you're using too many preconfigured components that can otherwise be built easily using a Script component. In this exercise, though we will be working with text files, the Script component can virtually work with any type of source—in fact, it will be correct to say that it can be configured to read and write from any source or destination that is used in most enterprises.

Script Component as a Data Source

In this part, let's explore how the Script component can be configured to work as a data source to import nonstandard structured data from a text file. We will import the Sales .txt file saved in the C:\SSIS\RawFiles folder. Open the file and note the structure of the data it contains. The file contains information of one record in six rows that could otherwise be one database row with some blank rows and some sundry comments in between the records. This is not a standard comma-delimited file that SSIS can easily

import. We have to write custom code to parse the good rows of data into the correct columns. So, let's start configuring our first Script component.

1. Add a new Integration Services package in the Programming SSIS project and rename it as **Extending Data Flow with Script Component.dtsx**.
2. Add a Flat File Connection Manager for C:\SSIS\RawFiles\Sales.txt file with default settings and rename it to **Sales**.
3. Add a Data Flow task in the newly created package and double-click it to go to the Data Flow tab. Drag and drop the Script component onto the Data Flow Designer surface. The Select Script Component Type dialog box will pop up on the screen (see Figure 11-10). Select the radio button to configure the component as a Source and click OK to place this task as a source on the Data Flow Designer surface.
4. Rename the component as **Script Source Component** and double-click to open the Script Transformation Editor. Change the ScriptLanguage to Microsoft Visual Basic 2008.
5. Rename the Output 0 in the Inputs And Outputs page to **SalesOutput**. Note that you have only one output available in this tab and no inputs. Though you can add more outputs if you wish to create a multiple outputs source, you can't add an input here. This is because the Sources have no inputs. You refer to outputs (and inputs) with their names suffixed by buffer in the code. For example, the SalesOutput will be referenced in the code with the name SalesOutputBuffer.

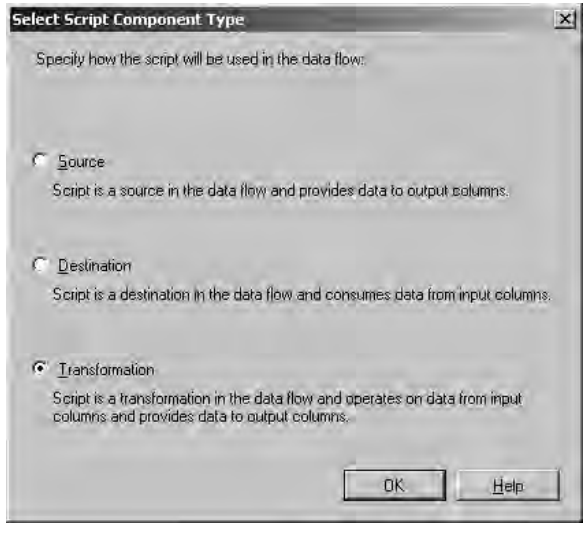


Figure 11-10 *Selecting the Script component type in the Data Flow Designer*

6. Expand the SalesOutput and click the Output Columns node. Now you can add columns to SalesOutput by clicking Add Column. Click this button four times to add four columns with the following details as shown in Figure 11-11.

Column Name	Data Type	Length
FirstName	string [DT_STR]	20
LastName	string [DT_STR]	20
Title	string [DT_STR]	20
SalesAmount	four-byte signed integer [DT_I4]	

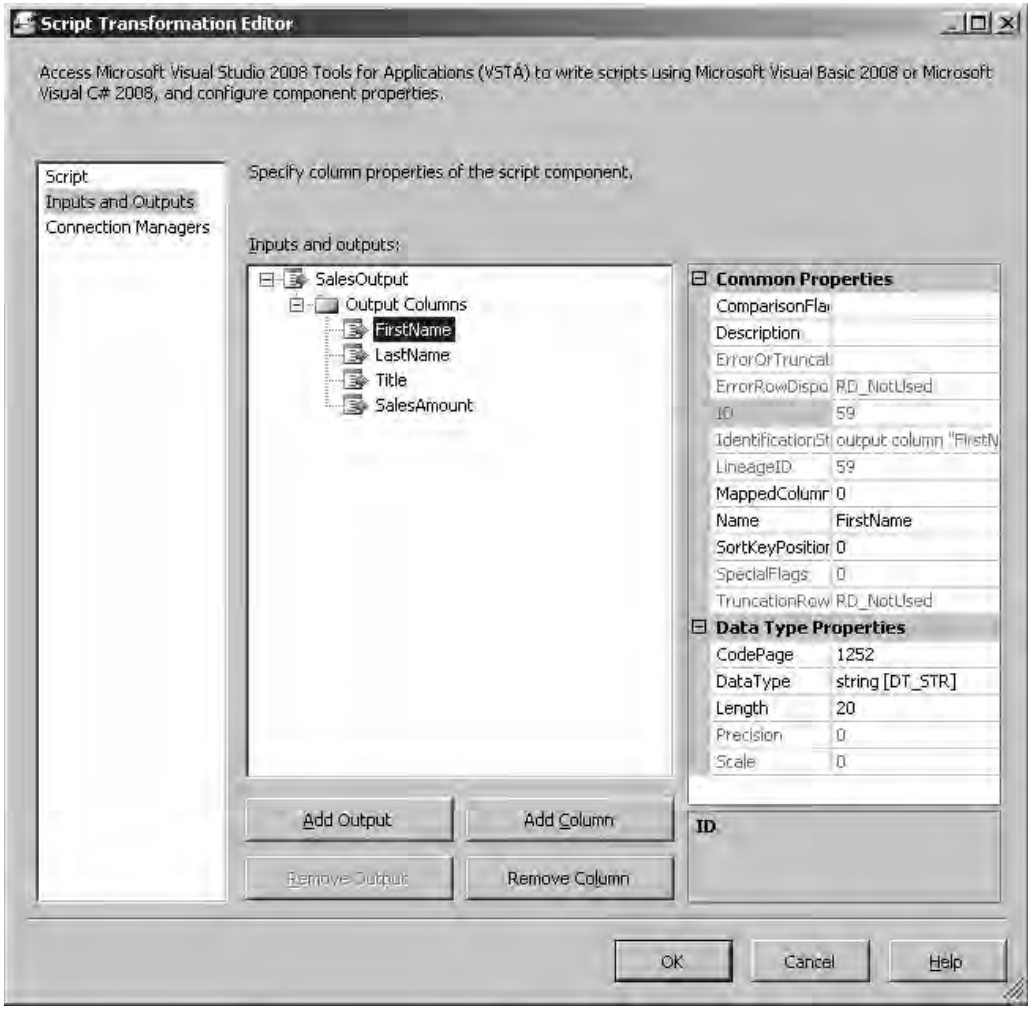


Figure 11-11 *Adding columns in the Script Source Component Outputs*

Remember that when you decide on a data type for the columns here, it is efficient to choose the correct data types; for instance, we used length 20 in this case instead of default 50. The space we save by using the correct data type that is sufficient enough to fit the data will mean more rows can be fit into the data buffers and hence SSIS can work faster. It is as if to say, you need to take fewer buckets out of the well if you fill your buckets up to the full capacity.

7. Now go to the Connection Managers page and add the Sales Connection Manager. Change the name from Connection to **Sales** in the Name field. This name is exposed inside the script and makes accessing the package connection managers quite easy in your script.
8. Now click the Edit Script button in the Script page to open the scripting environment. As you can see this environment is quite different than the one you've seen earlier in the Script task. So, let's spend some time here to understand various parts of this auto-generated code.

   First of all, notice in the Project Explorer window that there are three project items in the ScriptComponent project—BufferWrapper, ComponentWrapper, and Main.

   - **BufferWrapper** The classes in the BufferWrapper project item provide methods for working with the data flow buffers and typed properties for each column. Double-click the BufferWrapper.vb project item to the auto-generated code. It contains a public class for each output buffer, in our case, only one SalesOutputBuffer, and two typed write-only properties for each column; one with the column name to refer to the column in the code and the other one with the column name suffixed with _IsNull to set the column value to null. Scroll all the way down to see an AddRow method that is used to add an empty row to the output buffer, and a SetEndOfRowset method that determines, using the EndOfRowset function, that the current buffer is the last buffer of data and passes this information to the data flow engine. Note that the ScriptBuffer class serves as the base class for the read-only classes representing the input and the outputs. You are not supposed to edit this auto-generated code, as it will be overwritten when you modify the Script component.
   - **ComponentWrapper** The classes in the ComponentWrapper project item provide methods and properties to process data and to interact with the package objects. Double-click ComponentWrapper.vb to see the auto-generated code. This project item creates a UserComponent class that is inherited from the ScriptComponent class. The ComponentWrapper has an overridden implementation of PrimeOutput method that is called only once at run time. The PrimeOutput method prepares the outputs to accept

new rows and is used for the outputs where you add new rows to the output buffers such as a source or a transformation with asynchronous output. This method then passes the processing to CreateNewOutputRows method, the FinishOutputs method, and MarkOutputsAsFinished, which sets the end of rowset on the last output buffer. Note that the ComponentWrapper item also contains Connections and Variables collection classes. As with BufferWrapper, you are not supposed to edit this auto-generated code, as it will be overwritten when you modify the Script component.

- **Main** This project item contains the ScriptMain class, which inherits from the UserComponent class. In contrast to the other two project items where you don't write your code directly against them, you will be writing your code here using methods and properties provided by the derived classes in this project item. Double-click main.vb to see the auto-generated code for the configurations you've done earlier in the metadata design mode. Also, bear in mind that the auto-generated code in the Main.vb item is generated only when you click the Edit Script button for the first time. Later, when you make changes to the Script component, for example, you may add more outputs; the Main.vb code is not updated with those changes and you have to add methods and properties manually to perform the additional functions. This behavior also means that the code you write in the Main.vb project item is persisted within the Script component and the auto-generation of code doesn't affect it.

Note that the imports statements have two wrappers. These are the Primary Interop Assemblies for their respective namespaces. The run-time wrapper provides the classes and interfaces used to access the Control Flow components in the run time, while the pipeline wrapper provides the classes and interfaces used to create custom Data Flow components. This implies that the wrappers help your custom script to access package objects such as variables and connection managers, in the same way as the Dts global object provides access in the Script task.

The ScriptMain class is the entry Point class here, but there is no Main() subroutine such as you've seen in the Script task. In a script Source component, most of the work is performed in the CreateNewOutputRows() subroutine. The CreateNewOutputRows method is used along with the AddRow method to add new rows to the output and is primarily used when you're writing code for a source or for an asynchronous output. You will study the synchronous and asynchronous outputs in the next part of this Hands-On exercise. There are two more subroutines, PreExecute() and PostExecute(). As the names indicate, these subroutines run any one-time tasks necessary before and after the Script component has processed its inputs and outputs to perform, such as initializing or closing connections.

9. As we are going to use StreamReader to read our nonstandard text file, add the following namespace in the Imports statements.

```
Imports System.IO
```

Declare the following in the ScriptMain class:

```
Private textReader As StreamReader
Private SalesFile As String
```

10. Next, to use the package connection manager Sales in your script—the connection manager you specified in the Connection Managers page in the Script component GUI—you can use the AcquireConnections method along with the IDTSConnectionManager100 interface that returns a reference to the required connection manager. You can override the AcquireConnections method to retrieve the connection information from the Sales Connection Manager. Add the following subroutine below the previous statements as shown in Figure 11-12.

```
Public Overrides Sub AcquireConnections(ByVal Transaction As Object)
Dim connMgr As IDTSConnectionManager100 = Me.Connections.Sales
SalesFile = CType(connMgr.AcquireConnection(Nothing), String)
End Sub
```

11. As the connection has been set to the Sales.txt file using SalesFile connection manager string in the previous step, now you can initialize the textReader to connect to the SalesFile. As this is a one-time operation that needs to be performed before the component starts reading the file, we will use the PreExecute method. Add the following lines in the PreExecute subroutine:

```
MyBase.PreExecute()
textReader = New StreamReader(SalesFile)
```

12. While you have opened the text reader, it needs to be closed after all the rows have been processed. As this is a one-time operation that needs to be performed after processing of rows, you will use the PostExecute method for this. Add the following code in the PostExecute subroutine:

```
textReader.Close()
```

13. Finally, you can create new output rows as the text reader reads the text file. Add the following piece of code in the CreateNewOutputRows subroutine (refer to Figure 11-12).

```
Dim textLine As String
textLine = textReader.ReadLine
Do While textLine IsNot Nothing
  If textLine.StartsWith("BEGIN_RECORD") Then
    SalesOutputBuffer.AddRow()
  ElseIf textLine.StartsWith("FirstName: ") Then
    SalesOutputBuffer.FirstName = textLine.Remove(0, 11)
  ElseIf textLine.StartsWith("LastName: ") Then
    SalesOutputBuffer.LastName = textLine.Remove(0, 10)
```

```
main.vb*
ScriptMain                                  AcquireConnections(Object)
Imports System
Imports System.Data
Imports System.IO
Imports System.Math
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports Microsoft.SqlServer.Dts.Runtime.Wrapper

<Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute> _
<CLSCompliant(False)> _
Public Class ScriptMain
    Inherits UserComponent

    Private textReader As StreamReader
    Private SalesFile As String

    Public Overrides Sub AcquireConnections(ByVal Transaction As Object)
        Dim connMgr As IDTSConnectionManager100 = Me.Connections.Sales
        SalesFile = CType(connMgr.AcquireConnection(Nothing), String)
    End Sub

    Public Overrides Sub PreExecute()
        MyBase.PreExecute()
        textReader = New StreamReader(SalesFile)
    End Sub

    Public Overrides Sub PostExecute()
        MyBase.PostExecute()
        textReader.Close()
    End Sub

    Public Overrides Sub CreateNewOutputRows()
        Dim textLine As String
        textLine = textReader.ReadLine
        Do While textLine IsNot Nothing
            If textLine.StartsWith("BEGIN RECORD") Then
                SalesOutputBuffer.AddRow()
            ElseIf textLine.StartsWith("FirstName: ") Then
                SalesOutputBuffer.FirstName = textLine.Remove(0, 11)
            ElseIf textLine.StartsWith("LastName: ") Then
                SalesOutputBuffer.LastName = textLine.Remove(0, 10)
            ElseIf textLine.StartsWith("Title: ") Then
                SalesOutputBuffer.Title = textLine.Remove(0, 7)
            ElseIf textLine.StartsWith("SalesAmount: ") Then
                SalesOutputBuffer.SalesAmount = textLine.Remove(0, 13)
            End If
            textLine = textReader.ReadLine
        Loop

    End Sub

End Class
```

Figure 11-12 *Code listing for the Script Source Component*

```
    ElseIf textLine.StartsWith("Title: ") Then
      SalesOutputBuffer.Title = textLine.Remove(0, 7)
    ElseIf textLine.StartsWith("SalesAmount: ") Then
      SalesOutputBuffer.SalesAmount = textLine.Remove(0, 13)
    End If
  textLine = textReader.ReadLine
  Loop
```

This code reads a line of using the textReader ReadLine function and stores the line into the textLine string variable. The read process is in the while loop that goes on till there is nothing to read. Once a line has been read, it then passes through various case statements and, depending upon the evaluated conditions, it either adds a new row to the output buffer using the AddRow method or adds the data into any of the evaluated columns. Last, it is a good practice to use a SetEndOfRowset method that determines that the current buffer is the last buffer of data so that the downstream components know that no more rows are expected. One example of SetEndOfRowset is shown later in the chapter (refer to Figure 11-17). Close the scripting environment and click the OK button on the Script Transformation Editor as you've created a Script Source component.

14. Though we will debug the package when it is ready, running the package now to test the script source will be a good idea and will confirm that the code we have built so far is working. To run the package and see how the file has been read, we will use a Row Count transformation, as it can consume pipeline rows without requiring a destination. Also, we will need a variable to configure this transformation.
15. Create a variable varRecords at the package scope of the Int32 data type with 0 as its value. Drop the Row Count transformation below the Script Source component and join both of them with the data flow path.
16. Add a grid-type data viewer to the path to row count to see the records in a tabular format. To add a data viewer, double-click the data flow path, go to Data Viewers page, click Add, and choose the Grid type data viewer.
17. Execute the package and you will see the SalesOutput Data Viewer showing the records that we wanted to read in a tabular format as shown in Figure 11-13. At this time if you open the file, you can appreciate that the Script component has read the data from a nonstandard text file and formatted it in a much easier to read and easier to operate on format. Also, it has nicely ignored the comments in the text file that we didn't want to read anyway.

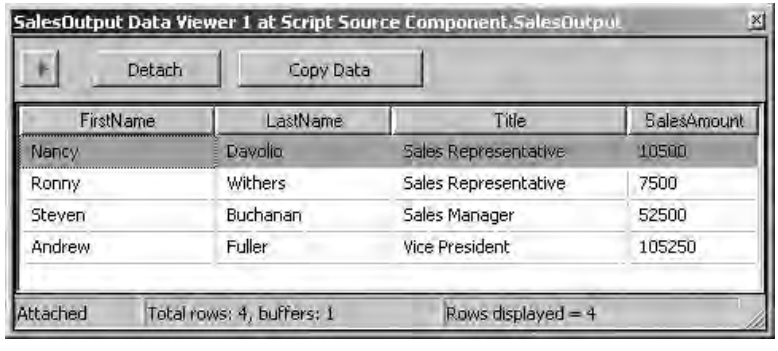


Figure 11-13 *SalesOutput Data Viewer showing the data in a grid format*

Script Component as a Transformation

In this section, you will learn to configure a Script component as a transformation. You will be writing code for a script transformation more often than writing code for a script source or a script destination. So, it is more important to understand the components involved in designing code for a transformation. You have imported Sales data into the pipeline using a script source; now in this part of the exercise, you will derive bonuses paid out to the employees based on title and the sales amount that each employee achieves. Business also wants to know the total sales amount and the total bonus disbursed as a separate reports. The business rules are defined in this way: Bonuses will be paid to all employees who achieve their targets. The target for a Sales representative is ten thousand dollars, that for a Sales Manager is fifty thousand dollars, and that for Vice President is one hundred thousand dollars. A bonus is paid at a fixed rate of 2 percent of the sales amount, and business wants to know who has been paid bonuses and how much. Second, business also wants to know the aggregated sales and bonus amounts.

From this description, you can very well understand that you need to derive an indicator for bonuses paid and derive the bonus amount per employee if he or she has achieved the target. Second, you need to aggregate sales amount and the bonus amount and write those values into a separate file. It is also evident that while the first requirement can be derived using row-based operations, the second requirement is a complete rowset-based operation. This also leads us to a brief discussion of synchronous and asynchronous components. These components will be covered in detail in Chapter 15, but here just keep in mind that a synchronous component is one in which the output is synchronous to the input—for instance, the rows get processed as they come and row-level operations are performed such as deriving a column value based on the other column values. On the other hand, the asynchronous components are the ones that perform operations on the complete rowset instead of one row, such as aggregations and sorting operations. These operations need all the rows before they can provide outputs. Moreover, the output rows can be different (generally less) than the input rows, which is opposite to the synchronous component, which outputs the same number of rows it receives at the input. One last but very important difference from the point of view of writing code: The synchronous components work on the same buffers of data sets and do not create or write data to new buffers; they simply add or change data in the same buffer. On the other hand, asynchronous components block inputs, collect all the input rows, perform the required operations, and write outputs to new buffers, which means they work with more than one buffer at a time. This also explains why the asynchronous components generally need more memory.

So, we will create two outputs for our script transformation. They will be, as you can guess, synchronous and asynchronous to meet both the requirements.

Scripting a Synchronous Transformation

In this part you will create an IsBonusPaid indicator based on the targets and will derive the bonus using the sales amount. Though this kind of operation doesn't need SSIS to be scripted, as the Derived Column transformation is quite capable of performing such operations, this example will show you how you can implement complex business requirements that otherwise can't be met using preconfigured components in Integration Services.

18. When you are ready to proceed, create a variable for the bonus calculation at the package scope with the following details:

Name	varBonusMultiplier
Data Type	Int32
Value	2

19. Delete the data flow path between Row Count and the Script Source component. Drop a Script component in between the Script Source component and the Row Count transformation. Choose Transformation and click OK in the Select Script Component Type dialog box. Rename this as the **Script Transformation Component**.
20. Join this new transformation component with the source and the row count transformation. Double-click the newly added script transformation component to open the editor. Change the ScriptLanguage to Microsoft Visual Basic 2008.
21. Just as with a Script task, you can make the package variables available to your script using ReadOnlyVariables or ReadWriteVariables fields. Select the User::varBonusMultiplier variable in the ReadOnlyVariables field.
22. Note that you have one additional page, this time called Input Columns, in which you can select the columns you want to work with. In this page, when you select an input column, you can specify the Usage Type as ReadOnly or ReadWrite. This is quite a handy feature from a data security perspective, as it won't let the columns marked for the ReadOnly usage type get accidentally updated. For example, you may want to derive data using your custom code and want to output that derived data in the new output column; it is advisable to mark the input column's usage type as ReadOnly. You can assign an alias to the output column in the Output Alias column in case this is a ReadWrite column and you're going to update this column. Also, keep in mind that you don't need to select all of the available fields in the input; rather, it is efficient to select only the fields you're going to work with. The unselected fields will be passed on to the downstream component as is without any changes. Typically the synchronous component works only on the columns used in the derivations in a buffer and rest of columns

in the buffer stay untouched and the buffer is passed on to the next component in the data flow.

23. Select the Title and SalesAmount columns and assign them a ReadOnly usage type.
24. Go to Inputs and Outputs page. Note that you've two items here, Input 0 and Output 0. This is because a transformation has both the inputs and the outputs. Rename the input 0 to **XfrInput** and the Output 0 to **SynchXfrOutput**. Expand the XfrInput to see the Title and SalesAmount fields that you've selected in the Input Columns page. Note that you can add more outputs here but not inputs. This is because a Script component can have only one input when used as a transformation or a destination. Also, remember that a script source doesn't have any input. Click the SynchXfrOutput. Here are two important properties that need to be understood properly, as they can be quite useful on some occasions.

    The first property is the ExclusionGroup, which is set to 0 by default. This implies that all the input rows are sent to all the outputs. If you add more than one output to your Script component and you do not want all the rows to be sent to all the outputs—i.e., you want to split the rows between outputs based on some criteria—you will need to set the ExclusionGroup value to be the same on all the outputs to indicate that you want to split the rows among all the outputs; this may be any arbitrary nonzero value. In this case you will use DirectRowTo<Output Name>Buffer method in your code to decide where to direct the different types of rows. For instance, along with usual data split, you can actually use this method to direct error rows to one of the outputs and have a made-up error output, which otherwise is not provided by the Script component.

    The second property is SynchronousInputID that by default contains the ID of the input for the first output. This tells the Data Flow task to add rows from the input buffer to the output buffer of the component. When you add a second output or more outputs to your component, you need to set this ID yourself, as it is not set automatically. If you set the value of SynchronousInputID to None, the output becomes asynchronous, in which case you must add the rows to the output buffer after applying transformation logic on the input rows. You will use this attribute while adding an asynchronous output to this component in the next part of this Hands-On exercise.
25. With this component, you will be creating two more derived columns to meet the exercise requirements. So, add two output columns in the SynchXfrOutput with the following details:

Column Name	Data Type	Length
IsBonusPaid	string [DT_STR]	1
BonusAmount	numeric [DT_NUMERIC]	

As you won't be connecting to any outside data source, you don't need to any connection manager in the Connection Managers page.

26. Now that you've completed the configurations of the synchronous transformation component in the Metadata design mode, it is time to write some code in the Code design mode. Click Edit Script to open the scripting environment. Let's take some time to understand the auto-generated code and the differences in this code compared to when we create a source.

    Open the BufferWrapper.vb project item. Note that there is only one class with the name of input buffer—i.e., XfrInputBuffer—and all the columns are exposed as typed properties, including the ones that were created in the output. There is only one buffer class instead of the two that one may expect—one for input and one for output. This indicates and confirms that in a synchronous transformation the transformations are applied on the same buffer in place and the buffer is passed on to the next component as is without moving data to a new buffer. This enables synchronous transformations to be able to flow the data as it appears in the input and perform the transformations at a very fast speed. The XfrInputBuffer exposes two typed accessor properties for each column, one with the column name to refer to the column and the other one with the <Column Name>_IsNull name used to set the value of a column to null. In the end, it contains the AddRow method and the SetEndOfRowset method along with a function EndOfRowset to mark the end of a rowset in the pipeline.

    Now open the ComponentWrapper.vb project item. Note that a ProcessInput method has been provided here. The ProcessInput method is repeatedly called for each row as it receives the buffers from the upstream component until the end of the rowset is reached. Another subroutine, XfrInput_ProcessInput, is created here. This method is also used in the ScriptMain class in the Main.vb project item. If you change the name of the input, the ComponentWrapper will regenerate the code and will use the updated name in these methods; however, the Main.vb item doesn't regenerate the code and hence, will need to be modified manually. Scroll down to where the Variables class declares a typed property varBonusMultiplier, the same name as our package variable, and returns the package variable value. We will use this property in our custom code to access the package variable in the Main.vb project item. Also in the ScriptMain class, note that you don't have CreateNewOutputRows as you did in the source component; rather, you have the XfrInput_ProcessInputRow method that is derived from the ComponentWrapper project item and will be run for each data row.

27. We will need two variables in the script to calculate the values for the output columns. Though the calculated values could also be passed on to the output variables directly without involving intermediate script variables, yet it is a better practice to use script variables. Use of script variables helps in capturing calculated values in your script that you can use for further processing or for auditing purposes.

If you pass the calculated value directly to an output column, you can't read back that value, as the output columns are write-only columns. We will see this clearly while we work with asynchronous output. Add the following lines just after the Inherits UserComponent line in the ScriptMain class (refer to Figure 11-14):

```
Dim BonusPaid As String
Dim Bonus As Decimal = 0
```

```
main.vb*
(General)                                   (Declarations)

Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports Microsoft.SqlServer.Dts.Runtime.Wrapper

<Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute> _
<CLSCompliant(False)> _
Public Class ScriptMain
    Inherits UserComponent

    Dim BonusPaid As String
    Dim Bonus As Decimal = 0
    Public Overrides Sub PreExecute()
        MyBase.PreExecute()

    End Sub

    Public Overrides Sub PostExecute()
        MyBase.PostExecute()

    End Sub

    Public Overrides Sub XfrInput_ProcessInputRow(ByVal Row As XfrInputBuffer)

        Dim Bonusfactor As Integer = Me.Variables.varBonusMultiplier
        If (Row.Title = "Sales Representative" And Row.SalesAmount > 10000) Then
            BonusPaid = "Y"
            Bonus = Row.SalesAmount * Bonusfactor / 100
        ElseIf (Row.Title = "Sales Manager" And Row.SalesAmount > 50000) Then
            BonusPaid = "Y"
            Bonus = Row.SalesAmount * Bonusfactor / 100
        ElseIf (Row.Title = "Vice President" And Row.SalesAmount > 100000) Then
            BonusPaid = "Y"
            Bonus = Row.SalesAmount * Bonusfactor / 100
        Else : BonusPaid = "N"
            Bonus = 0
        End If

        Row.IsBonusPaid = BonusPaid
        Row.BonusAmount = Bonus

    End Sub

End Class
```

Figure 11-14 *Code listing for Script Synchronous transformation*

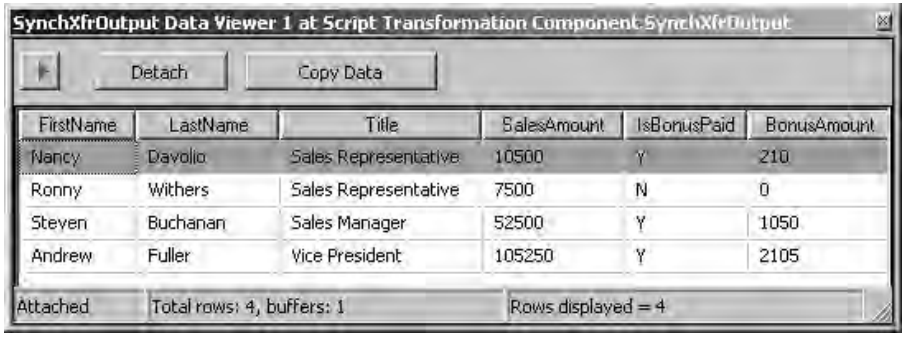
28. Add the following code in the XfrInput_ProcessInputRow subroutine in the ScriptMain class as shown in Figure 11-14.

```
Dim Bonusfactor As Integer = Me.Variables.varBonusMultiplier
  If (Row.Title = "Sales Representative" And Row.SalesAmount > 10000) Then
    BonusPaid = "Y"
    Bonus = Row.SalesAmount * Bonusfactor / 100
  ElseIf (Row.Title = "Sales Manager" And Row.SalesAmount > 50000) Then
    BonusPaid = "Y"
    Bonus = Row.SalesAmount * Bonusfactor / 100
  ElseIf (Row.Title = "Vice President" And Row.SalesAmount > 100000) Then
    BonusPaid = "Y"
    Bonus = Row.SalesAmount * Bonusfactor / 100
  Else : BonusPaid = "N"
    Bonus = 0
  End If
```

29. As you can see, scripting a synchronous component is quite easy. Close the scripting environment and the Script Transformation Editor. You've completed scripting a synchronous component.
30. As you tested the source component, you can test the synchronous component also. Add a grid-type data viewer to the path connected to Row Count transformation. Execute the package and you will see the two derived columns in the SynchXfrOutput Data Viewer as shown in Figure 11-15.

Scripting an Asynchronous Transformation

In this part, we will calculate total sales and total bonus paid. As you can see, this aggregation cannot be done unless all the records have been processed. This is a clear case for asynchronous output. You can add another Script component and can configure it as an asynchronous transformation. However, we will use the already-added script transformation and will add asynchronous output to it to demonstrate that a Script component can have both a synchronous output and an asynchronous output.



FirstName	LastName	Title	SalesAmount	IsBonusPaid	BonusAmount
Nancy	Davolio	Sales Representative	10500	Y	210
Ronny	Withers	Sales Representative	7500	N	0
Steven	Buchanan	Sales Manager	52500	Y	1050
Andrew	Fuller	Vice President	105250	Y	2105

Figure 11-15 *SynchXfrOutput Data Viewer showing the derived columns*

31. Double-click the Script transformation component to open the editor and go to the Inputs and Outputs page. Add a new output and rename it as **AsyncXfrOutput**. Note that the SynchronousInputID is set to None by default (Figure 11-16). You've learned about this property in the previous part, but just to remind you here, this property decides whether the output will be of a synchronous type or will be an asynchronous output. The None value here is okay with us, as we want to add an asynchronous output.

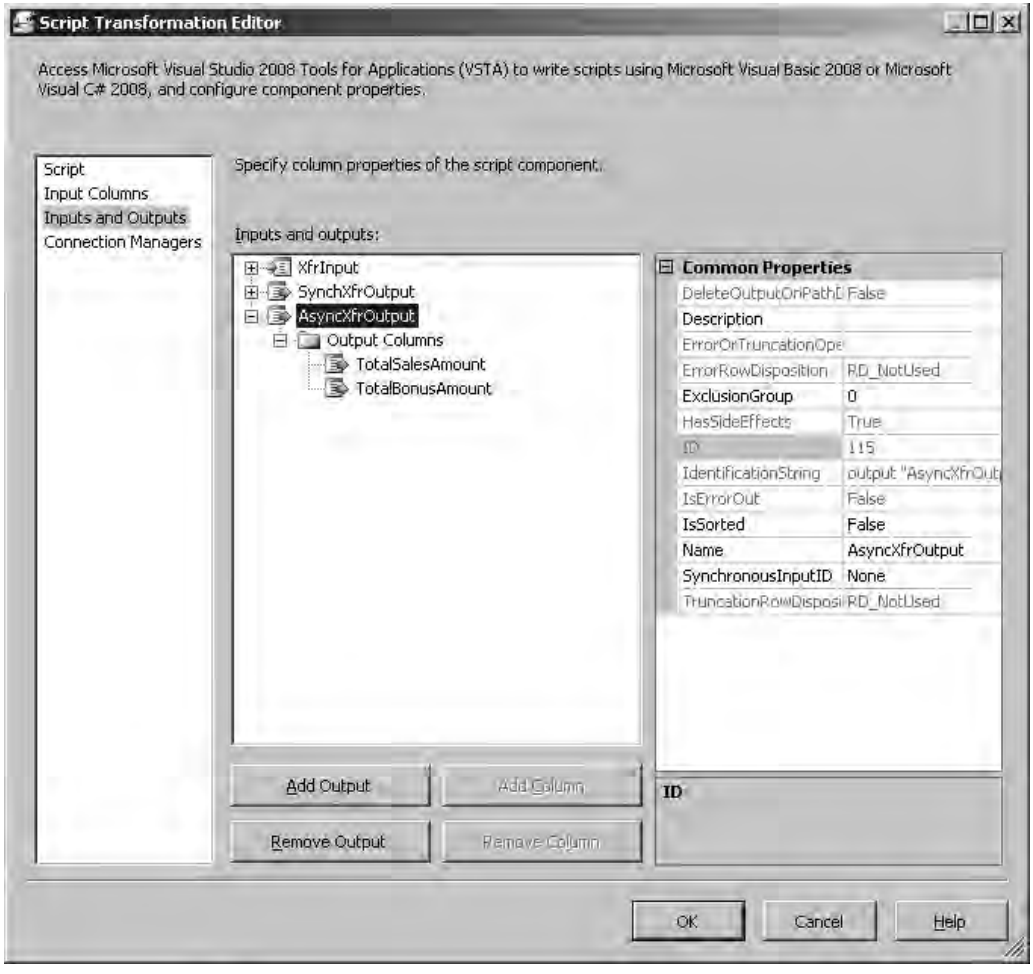


Figure 11-16 *Adding an Asynchronous Output to a Script transformation*

32. Click the Output Columns node and then add two columns for the asynchronous output as per the following details:

Column Name	Data Type
TotalSalesAmount	numeric [DT_NUMERIC]
TotalBonusAmount	numeric [DT_NUMERIC]

33. There is nothing more to configure in the metadata design mode. Let's go now to the code-design mode and click Edit Script to open the VSTA IDE. Let's see how the auto-generated code has behaved with the addition of a new asynchronous output.

    Open the BufferWrapper.vb project item. One thing you can immediately notice is that there are two buffer classes: the XfrInputBuffer class, as you had in the previous part, represents the synchronous output, while the other one, the AsyncXfrOutputBuffer class, represents the asynchronous output. There is still no buffer class for the SynchXfrOutput, as the synchronous output columns are contained in the XfrInputBuffer class. The new AsyncXfrOutputBuffer class contains two typed accessor properties for each column, one with the column name and the other one with <Column Name>_IsNull name to set the value of a column to null. Like the synchronous output class, it also has an AddRow method and a SetEndOfRowset method to mark the end of the rowset.

    Moving on to the ComponentWrapper.vb project item, it also differs from that of synchronous transformation. Here you can see the PrimeOutput method that was also used in the Script Source component. The PrimeOutput method prepares the outputs to accept new rows and is used for the outputs where you add new rows to the output buffers such as a source or a transformation with asynchronous output. This method calls CreateNewOutputRows to create a new row in our case. The ProcessInput method, which is repeatedly called for each row as it receives the buffers from the upstream component until the end of the rowset is reached, does a bit more work this time. It looks for the end of the rowset and lets the Data Flow task know that the outputs have finished. It uses Buffer .EndOfRowset to know that the current buffer is the final buffer in the rowset.

    When you look at the ScriptMain class in the Main.vb project item, you will realize that the code listed here has not changed. This further confirms that the code you add in this item is persisted and does not get affected with auto-generated code. The Main.vb item is auto-generated only the first time you click Edit Script, and the contents in the ScriptMain class depend upon the configurations you've done in the metadata design mode. In our case, obviously the code required for asynchronous output has not been generated in the ScriptMain class, as it already existed, so we will need to write the required code completely ourselves. But hey, that's not very difficult. We need to write the code for the following three steps:

First, we will add a new row to the asynchronous output in the CreateNewOutputRows method. In this method we can use the AddRow method to add a blank row as we did while writing the code for the source component.

Second, we will calculate the required values. As the XfrInput_ProcessInputRow method processes all the rows, we can add a bit more code here to aggregate the required values in this process.

Finally, we need to pass the calculated values to the output columns in the output buffer. However, we want to pass the aggregated values only when the end of rowset has been reached, so we will override the ProcessInput method. And that's it. Let's get started.

34. Write the following code as a separate subroutine in the ScriptMain class to create the CreateNewOutputRows method. We are using the AddRow method to add a blank row in the AsyncXfrOutputBuffer.

```
Public Overrides Sub CreateNewOutputRows()
    AsyncXfrOutputBuffer.AddRow()
End Sub
```

35. Next, to calculate total sales and total bonuses, we will need two script variables. So, write the following two lines at the top of the code after the Inherits UserComponent line.

```
Dim TotalSales As Decimal = 0
Dim TotalBonus As Decimal = 0
```

And then add the following two lines to calculate summary totals for these variables in the end of the XfrInput_ProcessInputRow method, but before the End Sub statement. Refer to Figure 11-17 if in doubt.

```
TotalSales = TotalSales + Row.SalesAmount
TotalBonus = TotalBonus + Bonus
```

Note that in the TotalBonus calculation, the Bonus script variable has been used instead of Row.BonusAmount. This is because Row.BonusAmount is an output column and the output columns are created as write-only columns in the Script component object model; hence, you cannot read values from output columns. If you need to reuse the calculated values in your script, it is better to use an intermediate script variable than to pass the value directly to the output column.

36. Last, write the following lines as a separate subroutine in the ScriptMain class:

```
Public Overrides Sub XfrInput_ProcessInput(ByVal Buffer As XfrInputBuffer)
    While Buffer.NextRow
        XfrInput_ProcessInputRow(Buffer)
    End While

    If Buffer.EndOfRowset Then
        AsyncXfrOutputBuffer.TotalSalesAmount = TotalSales
        AsyncXfrOutputBuffer.TotalBonusAmount = TotalBonus
        AsyncXfrOutputBuffer.SetEndOfRowset()
    End If
End Sub
```

```
main.vb*
ScriptMain                                  PostExecute()
<Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute> _
<CLSCompliant(False)> _
Public Class ScriptMain
    Inherits UserComponent

    Dim TotalSales As Decimal = 0
    Dim TotalBonus As Decimal = 0
    Dim BonusPaid As String
    Dim Bonus As Decimal = 0
    Public Overrides Sub PreExecute()
        MyBase.PreExecute()
    End Sub
    Public Overrides Sub CreateNewOutputRows()
        AsyncXfrOutputBuffer.AddRow()
    End Sub
    Public Overrides Sub XfrInput_ProcessInput(ByVal Buffer As XfrInputBuffer)
        While Buffer.NextRow
            XfrInput_ProcessInputRow(Buffer)
        End While

        If Buffer.EndOfRowset Then
            AsyncXfrOutputBuffer.TotalSalesAmount = TotalSales
            AsyncXfrOutputBuffer.TotalBonusAmount = TotalBonus
            AsyncXfrOutputBuffer.SetEndOfRowset()
        End If
    End Sub
    Public Overrides Sub XfrInput_ProcessInputRow(ByVal Row As XfrInputBuffer)

        Dim Bonusfactor As Integer = Me.Variables.varBonusMultiplier
        If (Row.Title = "Sales Representative" And Row.SalesAmount > 10000) Then
            BonusPaid = "Y"
            Bonus = Row.SalesAmount * Bonusfactor / 100
        ElseIf (Row.Title = "Sales Manager" And Row.SalesAmount > 50000) Then
            BonusPaid = "Y"
            Bonus = Row.SalesAmount * Bonusfactor / 100
        ElseIf (Row.Title = "Vice President" And Row.SalesAmount > 100000) Then
            BonusPaid = "Y"
            Bonus = Row.SalesAmount * Bonusfactor / 100
        Else : BonusPaid = "N"
            Bonus = 0
        End If

        Row.IsBonusPaid = BonusPaid
        Row.BonusAmount = Bonus
        TotalSales = TotalSales + Row.SalesAmount
        TotalBonus = TotalBonus + Bonus
    End Sub
    Public Overrides Sub PostExecute()
        MyBase.PostExecute()
    End Sub
End Class
```

Figure 11-17 *Code for adding an Asynchronous Output to the Script transformation*

As you can see, the overridden ProcessInput method calls the ProcessInputRow method to process all the rows for synchronous output and calculate totals for sales and bonus to be used in the asynchronous output. Once the end of rowset is reached, the aggregated total sales and total bonuses are written to the appropriate output columns.

37. You can test the asynchronous output by adding a new Row Count transformation into the data flow. Connect the AsyncXfrOutput to the Row Count 1 transformation using the data flow path. Add a data viewer on this new path as well as shown in Figure 11-18. You can use the varRecords variable again in this transformation to complete the configuration of Row Count 1 transformation.
38. Execute the package and the TotalSalesAmount and TotalBonusAmount values will be shown in the AsyncXfrOutput Data Viewer.

Script Component as a Destination

You are at the last part of the exercise, where you will configure the Script component as a destination. Here we will write the two outputs—synchronous output and asynchronous output—into the separate text files. We will write the synchronous

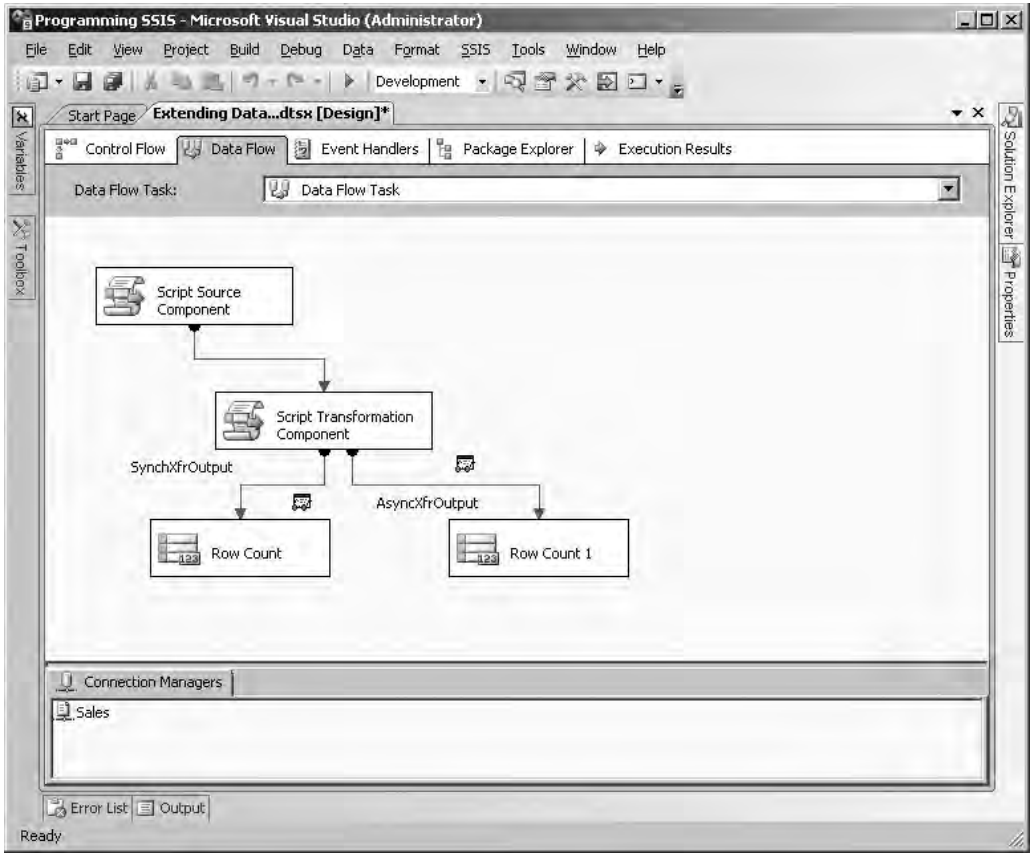


Figure 11-18 *Data flow layout for Script component package*

output to a comma-delimited text file and the asynchronous output to a nonstandard text file. In both these cases, configuring the Script component as a destination is not much different than configuring it as a source, though there is one very obvious but minor difference; we will use StreamWriter method to write into the text file instead of StreamReader to read from the text file. You will find the rest of the code quite similar to what you used in the script source.

Comma-Delimited Text File Destination

Here, you will write the synchronous output to a comma-delimited text file.

39. Delete both the row count transformations. Drop a Script component below the Script transformation component and choose Destination in the Select Script Component Type dialog box. Rename it as **Script Destination Component**.
40. Join the Script transformation to the Script Destination component. As you join them, an Input Output Selection dialog box will open. Choose SynchXfrOutput in the Output field and click OK.
41. In this step, we will add a connection to the destination file that we will need to use in our script. Right-click in the Connection Managers area and choose the New File Connection option. In the File Connection Manager Editor, select the Create File option in the Usage Type field, type **C:\SSIS\RawFiles\SalesWithBonus.txt** in the File field, and click OK to create the File Connection Manager.
42. Double-click the Script Destination Component to open the editor and change the ScriptLanguage to Microsoft Visual Basic 2008.
43. Go to Input Columns page and select all the columns. Now go to Inputs and Outputs page and rename the Input 0 to **DstSynchInput**. Expand it to see all the columns listed that you selected in the Input columns.
44. Go to Connection Managers page and click Add. Select SalesWithBonus.txt in the Connection Manager field and type **SalesWithBonus** in the Name field. The name specified here will be used in our script to refer to this connection manager.
45. Click the Edit Script button in the Script page to open the VSTA IDE. I would suggest you to spend few minutes to see the auto-generated code. The auto-generated code in both the BufferWrapper.vb and the ComponentWrapper.vb project items is familiar and very much expected, and there are no surprises.
46. First of all, as we are going to use StreamWriter to write into a text file, we will need to add an IO namespace. Add the following line in the Imports statements.

    ```
    Imports System.IO
    ```

47. Then type the following lines of code to declare variables required for the connection string, the StreamWriter instance, and the column delimiter, for which we are using a comma.

```
Dim cmSalesWithBonus As String
Private textWriter As StreamWriter
Private columnDelimiter As String = ","
```

48. Next add an acquire connection method to establish a connection to the SalesWithBonus Connection Manager. The code is similar to what you've already used in the Script Source component.

```
Public Overrides Sub AcquireConnections(ByVal Transaction As Object)
    Dim connMgr As IDTSConnectionManager100 = Me.Connections.SalesWithBonus
    cmSalesWithBonus = CType(connMgr.AcquireConnection(Nothing), String)
End Sub
```

49. In this step, we will perform one-time operations to write data to the text file in the PreExecute phase. We have two items that need to be implemented one time, one to initialize the StreamWriter and the other to write column headings in the file. Type the following lines in the PreExecute subroutine:

```
textWriter = New StreamWriter(cmSalesWithBonus, False)
With textWriter
    .Write("FirstName")
   .Write(columnDelimiter)
    .Write("LastName")
    .Write(columnDelimiter)
    .Write("Title")
    .Write(columnDelimiter)
    .Write("SalesAmount")
    .Write(columnDelimiter)
    .Write("IsBonusPaid")
    .Write(columnDelimiter)
    .Write("BonusAmount")
    .WriteLine()
End With
```

The first line initializes the textWriter to open a connection to the SalesWithBonus .txt file. The false argument lets the textWriter overwrite the contents in the file each time it is invoked. The remaining lines write the column headings in the text file using the Write and WriteLine methods.

50. As we have instantiated the StreamWriter, we must close it. We will close the StreamWriter in the PostExecute method.

```
textWriter.Close()
```

51. Now type the following code into the main DstSynchInput_ProcessInputRow method that will be repeatedly called for each row.

```
With textWriter
    If Not Row.FirstName_IsNull Then
        .Write(Row.FirstName)
    End If
    .Write(columnDelimiter)
```

```
        If Not Row.LastName_IsNull Then
            .Write(Row.LastName)
        End If
        .Write(columnDelimiter)
        If Not Row.Title_IsNull Then
            .Write(Row.Title)
        End If
        .Write(columnDelimiter)
        If Not Row.SalesAmount_IsNull Then
            .Write(Row.SalesAmount)
        End If
        .Write(columnDelimiter)
        If Not Row.IsBonusPaid_IsNull Then
            .Write(Row.IsBonusPaid)
        End If
        .Write(columnDelimiter)
        If Not Row.BonusAmount_IsNull Then
            .Write(Row.BonusAmount)
        End If
        .WriteLine()
    End With
```

Close the VSTA IDE and the Script Transformation Editor. We will now move on to create a script destination to write into a nonstandard text file.

Nonstandard Text File Destination

In this part you will write the asynchronous output to a nonstandard text file. Considering that you can now do some of the configurations yourself, this section is kept short and quick.

52. Add a File Connection Manager to create a SalesSummary.txt file in the C:\SSIS\RawFiles folder.
53. Add a Script destination component, rename it as **Script Non Standard Destination**, and join it with the asynchronous output from the Script Transformation component.
54. Perform the following in the editor:
    - Change the language.
    - Select both the input columns.
    - Rename the Input 0 to **DstAsyncInput**.
    - Add the SalesSummary.txt Connection Manager and name it as **SalesSummary**.

55. In the Code-design mode, perform the following:
    - Add the System.IO namespace in the Imports section.
    - Create two script variables:

      ```
      Dim cmSalesSummary As String
      Private textWriter As StreamWriter
      ```

    - Add the following code for the AcquireConnections method:

      ```
      Public Overrides Sub AcquireConnections(ByVal Transaction As Object)
          Dim connMgr As IDTSConnectionManager100 = Me.Connections.SalesSummary
          cmSalesSummary = CType(connMgr.AcquireConnection(Nothing), String)
      End Sub
      ```

56. Initialize the textWriter in the PreExecute method:

    ```
    textWriter = New StreamWriter(cmSalesSummary, False)
    ```

57. Close the textWriter in the PostExecute Method:

    ```
    textWriter.Close()
    ```

58. Last, type the following code in the DstAsyncInput_ProcessInputRow method:

    ```
    With textWriter
        .Write("This is a Summary Report for Sales")
        .WriteLine()
        .WriteLine()
        .Write("Total Sales Amount: " + Row.TotalSalesAmount.ToString)
        .WriteLine()
        .Write("Total Bonus Amount: " + Row.TotalBonusAmount.ToString)
        .WriteLine()
    End With
    ```

59. Execute the package and check out the output files. You will see the data from synchronous output written into a comma-delimited text file and the data from asynchronous output written into a nonstandard text file (Figure 11-19).

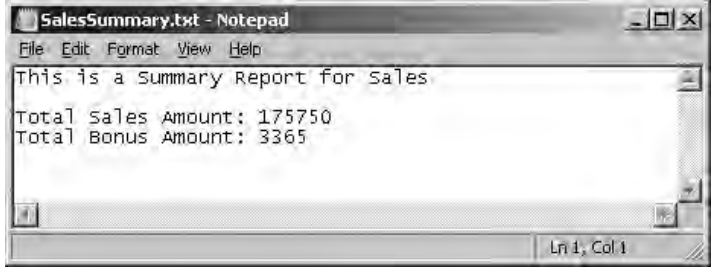


Figure 11-19 *A nonstandard text file written with the data from asynchronous output*

Debugging Techniques for Script Component

In the previous parts, you've used a combination of Row Count transformation and the data viewers to break the data flow and inspect the records visually. While this technique is very helpful in debugging and identifying the component that is causing issues, it fails if you want to know what's happening within your own developed component. You've used breakpoints within your scripts while working with the Script task to step through the code to isolate the error causing rows, but unfortunately the Script component does not support breakpoints. So, in a Script component you've to use alternate techniques to capture the information within your code. SSIS provides the following methods to capture information from within your script.

- **Raise informational messages** You can raise messages from within the script using the MessageBox.Show, Console.Write, or Console.WriteLine methods. You will be using these methods during development when you want to find out what rows or values are causing errors. For example, when you use the MessageBox.Show method, the output will be displayed in a modal message box, which you have to click to proceed further. You won't be using this method for the normal run; otherwise, you will have to keep clicking through the pipeline to progress the package execution. You can test your package by adding the following line in the Script Transformation component in the XfrInput_ProcessInputRow method just before the line where you pass the BonusPaid value to the Row.IsBonusPaid column (refer to Figure 11-20). This will show you the Bonus for each row, and you will have to click OK to progress to the next row.

```
MessageBox.Show("BonusAmount: " + Bonus.ToString)
```

- **Raise events** You can raise various events to let the package know about some information, warnings, or errors raised within the script. The Script component provides the following methods that you use to raise different events.

Event Method	Description
FireCustomEvent	Fires a user-defined custom event—e.g., you may want to capture certain data-related attributes.
FireError	Fires an error event.
FireInformation	Fires an informational message—e.g., information for auditing purposes.
FireProgress	Fires the on-progress event.
FireWarning	Fires a warning that is less critical than an error—e.g., a warning for a varchar(50) column is being mapped to varchar(10), which will work as long as the data doesn't exceed ten characters.

```
Public Overrides Sub XfrInput_ProcessInputRow(ByVal Row As XfrInputBuffer)

    Dim Bonusfactor As Integer = Me.Variables.varBonusMultiplier
    If (Row.Title = "Sales Representative" And Row.SalesAmount > 10000) Then
        BonusPaid = "Y"
        Bonus = Row.SalesAmount * Bonusfactor / 100
    ElseIf (Row.Title = "Sales Manager" And Row.SalesAmount > 50000) Then
        BonusPaid = "Y"
        Bonus = Row.SalesAmount * Bonusfactor / 100
    ElseIf (Row.Title = "Vice President" And Row.SalesAmount > 100000) Then
        BonusPaid = "Y"
        Bonus = Row.SalesAmount * Bonusfactor / 100
    Else : BonusPaid = "N"
        Bonus = 0
    End If

    MessageBox.Show("BonusAmount: " + Bonus.ToString)

    If Bonus > 1000 Then
        myMetaData = Me.ComponentMetaData
        myMetaData.FireInformation(0, "Script Transformation Component", "Bonus paid: " + Bonus.
    End If

    Row.IsBonusPaid = BonusPaid
    Row.BonusAmount = Bonus
    TotalSales = TotalSales + Row.SalesAmount
    TotalBonus = TotalBonus + Bonus
End Sub
```

Figure 11-20 *Raising events in a Script component*

As you know from the Script task, these methods were exposed by the event property of the Dts global object, but the Script component doesn't have any such object and though the event-firing methods have the same names, they are the methods of the IDTSComponentMetaData100 interface exposed by the ComponentMetaData property of the ScriptMain class. This also means the usage will be different than what you used in the Script task. Once the events have been raised by the Script component, the package event handlers then manage the event notifications. You can build your own event handlers to respond to a certain event—e.g., you can change the course of a package work flow in response to an event raised by the Script component.

To raise an event, you need to declare a variable for this interface as

```
Dim myMetadata as IDTSComponentMetaData100
```

and then add the following lines in the code where you earlier put the MessageBox (Figure 11-20).

```
If Bonus > 1000 Then
     myMetaData = Me.ComponentMetaData
     myMetaData.FireInformation(0, "Script Transformation
Component", "Bonus paid: " + Bonus.ToString, String.Empty, 0, False)
End If
```

This will raise informational messages to tell the package the amount of bonuses paid whenever the bonus is more than 1000. You can see these messages displayed in the output window and in the Progress tab. Refer to Books Online to know how different methods can be written and how they print their messages.

- **Log information** Much as when raising events, you can log detailed information about the execution progress, results, and errors. You can also raise events first and then log those events later. The main difference between first raising events and then logging compared to direct-logging to a log provider is that the direct logging method doesn't provide you an opportunity to build event handlers to respond to the events. If you want to use a log directly without using any other log event, then you must use the ScriptComponentLogEntry event while configuring logging for the Data Flow task as shown in Figure 11-21. The ScriptComponentLogEntry

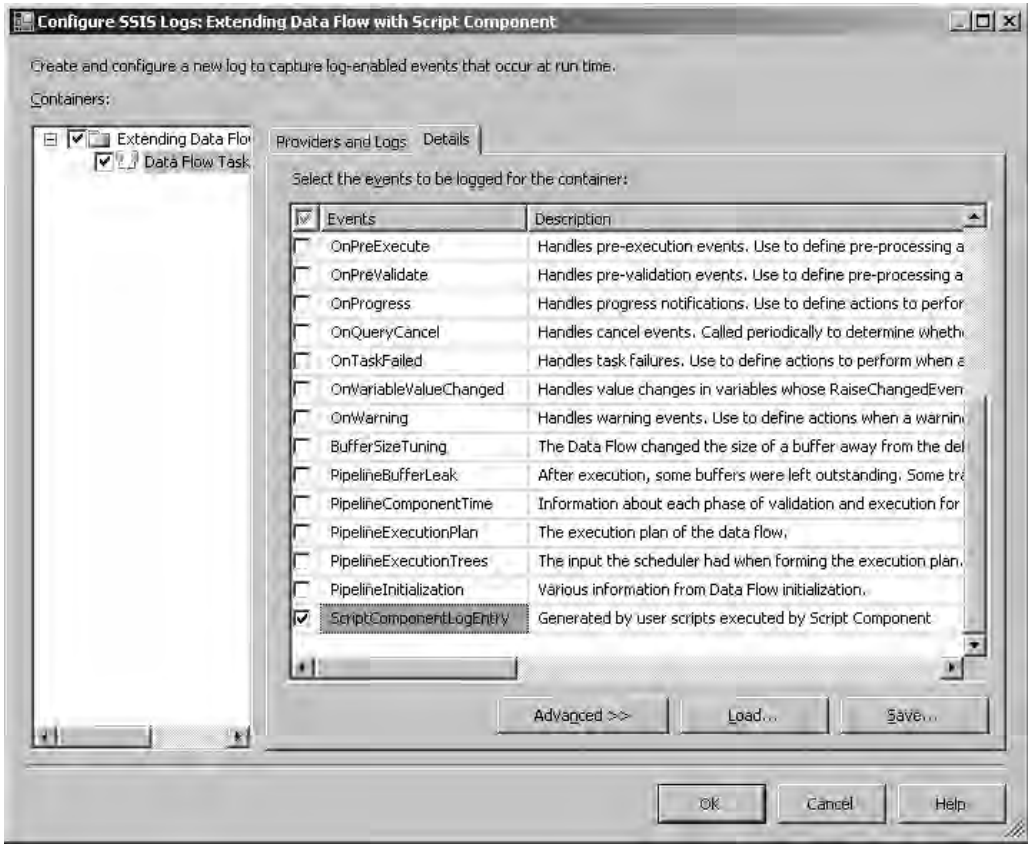


Figure 11-21 *Custom log entry for the Script component*

event gets added to the Data Flow task event in the Log Details tab when you add a Script component into the Data Flow task. After selecting this log entry, use the Log method within the ScriptMain class to log directly to the log providers enabled in the package. For example, if you add the following code in the Script Transformation component instead of the information event, the package will log the lines where the bonus is more than 1000:

```
Dim bt(0) As Byte
 If Bonus > 1000 Then
    Me.Log(Bonus.ToString, 0, bt)
 End If
```

Review

You've used the Script component to configure a source, a transformation, and a destination in this exercise. You've also learned the different types of outputs—i.e., synchronous and asynchronous outputs—that you can configure for a Script Transformation component. The general steps you followed to create any of these components is to configure the component first in the metadata design mode and then in the code design mode to write custom code for the processing rules. Do not underestimate the power of metadata design mode, as the settings you use in this mode are used to create code for you in the three project items—BufferWrapper.vb, ComponentWrapper.vb, and main.vb. While the code is auto-generated for both the BufferWrapper.vb and ComponentWrapper.vb items whenever you make a change, this does not happen for the main.vb project item. The code for the main.vb project item is auto-generated only once, when you invoke the code design mode for the first time. Though you can always write the code to fix anything that has been missed by the auto-generated code, it will be far easier and effective to let the auto-generation engine work for you, especially during the initial days of your working with the Script component. Also, make sure you understand the differences between a Script task and the Script component, especially how the different properties and methods are exposed within both the objects. For example, the variables and the connections are exposed by the Dts object in the Script task, while they are exposed using strongly typed accessor properties within the Script component. Last, the debugging techniques are different in both the objects—while you can use breakpoints to debug a Script task, you've to use alternate methods to debug a Script component such as MessageBoxes, data viewers, and events.

Summary

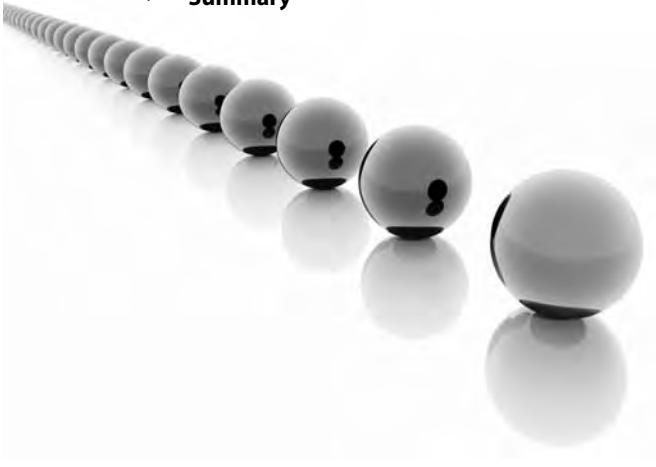
You've worked with the object model of Integration Services in this chapter and learned the different options it provides to extend an SSIS package. Based on your problem and the skills you have in-house, you can choose an appropriate method from scripting, developing custom objects, or programming packages from scratch. Later you worked with the Script task and the Script component and understood the various methods and properties provided by their object models. You also learned that though you have to write code differently for both the objects to use their methods, they provide the same functions, such as access to package variables or connection managers and the ability to raise events and log information. There are many samples contained in the Books Online that can enhance your understanding about the workings of SSIS object model. Also, check out the CodePlex site as well to see some rich programming content to hone your programming skills.

Chapter 12

Data Warehousing and SQL Server 2008 Enhancements

In This Chapter

- The Need for a Data Warehouse
- Data Warehouse Design Approaches
- Data Warehouse Architecture
- Data Warehouse Data Models
- Loading a Dimension Using a Slowly Changing Dimension
- Data Model Schema
- Building a Star Schema
- SQL Server 2008 R2 Features and Enhancements
- SQL Server 2008 R2 Data Warehouse Editions
- SQL Server 2008 R2 Data Warehouse Solutions
- SQL Server 2008 R2 Data Warehouse Enhancements
- Summary



Now that you have gone through all the development chapters on Integration Services, it is time to take a slight digression to understand the facilities that are available outside SSIS that can be very helpful in developing a well-performing data operations and management system. Soon after developers start using an ETL tool such as Integration Services, they will find themselves working on some operation of a data mart or a data warehouse. It is therefore necessary that you understand the data warehousing and the processes involved along with the tools and utilities that are available in SQL Server 2008, which can be exploited to develop data warehousing processes. You can use the tools and utilities provided in SQL Server 2008 to enhance an SSIS package or replace some of the package functionality so that overall solution performs well. This chapter is primarily split in two parts, with first part focusing on data warehousing and the second part on SQL Server 2008 database engine enhancements, such as change data capture (CDC), backup compression, and the MERGE statement, along with two new editions specially targeted to data warehouses. The purpose of this chapter is not to cover how to use or configure the new functionalities, as it would be too great a deviation from the subject of this book, but to know how you can use them alongside SSIS to best utilize the toolset you have. However, if I succeed in my little effort to induce enough interest in you in data warehousing, there are tons of books and other online material available that I would recommend for you to check out.

The Need for a Data Warehouse

As a business starts, it needs to capture information and record transactions to support business needs. The applications that help to serve this need are called line of business (LOB) applications. The systems used to support line of business applications are referred to as online transactional processing systems. OLTP systems are designed to respond very quickly for accessing and recording small chunks of information. Generally, lots of users are connected to such systems performing transactions to support activities such as sales, billing, and order management. Technically, OLTP systems are highly normalized to capture the current state of information and not to keep the historical data. As the business grows, the business managers need some reporting out of the information systems to take business forward, for example, how many sales yesterday or the how many deliveries completed. These kinds of reports are called operational reporting and to some extent can be served from OLTP systems. However, when the business has grown enough and is in such a good position that the OLTP systems are running busy most of the time, you can't run reports on them. Running reports touches more data and requires more resources that affect the performance of already-busy OLTP systems and can result in lost sales. So, the IT extracts data from OLTP systems into new systems so as to serve reports. As the report recipients generally do not want to

look at detailed information related to individual transactions, it makes sense to extract data to only the lowest level of granularity required.

Think of a web-based business that allows customers to buy products online. The underlying OLTP system keeps information about all the activities of the customers that may not be important for creating business reports. For example, a customer might have added three similar types of products in the basket, but while checking out removed two of the products from the basket after looking at the delivery charges for each of the items. While this data might be interesting for purposes such as data mining or to understand customer behavior, it is not very interesting for the business manager, who is concerned about the sales of the day only. So, while taking out data from an OLTP system, data is cleaned at the individual sales level, removing unwanted details. This data is still the detailed data as per requirements of the business manager and is kept more or less in the same format as that of the source OLTP system. Furthermore, this data might be combined with data from other source systems to create a bigger picture. In our example, consider joining the sales information with the information coming from the customer repository system to know who is an active buyer. These systems that keep cleaned, subject-oriented information at a detailed level and collect this information in real- or near-real-time mode are called operational data stores, or ODS in short.

Once the business creates an ODS system, they look forward to get the right information at the right time to make well-informed decisions. However, an ODS is limited by its very definition and cannot answer the questions that are spread across the enterprise or are related to multiple departments. This is the beginning of the need to have a decision support system (DSS) within the organization. As an organization grows further where it becomes important to improve business performance and make strategically right decisions, more complex questions are being put to information systems. These complex questions generally contain comparisons with a previous period such as sales this quarter compared to sales in the same quarter last year. This creates a need to keep data over longer periods of time. Some businesses that fall under strict regulations—e.g., the financial services industry—may be required to keep data as long as five to ten years. Keeping data over longer periods is not just to comply with regulations, but also to get some answers that otherwise would not be possible. A business dependent on sales around Christmas day might ask the sales comparison over a week not with the last year, but with the year when Christmas was on the same day of the week (say Thursday), previously. Also, as the organization evolves and starts to sell multiple products across multiple zones and countries, another need arises to see the single version of truth. This will mean that the data from multiple source systems needs to be collected in a common place and business rules need to be applied to collate the data to create an information store that can answer the complex questions across countries, currencies, products, departments, and time. The operational reports created on top

of an ODS cannot provide answers to such complex questions because either the data is not available in the ODS systems or the data model doesn't support such analysis. This results into creation of a data warehouse or a decision support system. A data warehouse or a DSS collects data from OLTP or ODS systems and might keep it in multiple forms that is, in its most granular form and in aggregated form. A data warehouse keeps data for longer periods of time (generally spread across several years) even after it has been deleted from the source systems.

Data Warehouse Design Approaches

Now we know a bit about a data warehouse: that it keeps years of history, that it keeps data in granular as well as aggregated format, and that the primary function of a data warehouse is to do business intelligence or analytical reporting. So the next question is how we design a data warehouse. There are two primary schools of thought—top-down and bottom-up approaches—and both are good and relevant to their own applications but also have associated risks involved.

Top-Down Design Approach

Bill Inmon, who is best known as the father of data warehousing, has defined this approach in which he suggests a data warehouse to be at the center of the Corporate Information Factory (CIF) designed using a normalized data model. The Corporate Information Factory approach takes the holistic view of the enterprise and its informational needs. In such a data warehouse, data is collected from most of the organization's operational systems and is held at the atomic level, that is, at the lowest level of detail. Further, the subject-oriented dimensional data marts containing aggregated data are built from the central atomic data warehouse to serve the departmental needs. As the data warehouse becomes a source system for all the analytical data marts and organizational reporting, this creates a consistent view—a single version of truth across the enterprise. It is easier to realign the data warehouse built with this approach to support business changes, and the data marts can be recreated easily from the central data warehouse. However, the downside of this approach is that the difficulty involved in building such a data warehouse, collating the entire enterprise data from almost all the operational data sources, makes this a very large project. So, implementing this project requires a high upfront investment, a large effort, and a long time to deliver. This long delivery time results in users losing patience and developing their own solutions, thus deviating from the original point of consolidation. The departments or business units generally don't buy into such projects with large delivery timescales.

Bottom-Up Design Approach

In the absence of a central data warehouse, when departments or business units need a data mart for a specific business process, they can't wait for the central data warehouse to be built but create their own departmental data mart. Ralph Kimball, who is a well-known author on data warehousing, is a proponent of this approach, for which he suggests creating business-specific dimensional-modeled data marts first. These data marts can contain atomic as well as aggregated data in a denormalized data model. These data marts are created in phases and are linked together via conformed dimensions. A strict discipline is adopted to implement conformed dimensions that are consistent with each other and use the same structure, attributes, and attribute values. Either these dimensions are exactly the same, including the keys, or one is a perfect subset of the other. The backbone of this approach is the conformed dimension's principle, and management of the conformed dimensions is fundamental to maintaining integrity of the data warehouse. Such a structure in which a data mart is linked to another data mart via a conformed dimension is called data warehouse bus architecture. Later these data marts are joined together to create an enterprise data warehouse. Because dimensional data marts are at the center of the bus architecture approach, the data model is simple to understand and use. The dimensional or star schema data model enables queries to retrieve data very efficiently. Implementing a data warehouse using this approach realizes business investment quite quickly—as soon as the data marts start functioning. The data warehouse is developed in phases and hence is more feasible than the central data warehouse approach. The downsides of choosing this approach are complicated data loading routines to maintain integrity within the data marts and the difficulty in modifying data warehouse structures when business changes occur.

Data Warehouse Architecture

Based on the previous two design approaches, you can have two different architecture layouts.

Centralized EDW with Dependent Data Marts

This architecture is based on Bill Inmon's Corporate Information Factory model. As the name implies, in this approach the data and applications reside on a central mainframe system. In this model, the data marts are fed exclusively from the centralized EDW in order to assure the single version of truth, thus creating a structure in which the data marts depend on the centralized EDW. Large enterprises use this approach, as it provides some key benefits that are vital to their success. However, there are instances

when this approach has failed to achieve desired results due to long deployment periods and huge upfront investments. The benefits and the issues involved are listed here:

- A single version of truth is the biggest reason why enterprises adopt this approach.
- Centralized data management makes it easy to apply corporate standards and practices and allow businesses to comply with the legal requirements; a recent example of compliance is the application of PCI DSS standards issued by the PCI Security Standards Council.
- With a holistic approach, the development of such a warehouse does not follow the general principle for phased-iterative development.
- Due to their implementation plans, centralized EDWs can become quite inflexible. When business units want to create analytical data marts, they get frustrated by this inflexibility and start moving away from this approach by creating federated data marts.
- Historically, the systems deployed in this architecture have been proprietary to a vendor that is slow to respond to fast improvements in technology and hence, they do not benefit from the advances in general-purpose computer systems.
- This is a high-risk approach, as implementing a centralized EDW is very expensive with a huge upfront investment that is realized only after the data warehouse project is completed and data marts are beginning to emerge. The maintenance and scalability costs have been found to be high in such an approach due to inflexibility. All these costs add up to a very high total cost of ownership.

Distributed Independent Data Marts

This architecture is based on the Ralph Kimball's Bus Architecture approach and is opposite to the monolithic design of a centralized EDW. In this architecture, distributed departmental data marts are created as and when the need arises. These data marts are designed independent of other data marts, and they get their data from operational source systems rather than centralized data warehouse. These data marts are highly relevant to their departments or the business units and are quick to build, as the scope is generally quite clear. However, due to the lack of control over their implementation, they often result in many versions of truth and are very difficult to keep consistent across the enterprise. The benefits and issues affecting this architecture are listed here:

- The data marts are easy and relatively quick to build. The cost to build one data mart seems low, as the return on investment starts pretty soon.

- Though the cost of developing a data mart seems low, the overall cost to develop a distributed data warehouse is not low. The cost is spread across several data marts and remains unnoticed in the overall expenditure.
- The data management across independent data marts is not easy. A lot of redundant data exists across many data marts.
- Many versions of truth exist across an enterprise, and a data mart does not align with another department's data mart, though the dimensions may align.

Data Warehouse Data Models

You can choose either of the two data modeling techniques–Entity-Relationship (ER) modeling and dimensional modeling—depending on your application scenario. The entity relationship model has been in use much longer than the relatively new dimensional model.

Entity-Relationship Model

The ER modeling is more popular in operational applications or OLTP systems, though it is also used in data warehousing as well. This is a requirement in the top-down approach, where a central data warehouse is built using the ER model. There are several graphical tools that help you to create an entity-relationship diagram (ERD) to conceptualize data. These tools primarily use three basic constructs: entity, relationship, and attribute.

- **Entity** An entity is a concept, real or abstract, about which information is collected. It represents a class of objects such as products, an object such as a car, or an event such as a sale that can be classified by their properties and characteristics. It will usually have a business definition and is defined uniquely using primary keys in the data model.
- **Relationship** A relationship is an association among entities in a model and indicates how two or more entities are related to one another. For example, a customer owns a car—this indicates the relationship, how the customer is related to the car. It is represented by a line drawn between entities. The entities are related to each other in differing cardinality, which defines the maximum number of instances of one entity related to a single instance of another entity. The relationship has one-to-one, one-to-many, and many-to-many cardinalities.
- **Attributes** Both entities and relationships have attributes that describe their characteristics or properties. For example, a car has a VIN number attribute.

The relational databases are designed using the normalized ER model to remove redundant data. Six normal forms have been defined to date, but a database is considered adequately normalized if it is in third normal form (3NF). Normalization is a systematic process for assigning attributes to entities to ensure that a database is integral by breaking down information to its smallest divisible parts and removing data duplication. It is an incremental process, which means that to be transformed into 3NF, the entities must first qualify for 2NF. The 1NF removes repetition in data by creating one-to-many relationships between master and detail entities. For example, you will remove repeating similar columns from a table into another table. 2NF takes removal of repeating data a further step by removing duplicate rows of data from a table into a separate table. 3NF removes the columns that are not dependent on the primary key and resolves many-to-many relationships into unique values.

As the normalization level increases, the data is further broken down and granularity of data is increased. Such highly granular data models are very efficient in returning small amounts of information or updating small amounts of data. This is required by OLTP systems that have many users working on small pieces of information. That's the reason the ER Model is highly successful in relational database applications. However, the requirements of a data warehouse are different. The queries are usually small in number, but they can perform huge I/O activity on the server. These requirements are met with the dimensional model.

Dimensional Model

Dimensional modeling is a relatively newer modeling technique than ER modeling. Recent trends in modeling preference favor dimensional data modeling, as it is simple to build, is easily to understand even by business folks, and aligns with the questions usually asked of a data warehouse. This model is very efficient at summarizing values and presenting data to analytical tools. This model keeps the numerical values called *facts* in one table, while the attributes that measure these values are grouped together in tables called *dimensions.* This structure makes it particularly suitable to answer business questions such as sales this quarter or the sales this quarter compared to sales the previous quarter.

Fact

The numeric values along with some contextual data are stored in fact tables. A data warehouse contains one or more subject-oriented fact tables. A *fact* typically represents an item, an event, or a business transaction such as sale of an item that can be used to perform business analysis. Further, a fact consists of some columns containing values and some foreign keys linking to dimensions. As the transactions are added to the fact

table, it can soon become quite large; consider that each transaction could represent a row in the fact table.

It is the granularity that can really affect the size of your fact data. You have to understand the business requirements completely—what they call for now and in the future—to decide the level of detail you want to keep in the fact data. You might choose to keep every transaction, or you might choose to keep aggregated data at the day level if business is never going to drill down to the transaction-level detail. Thus the lower the granularity level, the more the data and hence, the more disk space will be required. However, disk space should not be an issue, because data warehouses are meant to be large in size. Always be careful while choosing granularity, as a change in granularity means the whole data warehouse has to be reseeded.

Measure

A *measure* is closely related to a fact, and sometimes the terms are used interchangeably. A measure is what you want to measure, and the fact is a measure with context. So, a measure is a numeric value used to indicate the performance of the business, and the fact equates to a row in the Fact table. A measure is used in combination with dimension members, while the value is taken from facts. For example, TotalSales-by-year and SupplyCost-by-month are measures.

Dimension

A *dimension* contains the same type of information broken down in levels of interest. For example, a time dimension can contain year, quarter, month, and day levels. A dimension contains the information that a business wants to analyze the facts with. This information does not change very often. Typically, a dimension table is relatively quite small compared to a fact table. A data warehouse can have many dimensions attached to each fact table. Some of the common dimensions could be Time, Product, Employee, Location, and Customer. A dimension is made up of members and hierarchies.

Dimension Members

Member of a dimension are arranged in levels, for instance, members in a time dimension have different levels: day, week, month, quarter, and year. Similarly all cities, states, and countries are members of a geography dimension. While analyzing the data, you can choose any dimension member level and associated facts will be used in analysis automatically.

Dimension Hierarchies

The members of a dimension can be arranged in a hierarchical order with multiple levels to create dimension hierarchies. You can create more than one hierarchy and a dimension member can participate in more than one hierarchy. For example, you can

create two hierarchies for time dimension—one with Day, Month, Quarter, and Year as levels and the other one with Day and Week as levels.

Dimension Types

Dimensions can be designed in different ways to meet specific business functions and to enhance performance. Four types of configurations are covered here.

Conformed Dimension

At times you will use a dimension in more than one subject-oriented data marts. If you are keeping the dimensions exactly same or are sourcing them from the central data warehouse, obviously making them the same, then these dimensions are called *conformed* dimensions. A conformed dimension does not need to be exactly the same as the main dimension; it is still conformed to the main dimension if it is a subset of the detailed dimension. In this case, the attributes in a conformed dimension need to be labeled exactly in the same way as in a detailed dimension. The most common example could be a date dimension used across many data marts having same attributes such as date, month, quarter, and year.

Junk Dimension

The business data generally contains some attributes that are not related to any dimension but are associated more with the fact. These can be easily identified, as generally these attributes represent themselves in the form of indicators or flags. For example, in a car rental business flags such as IsDamaged, IsStolen, and IsExchanged are common in the databases. These flags are not exactly part of any dimension, but businesses do want to analyze data using these flags. If you leave them with the fact data, the performance of the queries will be very poor due to the large size of the fact table, and indexes won't help due to yes/no nature of such flags. You could put them in their own dimension, in which case you would have as many dimensions as there are such attributes. Very frequently you will see that the number of indicators or flags that exist in the data reaches 20 plus. In this case, your data mart design will be cluttered with lots of dimensions that have only one member, enough to confuse users. A recommended approach is to club all these nonrelated attributes and put them in a table, thus creating a *junk dimension.* For instance, you can select distinct combinations of the attributes and add them to the junk dimension where each distinct row is assigned a surrogate key that is referenced in the Fact table. Keeping the flags and indicators in one dimension makes it easier for users to find out these attributes and is useful in that the queries perform much better.

Degenerate Dimension

Another type of attribute in the data is the number associated with each business transaction, such as an invoice number or a ticket number. Though these attributes are not actively used in analysis, every now and then businesses do have a requirement to look for such attributes, as they link back with operational systems. These attributes are also very useful in reconciling the data warehouse with ODS systems. When the grain of the fact table is the same as that of a transaction, the likelihood of a *degenerate* dimension increases. As the fact grain is the same as that of the degenerate dimension, so sometimes it forms an integral part of the primary key of the fact table. A degenerate dimension should exist along with the facts in the fact table. There is no point in creating a separate dimension table for a degenerate dimension, as it will grow with the fact and will become quite large.

Role-Playing Dimensions

These types of dimensions exist when the same attribute is used multiple times. For instance, a fact row could contain SaleDate and DeliveryDate columns, both pointing to the date attribute of the date dimension. In this case, the key of the date attribute of the date dimension is used multiple times in the fact row and is often referred to as a role-playing dimension. You create a table alias or a view to use the date dimension foreign key in the fact table.

Loading a Dimension Using a Slowly Changing Dimension

Loading a static dimension is a simple one-off task, but you will come across dimensions that change with time and will find loading such a dimension a challenging task. Some of the data warehouse dimensions do not change that often. For instance, a date dimension's members stay as is and never change. By contrast, other business dimensions do change with time as a business evolves; for example, a product could change in size and volume. Typically, a row in the dimension table will have different attribute requirements:

- Some attributes never change, such as IDs, or in the case of a car, a VIN number.
- Some attributes will change but a business always want to see the current value; for instance, a business may not bother to see the old description of the product if the description of a product is changed. This type of change is called a Type 1 change in which the attribute value is overwritten and a business cannot go back and see the old value or learn when the change in value has been applied.

- Some attributes will change and the business will want to see the current value as well as the historical value. This happens when a business is interested in analyzing the facts before and after a particular change has been applied. For instance, a business might want to see the effect on sales when the size of a can of beans is changed from 400 g to 350 g. This type of change is called a Type 2 change.
- Most of the business requirements can be covered with use of the preceding types. Other types of changes have been defined such as Type 3 and so on; however, their primary function is to improve storage efficiency and query performance. These change types are not covered here. Refer to data warehousing books if you want to know more about them. Also, the SCD transformation in SSIS supports only up to Type 2 changes.

You have studied the Slowly Changing Dimension (SCD) transformation in Chapter 10 and have done a Hands-On exercise as well. Here just to recap: the SCD transformation is designed to help you load a dimension that changes in time, which is generally a challenge with an ETL tool. The SCD transformation supports the following attribute change types to support the previously mentioned requirements.

- **Fixed Attribute Change Type** This change type supports the attributes that do not change (Type 0) and align with the first scenario mentioned previously.
- **Changing Attribute Change Type** Using this change type, you can load the attribute values that are Type 1 changes, and it overwrites the existing values with the new values. This is an in-place modification.
- **Historic Attribute Change Type** In this change type, a new row is added that will be valid for the current or future transactions. Typically, three columns are used to handle this type of change—StartDate, EndDate, and IsActive. When a row is getting a Type 2 update, it will update the IsActive flag to 'N' and will timestamp the EndDate with the current date and time to indicate that this row is no longer active, while the activity period of this row can be found using StartDate and EndDate. Also, at the same time, a new row is added with same values in all the columns except in the Type 2 column that is getting the update. In this Type 2 column, the updated new value is inserted, StartDate gets the current date and time stamp, EndDate is kept as null, and the IsActive column gets a 'Y' value to indicate that this row is active for the particular member.

Among other methods, SCD transformation can be tested to load a dimension. If you think that your dimension is too huge and the SCD transformation is not a fit for the purpose, you can always create a script in your package to load a slowly changing dimension.

Data Model Schema

You can model your data model with two different types of schema.

Star Schema

As mentioned earlier, a data mart is a subject-oriented mini–data warehouse and will have one or few fact tables surrounded by a relatively large number of dimensions. When these are drawn on a piece of paper, the structure looks like a star, hence the term star schema. The star schema is a simple model in which the denormalized dimensions are connected to facts using foreign key relationships. This model has become a building block in dimensional modeling. In a large data warehouse where multiple fact entities can exist, you can imagine a multiple star schema model with each dimension connected with several fact entities. Figure 12-1 shows a simple example of a star schema data model.

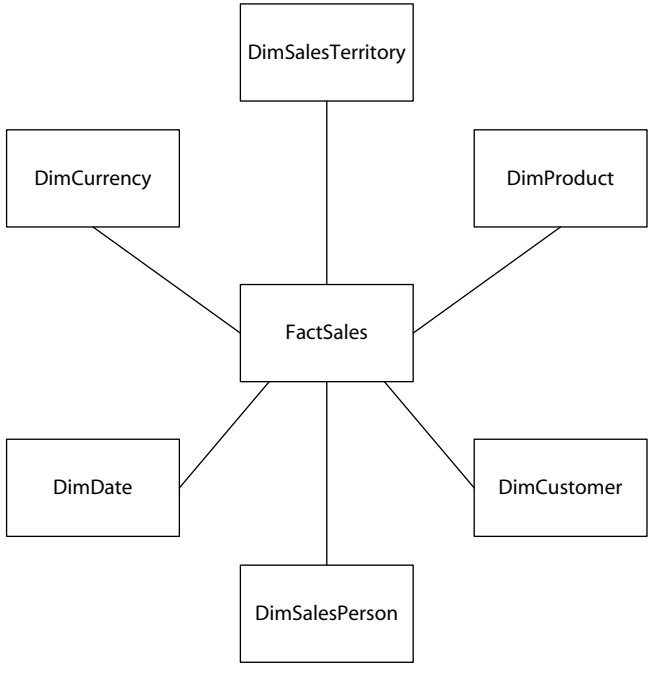


Figure 12-1 *A simple star schema data model*

In this model, the dimensions are denormalized and a business view of a dimension is represented as a single table in the model. Also, the dimensions have simple primary keys and the fact table has a set of foreign keys pointing to dimensions; the combination of these keys forms a compound primary key for the fact table. This structure provides some key benefits to this model by making it simple for users to understand. Writing select queries for this model is easier due to simple joins between dimensions and the fact tables. Also, the query performance is generally better due to the reduced number of joins compared to other models. Finally, a star schema can easily be extended by simply adding new dimensions as long as the fact entity holds the related data.

Snowflake Model

Sometimes it is difficult to denormalize a dimension, or in other words, it makes more sense to keep a dimension in a normalized form, especially when multiple levels of relationships exist in dimension data or the child member has multiple parents in a dimension. In this model, the dimension suitable for snowflaking is split into its hierarchies and results into multiple tables linked to each other via relationships, generally, one-to-many. The many-to-many relationship is also handled using a bridge table between the dimensions, sometimes called a FactLessFact table. For example, the AdventureWorksDW2008 products dimension DimProduct is a snowflake dimension that is linked to DimProduct SubCategory, which is further linked to the DimProductCategory table (refer to Figure 12-2). This structure makes much sense to database developers or data modelers and helps users to write useful queries, especially when an OLAP tool such as SSAS supports such a structure and optimizes running of snowflaked queries. However, business users might find it a bit difficult to work with and would prefer a star schema, so you have to find a balance in choosing when to go for a snowflake schema. Though there is some space saving by breaking a dimension into a snowflake schema, it is not high on the preference list because first, the space is not very costly, and second, the dimension tables are not huge and space savings are not expected to be considerable in many cases. The snowflaking is done for the functional reasons rather than savings in disk space. Finally, the queries written against a snowflake schema tend to use more joins (because more dimension tables are involved) compared to a star schema, and this will affect the query performance. You need to test for user acceptance for the speed at which results are being returned.

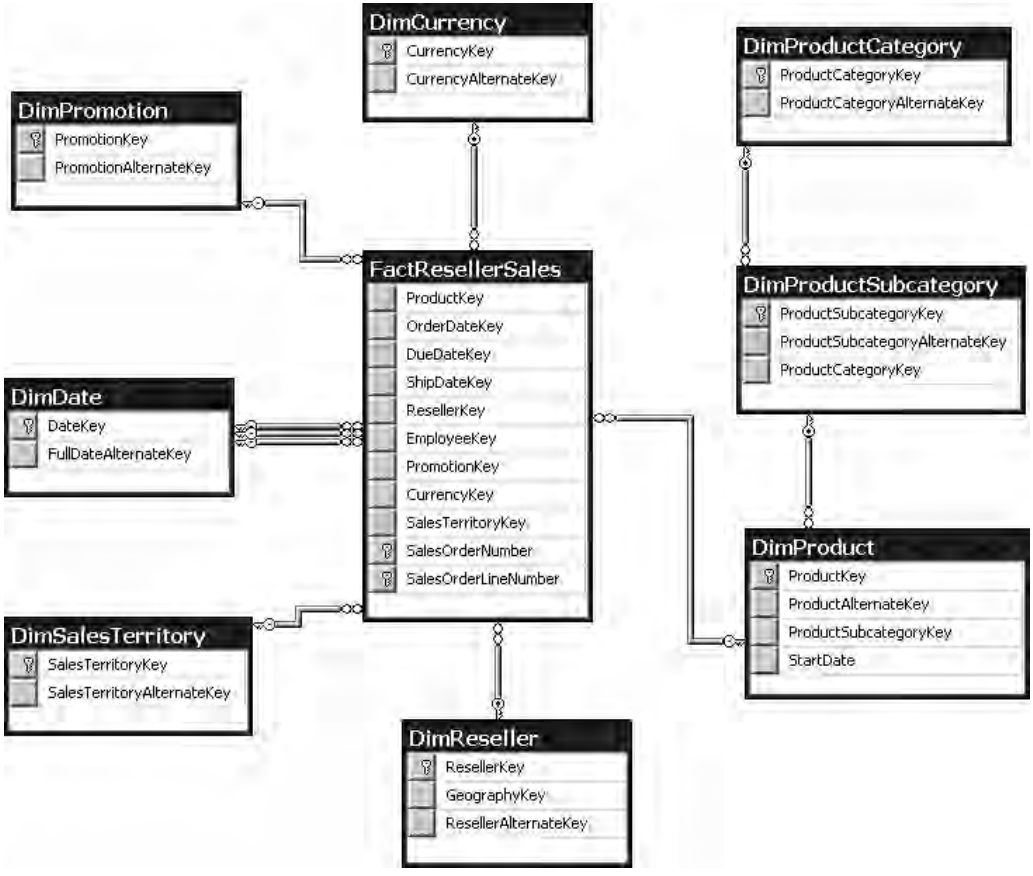


Figure 12-2 *AdventureWorksDW2008 simplified snowflake schema*

Building a Star Schema

The very first requirement in designing a data warehouse is your focus on the subject area for which a business has engaged you to build a star schema. It's easy to sway in different directions while building a data warehouse, but whatever you do later in the process, you must always keep a focus on the business value you are delivering with the model. As a first step, capture all the business requirements and the purposes for which they want to do this activity. You might end up meeting several business managers and process owners to understand the requirements and match them against the data available in source systems. At this stage, you will be creating a high-level data source mappings document to meet the requirements. Once you have identified at a high

level that the requirements can be met with the available data, the next step is to define the dimensions and the measures in the process. While defining measures or facts, it is important that you discuss with the business and understand clearly the level of detail they will be interested in. This will decide the grain of your fact data. Typically, you would want to keep lowest grain of data so that you have maximum flexibility for future changes. In fact, defining the grain is one of the first steps while building a data warehouse, as this is the cornerstone to collating the required information, for instance, defining the roll-up measures.

At this stage, you are ready to create a low-level star schema data model and will be defining attributes and fields for the dimension and fact tables. As part of the process, you will also define primary keys for dimension tables that will also exist in fact tables as a foreign key. These primary keys will need to be the new set of keys known as the surrogate keys. The use of surrogate keys instead of source system keys or business keys provides many benefits in the design; for instance, they provide protection against changes in source systems keys, maintain history (by using SCD transformation), integrate data from multiple source, and handle late arriving members, including facts for which dimension members are missing. Generally, a surrogate key will be an auto-incrementing non-null integer value like an identity and will form a clustered index on the dimension. However, the Date dimension is a special dimension, commonly used in the data warehouses, with a primary key based on the date instead of a continuous number; for instance, 20091231 and 20100101 are date-based consecutive values, but not in a serial number order.

While working on dimensions, you will need to identify some special dimensions. First, look for role-playing dimensions. Refer to Figure 12-2 and note that the DateKey of the DimDate dimension is connected multiple times with the fact table, once each for the OrderDateKey, DueDateKey, and ShipDateKey columns. In this case, DimDate is acting as a role-playing dimension. Another case is to figure out degenerate dimensions in your data and place them alongside facts in the fact table. Next, you need to identify any indicators or flags used in the facts that are of low cardinality and can be grouped together in one junk dimension table. These miscellaneous flags held together at one place make the user's life much easier when they need to classify the analysis by some indicators or flags. Finally, you will complete the exercise by listing the attributes' change types, that is, whether they are Type 1 or Type 2 candidates. These attributes, especially Type 2, help in maintaining history in the data warehouse. Keeping history using SCD transformations has been discussed earlier in the chapter as well as in Chapter 10. Though your journey to implement a data warehouse still has a long way to go, after implementing the preceding measures, you will have implemented a basic star schema structure. From here it will be easy to work with and extend this model further to meets specific business process requirements, such as real-time data delivery or snowflaking.

SQL Server 2008 R2 Features and Enhancements

Several features have been provided in SQL Server 2008 R2 that can help you in your data warehousing project by either providing new functionality or improving the performance of commonly used features. It is not possible to cover them all here unless I want to stretch myself beyond the scope of this book, especially when those features do not reside in the realm of Integration Services. But I will still try to cover some features that most data warehousing projects will use, while other features such as data compression, sparse columns, new data types, Large UDTs, and minimal logging are not covered.

SQL Server 2008 R2 Data Warehouse Editions

Microsoft has just launched SQL Server 2008 R2, which is built upon the strong foundations and successes of SQL Server 2008 and is targeted at very large-scale data warehouses, and at higher mission-critical-scale and self-service business intelligence. SQL Server 2008 R2 has introduced two new premium editions to meet the demands of large-scale data warehouses.

SQL Server 2008 R2 Datacenter

This edition is built on the Enterprise Edition code base, but provides the highest levels of scalability and manageability. SQL Server 2008 R2 Datacenter is designed for the highest levels of scalability that SQL Server platform can provide, virtualization, and consolidation, and it delivers a high-performing data platform. Typical implementations include a large-scale data warehouse server that can scale up to support tens of terabytes of data, provide Master Data services, and implement very large-scale BI applications such as self-service or power pivot for SharePoint. Following are the key features:

- As the Enterprise Edition is restricted to up to 25 editions and 4 virtual machines (VMs), the Datacenter Edition is the next level if you need more than 25 instances or more VMs. This also provides application and Multi-Server Management for enrolling and gaining insights.
- The Datacenter Edition has no limits on server maximum memory; rather, it is restricted by the limits of the operating system only. For example, it can support up to 2TB of RAM if running on the Windows Server 2008 R2 Datacenter Edition.

- It supports more than 8 processors and up to 256 logical processors for the highest levels of scale.
- It has the highest virtualization support for maximum ROI on consolidation and virtualization.
- It provides high-scale complex event processing with SQL Server StreamInsight.
- Advanced features such as the Resource Governor, data compression, and backup compression are included.

SQL Server 2008 R2 Parallel Data Warehouse

Since acquiring DATAllegro, a provider of large-volume, high-performance data warehouse appliances, Microsoft has been working on consolidating hardware and software solutions for high-end data warehousing under a project named Madison. The SQL Server 2008 R2 Parallel Data Warehouse is the result of Project Madison. The Parallel Data Warehouse is an appliance-based, highly scalable, highly reliable, and high-performance data warehouse solution. Using SQL Server 2008 on Windows Server 2008 in a massively parallel processing (MPP) configuration, Parallel Data Warehouse can scale from tens to hundreds of terabytes, providing better and more predictable performance, increased reliability, and lower cost per terabyte. It comes with preconfigured hardware and software that is carefully balanced in one appliance, thus making deployment quick and easy. Massively parallel processing enables it to perform ultra-fast loading and high-speed backups, thus addressing two of the major challenges facing modern data warehouses: data load and the backup times. You can integrate existing SQL Server 2008–based data marts or mini–data warehouses with parallel data warehouses via a hub-and-spoke architecture. This product was targeted to be shipped alongside the SQL Server 2008 R2 release; however, the release of this product has been slightly delayed awaiting customer feedback from the customer Technology Adoption Program (TAP). The parallel data warehouse principles and architecture are detailed later in this chapter.

SQL Server 2008 R2 Data Warehouse Solutions

Microsoft has recognized the need to develop data warehouse solutions to build upon the successes of SQL Server 2008. This has resulted in Microsoft partnering with several industry-leading hardware vendors to create best-of-breed balanced configurations combining hardware and software to achieve the highest levels of performance. Two such solutions are now available under the names of Fast Track Data Warehouse and Parallel Data Warehouse.

Fast Track Data Warehouse

The Fast Track Data Warehouse solution implements a CPU core-balanced approach on the symmetric multiprocessor (SMP)–based SQL Server data warehouse using a core set of configurations and database best practice guidelines. The fast track reference architecture is a combination of hardware that is balanced for performance across all the components and software configurations, such as Windows OS configurations, SQL Server database layout, and indexing, along with a whole raft of other settings, best practices, and documents to implement all of these objectives. The Fast Track Data Warehouse servers can have two-, four-, or eight-processor configurations and can scale from 4 terabytes to 30-plus terabytes and even more if compression capabilities are used. The earlier available reference architectures are found to suffer from various performance issues due to the simple fact that they have not been specifically designed to suit the needs of one particular problem and hence, suffer from an unbalanced architecture. For example, you may have seen that a server is busier and processing too much I/O, but still the CPU utilization is not high enough to indicate the work load. This is a simple example of mismatch or unbalance existing between various components in currently available servers. The Fast Track Data Warehouse servers are built on the use cases of a particular scenario—i.e., they are built to capacity to match the required workload on the server rather than with a one-size-fits-all approach. This approach of designing a balanced server provides predictable performance and minimizes the risk of going over spec on the components such as by providing CPU or storage that will never be utilized. The predictable performance and scalability is achieved by adopting core principles, best practices, and methodologies, some of which have been listed next.

- *It is built for data warehouse workloads.* First of all, understand that the data warehouse workload is quite different from that on the OLTP servers. While OLTP transactions are made up of small read and write operations, data warehouse queries usually perform large read and write operations. The data warehouse queries are generally fewer in number, but they are more complex, requiring high aggregations, and generally have date range restrictions. Furthermore, OLTP transactions generate more random I/O, which causes slow response. To overcome this, a large number of disks have been traditionally used, along with some other optimization techniques such as building heavy indexes. These techniques have their own maintenance overheads and cause the data warehouse loading time to increase. The Fast Track Data Warehouse uses a new way of optimizing data warehouse workloads by laying data in a sequential architecture. Considering that a data warehouse workload requires ranges of data, reading sequential data off the disk drives is much efficient compared to random I/Os. All efforts are targeted to

preserving sequential storage. The data is preferred to be served from disk rather than from memory, as the performance achieved is much higher with sequential storage. This results in fewer indexes, yielding savings in maintenance, decreased loading time, lesser fragmentation of data, and reduced storage requirements.

- *It offers a holistic approach to component architecture.* The server is built with a balance across all components, starting with disks, disk controllers, fiber channels HBAs, and the Windows operating system, and ranging up to the SQL Server and then to the CPU core. For example, on the basis of how much data can be consumed per CPU core (200 MBps), the number of CPU cores is calculated for a given workload and then the backward calculations are applied to all the components to support the same bandwidth or capacity. This balance or the synchronization in response by individual components provides the required throughput to match the capabilities of the data warehouse application.
- *It is optimized for workload type.* The Fast Track Data Warehouse servers are designed and built considering the very nature of the database application. To capture these details, templates and tools are provided to design and build the fast track server. Several industry-leading vendors are participating in this program to provide you out-of-the-box performance reference architecture servers. Businesses can benefit from reduced hardware testing and tuning and rapid deployment.

Parallel Data Warehouse

When your data growth needs can no longer be satisfied with the scale-up approach of the Fast Track Data Warehouse, you can choose the scale-out approach of Parallel Data Warehouse, which has been built for very large data warehouse applications using Microsoft SQL Server 2008. The symmetric multiprocessing (SMP) architecture used in the Fast Track Data Warehouse is limited by the capacity of the components such as CPU, memory, and hard disk drives that form part of the single computer. This limitation is addressed by scaling out to a configuration consisting of multiple computing nodes that have dedicated resources—i.e., CPUs, memory, and hard disk space, along with an instance of SQL Server—connected in an MPP configuration.

Architecture and Hardware

The appliance hardware is built on industry-standard technology and is not proprietary to one manufacturer, so you can choose from well-known hardware vendors such as HP, Dell, IBM, EMC^2, and Bull. This way, you can keep your hardware maintenance costs low, as it will integrate nicely with already-existing infrastructure.

As mentioned earlier, the Parallel Data Warehouse is built on Microsoft Windows 2008 Enterprise server nodes with their own dedicated hardware connected via a high-speed fiber channel link, with each node running an instance of the SQL Server 2008 database server. The MPP appliance basically has one control node and several compute nodes, depending on the data requirements. This configuration is extendable from single-rack to multirack configurations; in the latter case, one rack could act as a control node. The nodes are connected in a configuration called Ultra Shared Nothing (refer to Figure 12-3), in which the large database tables are partitioned across multiple nodes to improve the query performance. This architecture has no single point of failure, and redundancy has been built in at all component levels.

Applications or users send requests to a control node that balances the requests intelligently across all the compute nodes. Each compute node processes the request it gets from the control node using its local resources and passes back the results to the control node, which then collates the results before returning to the requesting application or user. As the data is evenly distributed across multiple nodes and the nodes process the requests in parallel, queries run many times faster on an MPP appliance than on an SMP database server.

Like a Fast Track Data Warehouse server, an MPP appliance is also built under tight specifications and carefully balanced configurations to eliminate performance bottlenecks. Reference configurations have been designed for different use case scenarios taking into account different types of workloads such as data loading, reporting, and ad hoc queries. A control node automatically distributing workload evenly, compute nodes being able to work on queries autonomously, system resources balanced against each other, and design of reference configurations on use case

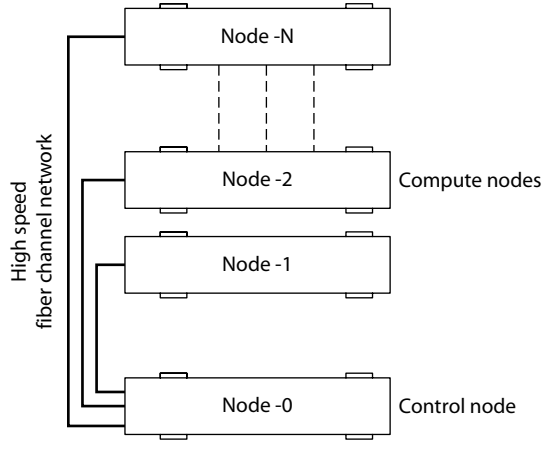


Figure 12-3 *Parallel Data Warehouse architecture*

scenarios enable an MPP appliance to achieve predictable performance. Scalability follows from here with the simple addition of capacity as the data volumes grow.

Hub-and-Spoke Architecture

Another important advantage with an MPP appliance is that it can be deployed in a hub-and-spoke architecture. In this way, you can use an MPP appliance as a hub while the spokes could be either MPP appliances or standard SQL Server 2008–based symmetric multiprocessing (SMP) servers (see Figure 12-4). Typically, department users will connect to spokes to access data in their required formats. In this configuration, the MPP appliance at the hub will host the enterprise data in the lowest granularity and the spokes will contain data for their relevant department in the schema and aggregations they require. This is possible because the spokes could host any database application such as the SQL Server 2008 data mart or SQL Server Analysis Services data mart, as best fits the user requirements. So, this architecture with an MPP appliance at the hub and SMP database servers or MPP appliances as spokes is a specialized configuration in which a grid of computers forms a very large-scale data warehouse in a federated model. This grid of computers can be connected via a high-speed network. Also, the nodes of an MPP appliance are connected via a high-speed link and the hub processes data differently in different nodes, enabling parallel

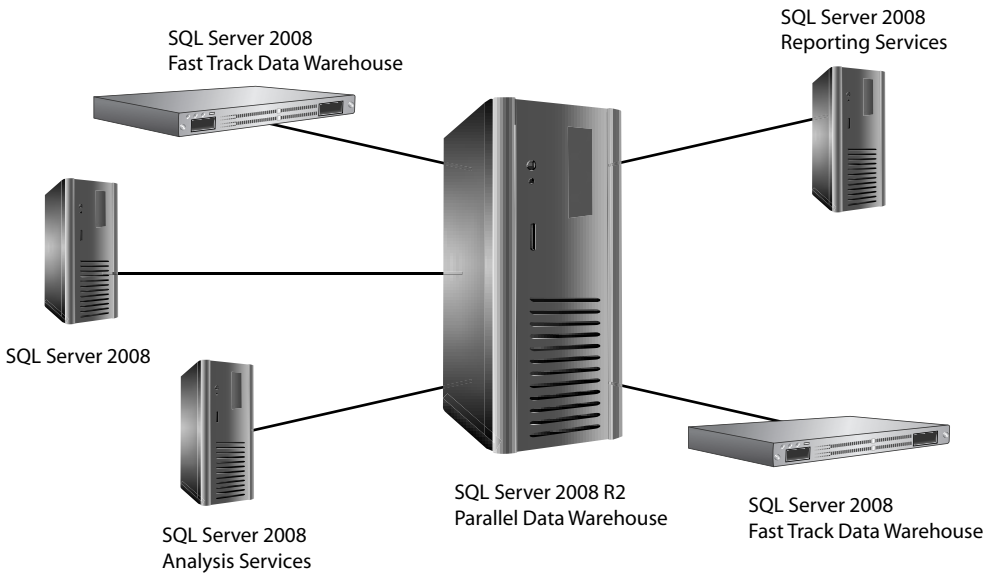


Figure 12-4 *A parallel data warehouse in hub-and-spoke architecture*

high-speed data transfer from node to node between hub and spoke units. Data transfer speeds approaching 500GB per minute can be achieved, thus minimizing the overhead associated with export and load operations.

The SQL Server 2008 R2 Parallel Data Warehouse MPP appliance integrates very well with BI applications such as Integration Services, Reporting Services, and Analysis Services. So, if you have an existing SQL Server 2008 data mart, it can be easily added as a node in the grid. As spokes can be any SQL Server 2008 database application, this architecture provides a best-fit approach to the problem. The enterprise data is managed at the center in the MPP appliance under the enforcement of IT policies and standards, while a business unit can still have a spoke that they can manage autonomously. This flexible model is a huge business benefit and provides quick deployments of data marts, bypassing the sensitive political issues. This way, you can easily expand an enterprise data warehouse by adding an additional node that can be configured according to the business unit requirements.

The Parallel Data Warehouse hub-and-spoke architecture utilizes the available processing power in the best possible way by distributing work across multiple locations in the grid. While the basic data management such as cleansing, standardization, and metadata management is done in the hub according to the enterprise policies, the application of business rules relevant to the business units and the analytical processing is handled in the relevant spokes. The hub-and-spoke model offers benefits such as parallel high-speed data movement among different nodes in the grid, the distribution of workload, and the massively parallel architecture of the hub where all nodes work in parallel autonomously, thus providing outstanding performance.

With all these listed benefits and many more that can be realized in individual deployment scenarios, the hub-and-spoke reference architecture provides the best of both worlds—i.e., the ease of data management of centralized data warehouses and the flexibility to build data marts on use-case scenarios as with federated data marts.

SQL Server 2008 R2 Data Warehouse Enhancements

In this section some of the SQL Server 2008 enhancements are covered, such as backup compression, MERGE T-SQL statements, change data capture, and partition-aligned Indexed views.

Backup Compression

Backup compression is a new feature provided in SQL Server 2008 Enterprise Edition and above and due to its popularity, has since been included in the Standard Edition and in the SQL Server 2008 R2 release as well. Backup compression helps to speed up

backups and reduce the size of the backup file. As SQL Server backup compression compresses the backup, the device I/O decreases due to reduced size, but the CPU usage increases due to compression overhead. However, the effect of reduced size and I/O makes backups much faster. If you find that the CPU usage has increased to the level that it starts affecting other applications, you can create a low-priority compressed backup in a session whose CPU usage is limited by the Resource Governor. The Resource Governor is another new component that is released in SQL Server Enterprise Edition to enable you to manage SQL Server workload and resources by specifying limits on resource consumption by the incoming requests. Refer to Books Online for more details on the Resource Governor. Though the backup compression is a feature in Enterprise Edition, you can restore compressed backups on any edition of SQL Server.

There are some restrictions that you should be aware of; for instance, you cannot mix compressed and uncompressed backups on the same media, and you don't get much compression of the database when you use transparent data encryption at the same time; hence the use of both together is not recommended. You can use backup compression in large data warehouse implementations to achieve the following goals:

- Reduce the size of SQL backups in the vicinity of 50 percent or more.
- Save hard disk space to keep backups online.
- Be able to keep more copies of backups online for the same storage space.
- Reduce the time required to back up or restore a database.

Backup compression configuration is specified at the server level but can be overridden. The default setting is off for the server. You can change this setting in SSMS by checking the Compress Backup option in the Database Settings page of Server Properties or by using the sp_configure stored procedure to set the default value of backup compression and then execute the reconfigure statement. You can override the server level setting for one-off or single backups by using one of the following methods:

- Specify the Compress Backup or Do Not Compress Backup option in the Set Backup Compression field while using the Back Up Database task in an SSIS package. Refer to the bottom of Figure 5-30 in Chapter 5 to see this option.
- Specify the Compress Backup or Do Not Compress Backup option in the Options page of the Back Up Database dialog box while backing up a database in SSMS.
- Specify the WITH NO_COMPRESSION or WITH COMPRESSION switch in the Backup statement.

MERGE Statement

SQL Server 2008 includes this new T-SQL statement that can be used to perform multiple DML operations—i.e., INSERT, UPDATE, and DELETE operations—on a table or view in a single statement. As the operations are applied in one statement, by their very nature they are executed under a single atomic operation. Using a MERGE statement, you join a source table with a target table or view and then perform multiple DML operations against the target table based on the results of that join. Typically, you apply these operations using a batch or a stored procedure individually, which means the source and the target tables are evaluated and processed multiple times, once for each operation. For a MERGE statement this evaluation happens only once for all the operations, thus making it more efficient when compared to applying the DML operations individually.

The MERGE statement specifies a target table in the MERGE INTO clause and the source table in the USING clause. Then it uses the ON clause to specify the join condition for the source and target tables, which primarily determines the rows in the source table that have a match in the target table, the rows that do not, and the rows in the target table that do not have a match in the source table. For the ON clause, it is recommended that at least you have unique indexes on join columns in both the source and target tables for better performance. After evaluation of match cases, you can use any or all of the three WHEN clauses to perform a specific DML action on a given row. The clauses are:

- **WHEN MATCHED THEN** You can UPDATE or DELETE the given row in the target table for every row that exists in both the target table and the source table.
- **WHEN NOT MATCHED [BY TARGET] THEN** You can INSERT the given row in the target table for every row that exists in the source table, but not in the target table.
- **WHEN NOT MATCHED BY SOURCE THEN** You can UPDATE or DELETE the given row in the target table for every row that exists in the target table, but not in the source table.

You can also specify a search condition with each of the WHEN clauses to choose the rows to apply to the DML operation. The MERGE statement also supports the OUTPUT clause, enabling you to return attributes from the modified rows. The OUTPUT clause includes a virtual column called $action, which returns the action that modified the row—i.e., INSERT, UPDATE, or DELETE.

You can embed a MERGE statement inside the Execute SQL task to use it in your SSIS package. Typically, you would stage data to a staging table to change capture before loading it in the data warehouse with an Integration Services package. Such an SSIS package would include a Lookup task to identify whether a row is a new row or an update, an SQL Server Destination to INSERT new rows, and OLE DB Command transformations to perform UPDATE and DELETE operations. The Lookup transformation and the OLE DB Command transformation work on a row-by-row basis; thus they perform at a speed that can't match the set-based operation of a MERGE statement. If you stage data for change capture and make it as a source table for a MERGE statement, you will find that the data is loaded at a far better rate than using SSIS with or without staging data. This performance becomes better especially on servers where lookup is working against large data sets and is running short of memory.

You can also replace a Slowly Changing Dimension (SCD) transformation with a MERGE statement in some cases. Again, as the MERGE evaluates the source and the target data only once, it performs much better than the SCD transformation, which otherwise has been recognized as a performance pain point in SSIS. In the following example code, DimProduct is updated with the changes being received in the ProductChanges table. The changes might include changes in price for some existing products that need to be updated and some new products that need to be added. Considering the changes as Type 2, the IsCurrent flag has to be reset to 0 for the updates and new rows for them need to be inserted along with the new products. Look at the inner query where the MERGE statement updates the IsCurrent flag to 0 under the WHEN MATCHED clause and adds a new row under the WHEN NOT MATCHED clause for the new products. Finally, it outputs the rows affected with the action taken for them and then the outer query inserts the rows that have been updated back into DimProduct table.

```
INSERT INTO DimProduct (ProductID, ProductName, Price, IsCurrent)
    SELECT ProductID, ProductName, Price, 1
    FROM
    (
       MERGE DimProduct as TGT
        USING ProductChanges AS SRC
        ON (TGT.ProductID = SRC. ProductID and TGT.IsCurrent = 1)
        WHEN MATCHED THEN
            UPDATE SET TGT.IsCurrent = 0
        WHEN NOT MATCHED THEN
            INSERT VALUES (SRC.ProductID, SRC.ProductName, SRC.Price, 1)
        OUTPUT $action, SRC.ProductID, SRC.ProductName, SRC.Price
    ) AS Changes (action, ProductID, ProductName, Price)
    WHERE action = 'UPDATE';
```

Just as MERGE can replace of some of the SSIS components for performance reasons, it can be used with change data capture (CDC) functionality to perform inserts and updates, thus replacing parameterized OLE DB Command transformation and resulting in a considerable gain in performance. One such application could be moving staged data into production servers.

GROUP BY Extensions

The GROUP BY clause has been enhanced in SQL Server 2008 and now adds a new operator: GROUPING SETS. Actually, the GROUP BY clause in SQL Server 2008 now supports ROLLUP, CUBE, and GROUPING SETS operators in line with ANSI SQL-2006 standards. So, an ISO-compliant syntax has been adopted, though a non-ISO-compliant syntax is still supported for backward compatibility—e.g., the earlier operators supported in SQL Server 2005 (WITH CUBE and WITH ROLLUP) are still supported but will be removed from future versions. As you know, a GROUP BY clause enables you to select one summary row for each group of rows created from a selected set of rows on the basis of values of one or more columns or expressions. Adding the operators to a GROUP BY clause provide an enhanced result set.

The ROLLUP operator generates simple aggregate groupings with expressions rolled up from right to left and the number of groupings in the result set equal to number of expressions plus one. The CUBE operator generates groupings for every combination of the expressions without any regard to expression order and generates 2^n groupings, where n is the number of expressions used with the CUBE operator. The GROUPING SETS allow you to produce multiple groupings of data only for the specified groups in a single result set. This is simple and better than the ROLLUP and CUBE operator that generate the full set of aggregations. GROUPING SETS can specify groupings equivalent to those returned by ROLLUP or CUBE and can also work in conjunction with ROLLUP and CUBE. The result set thus generated is equivalent to a UNION ALL of the specified groups. So using GROUPING SET, you can generate only the required levels of groupings and the information can be readily made available to reports or requesting applications in a ready-to-digest format. You might use a pivot table to display the results. However, the important thing to take away is that using the GROUPING SETS operator can allow you to retrieve aggregated information in the required levels of grouping all in one single statement and above all efficiently. This could enhance your data analysis experience and improve reporting performance. If you are using an Aggregate transformation to produce aggregations for a report, you may find using GROUPING SETS performs better in some cases, especially where you are running a data flow task only for performing aggregations.

Star Join Query Processing Enhancement

This is one of those improvements in SQL Server 2008 Enterprise Edition that is applied automatically and does not require users to make any changes to their queries. If you are using a dimensional data model for your warehouse, you may realize this performance gain just by upgrading to SQL Server 2008 and without making any change to your database structure or the queries. Microsoft claims to have achieved 15 to 30 percent improvement in the whole of the star join payload on the dimensional database server, whereas some individual queries can benefit from it by a factor of seven. The star join optimization is provided in Microsoft SQL Server 2008 Enterprise Edition.

In a dimensional data warehouse, most of the queries perform aggregations on columns of a large fact table and join it to one or many smaller dimensional tables, apply filter conditions on the non-key dimensional table columns and form groups by one or many dimensional table columns. Such queries are called star join queries. These queries follow a similar processing pattern that you can find out by looking at the execution query plan of some of these queries. Typically, the fact table will be scanned for a range of data by running a seek on the clustered index that will be hash-joined with the results of the seek on one of the dimension table. These hash joins are repeated for as many times as there are dimensions involved in the query, and finally the results are sent to an aggregation hash.

If you check out this plan on an SQL Server 2008 server, you will notice some bitmap filters have also been applied. The bitmap filters, also called the bloom filters, are generated as a by-product of the hash joins. The bloom filters are data structures that can probabilistically test whether an element belongs to a set. Though this technology can allow some false positives, it does not allow any false negatives. So, once a fact table row fails the test, it will be excluded from further processing. SQL Server 2008 can generate multiple bloom filters, as against SQL Server 2005, which could do only one. These filters are pushed down in the query execution path to the scan of the fact table to eliminate the nonqualifying rows. Also, the SQL Server 2008 query execution engine can change the sequence in which these filters are applied on the basis of their selectivity—i.e., by placing the most selective filter first, and then the next most selective filter, and so on—thus extracting maximum performance out of this enhancement. So, the bitmap filters enable SQL Server to eliminate nonqualifying fact table rows from further processing quite early on during the query evaluation. This avoids unnecessary processing of the data rows that would have been dropped later in the query processing anyway and results in saving a considerable amount of CPU time.

Change Data Capture

When you are loading data into a data warehouse system from a source system, you need to know about the rows in the source system that have been either changed or

inserted or deleted. Until now, the options were to either extract complete source system data to a staging table to compare it against the data warehouse, so as to determine the changes or to use alternate techniques such as timestamp columns, triggers, or complex queries for this purpose. Though these methods work, they have some issues. The first option is very inefficient due to its doing lot of work to find out the changes. Implementing the second option is also not a good choice, as timestamp columns require application changes and triggers or complex queries are not efficient. SQL Server 2008 Enterprise Server (and Developer Edition) has provided an efficient feature called Change Data Capture (CDC) to track data changes such as inserts, updates, and deletions. So, instead of comparing staged data with data warehouse data or using intrusive methods to find the changes in data, you can use CDC to identify changes and eventually load the data warehouse in incremental steps.

The Change Data Capture feature is not enabled by default, so you have to enable CDC first in order to use it. The CDC feature is enabled at two levels—the database and table levels. When you enable CDC on a database, it creates a new schema, CDC, and creates a user account, CDC, in the database. The CDC schema is used to store all the change tables and their metadata. When you decide to track a table for changes, you enable CDC on that table. Enabling CDC on first table creates the following:

- A change table to capture changes to data and metadata
- Up to two query functions to allow you to retrieve data from the change tables. By default, all changes will be captured and only one query function is created in this case; however, if you want to track only the net changes over a period of time, you can do so. You can specify a parameter @support_net_changes = 1 in the CDC enabling command to return only the net changes. This setting creates the second query function that allows you to return only the net changes. This feature can also potentially reduce the number of updates you perform on your data warehouse.
- A group of change data capture metadata tables to retain metadata configuration detail
- Two SQL Server jobs: capture job and cleanup job. The SQL Server Agent service must be running when you enable CDC tracking on a table.

When the changes are applied to the tracked source table, the database engine as usual writes the changes in to the transaction log. The CDC capture job reads the log automatically and adds information about the changes into the associated change table. This table holds the capture columns from the source table to capture the changed data and five additional metadata columns to provide information relevant to the captured change. Two columns are of particular importance and worth mentioning here. The first column, __$start_lsn, records the commit log sequence number (LSN) that

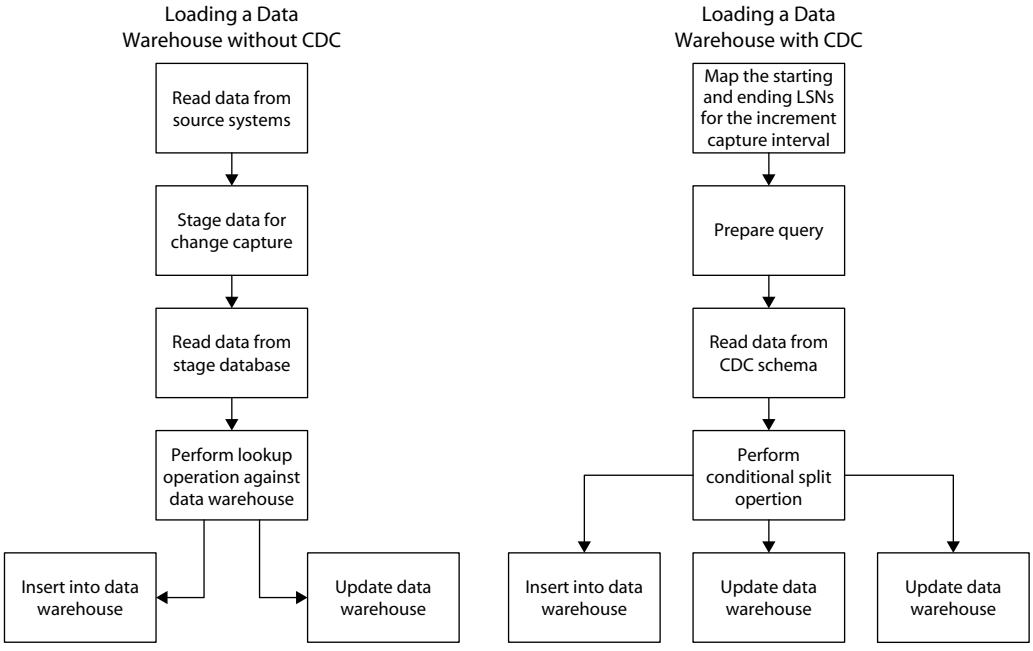


Figure 12-5 *Loading a data warehouse using Change Data Capture*

was assigned to the change. The commit LSN not only identifies changes that were committed within the same transaction but orders them as well.

The second column we want to discuss is __$operation, which records the operation that is associated with the change, for instance, 1 = delete, 2 = insert, 3 = source record prior to update, and 4 = source record after update. This column makes the ETL very efficient by eliminating the need to identify whether a row is an update, a delete, or an insert. Refer to Figure 12-5, which shows methods to load a data warehouse both with and without CDC. With CDC, instead of performing an expensive lookup operation, you just split data conditionally using a Conditional Split transformation. The split condition uses the __$operation column to divert the rows to appropriate output.

Partitioned Table Parallelism

In SQL Server 2005, when you run a query against large tables that have been partitioned across several disks probably based on dates, it has been observed that the executor allocates threads in an inconsistent manner. If a query touches only one partition, the executor allocates all the threads to the query; however, if a query needs to touch more

than one partition, it gets one thread allocated per partition, though additional threads might still be available on the server. Sometimes this has produced inconsistent results for the same or similar queries, depending on when they have been executed. In SQL Server 2008, the parallel query plans against partitioned tables have been improved to utilize more threads regardless of how many partitions a query touches. The SQL Server query executor allocates all the available threads to partition table queries in a round robin fashion, keeping CPUs fully utilized at all times so that the performance of the same or similar queries over time is more comparable. The performance boost achieved with this round robin–style thread allocation is noticeably high, especially when more processor cores are available compared to the number of partitions a query touches. You don't need to configure anything in the SQL Server, as this feature works by default.

Partition-Aligned Indexed Views

In SQL Server 2005, both indexed views and the partition tables can be used together. For instance, if you have a very large fact table that has been partitioned, say, on a yearly basis and you want to create summarized aggregates on this fact table, you use indexed views to achieve this. So far, so good; however, there is an issue with the way people use fact table partitioning. The fact table is generally partitioned on the basis of a time period, which could be a month, quarter, or year, depending upon your business needs, and as a time period completes, the oldest partition need to be switched out and a new partition has to be switched in to keep data for a fixed period of time. For example, if you are required to keep data for five years and you are keeping partitions on year-by-year basis, then at the beginning of a year you need to remove the oldest partition (six years old) and add a new partition for the new running year. The point to note here is that switching a partition in and out from the fact table is very efficient and is best possible way you can manage data in terms of retention. This way, your data warehouse will always keep the data for the defined period of time and the retained data is partitioned for performance. This is known as a *sliding window scenario.* The issue with SQL Server 2005 is that you cannot switch a partition in or out from a partitioned table that has indexed views created on top of them. You will end up dropping the indexed view, switching the partition in or out, and then recreating the indexed view. The creation of an indexed view could be a very expensive process that can delay data delivery to your end users.

SQL Server 2008 aligns both these features and makes them work together. You don't need to drop and recreate indexed views when switching a partition in or out of it. With the enhancement of partition-aligned indexed views, you save lot of extra processing that would otherwise be required to rebuild aggregates on the entire partitioned table.

Summary

This chapter has covered basics about data warehousing and some of the enhancements provided in the SQL Server database engine. The two new editions that have been introduced in the R2 release are interesting to watch, as they realize an approach to bring together software, hardware, and best practices to achieve the highest performance. This chapter is targeted to wet your feet, so if you feel interested, go and take a deep dive in the area of data warehousing. For now, stay on as we still have some interesting topics to cover: migration, package deployment, troubleshooting, and performance tuning.

Chapter 13

Deploying Integration Services Packages

In This Chapter

- Package Configurations
- Deployment Utility
- Deploying Integration Services Projects
- Custom Deployment
- Summary

Till now you have learned how to build SSIS packages on a server and run them from within BI Development Studio (BIDS) or using the dtexec command prompt utility, but you have not deployed the packages to other environments or servers. In real life, you will be developing packages in development environment and then deploying them into the production environment. The predecessor of SSIS, DTS 2000, struggled with the limitation of moving packages between development, test, and production environments. Typically in DTS 2000, you had to edit your packages manually to point the connection objects to the server to which you were deploying your packages. DTS 2000 used an object called the Dynamic Properties task to update package properties with the values retrieved from sources outside a DTS package during run time to overcome the deployment issues, but it was not sufficient to address the problems associated with complex deployments. This object has been removed from the SSIS object model and has been replaced with SSIS package configurations.

Integration Services addresses the issues involved with complex deployments using package configurations, as you will learn in this chapter. You can use different tools and utilities to deploy a package in a step-by-step process. You can update properties of package elements at run time using package configurations, create a deployment utility to deploy the package, and finally run the Installation Wizard to install the package either to the file system or to an instance of SQL Server.

The following table lists the development journey of a package:

Action	Tool Used	Result
Create package	BIDS or SSIS Import and Export Wizard or Programmatically	An XML-formatted SSIS package created—e.g., package.dtsx
Create package configurations	BIDS Package Creation Wizard	Package configurations created in XML/SQL
Create deployment utility	Build project	Package deployment utility with manifestation file
Install package	Package Installation Wizard	Package deployed in SQL or file system

The first step in the deployment journey is to create a package, which you have already learned in the previous chapters; you will be using those packages here to learn about the various aspects of deployment.

Package Configurations

The next step in the deployment journey after having created a package is to create package configurations. By now, you understand the versatility of variables and their preferable use in packages over hard-coding the values. Also in Integration Services, you can update the variable values using other components at run time to add some dynamics to your package. Package configurations extend this concept to properties of SSIS control flow components to add flexibility to the packages. Package configurations

allow properties of a package; its control flow container; or any of the tasks, variables, or connection managers to be changed at run time. Package configurations are the property/value pairs that you use to update the values of properties of a package or its control flow components at run time. The package configurations make deploying a package to other servers easier by allowing you to update properties exposed by the package and its tasks at run time without actually editing them. You can use the same configurations to deploy a package to several servers.

To use package configurations, you must first enable them for your package using the Package Configurations Organizer utility provided in BIDS. The package configurations are saved inside the package, which is an XML file. So if you want, you can simply edit this file even in a basic tool such as Notepad to modify or add a new package configuration; however, most people will prefer to use the Package Configurations Organizer utility, as it makes creating a configuration very easy by providing access to various objects such as already-defined configurations and available properties to export, all inside a wizard-driven interface.

Once the package configurations have been enabled, you can add package configurations. You can define more than one configuration for a package or apply the same configuration to more than one package defined in your solution. You can enable package configurations for one of your packages without affecting the way the configurations are used in other packages. Various storage options are available for storing your configurations, including reading them directly from the environmental variables. The storage options for package configurations, discussed in detail in the following section, include an XML configuration file, an SQL Server table, a registry entry, or a variable.

Types of Package Configurations

You can open the Package Configurations Organizer dialog box by choosing the Package Configurations option under the SSIS menu and enable the package configurations. After you enable package configurations, you will choose the configuration types (see Figure 13-1) in the Package Configuration Wizard before you can proceed to building package configurations.

Depending on the configuration type you choose here, the package configurations will be stored to different storage locations. The following list shows five different types of stores supported by Integration Services package configurations. You can choose any of these configuration types while adding a package configuration using the Package Configuration Wizard:

- XML configuration file
- Environment variable

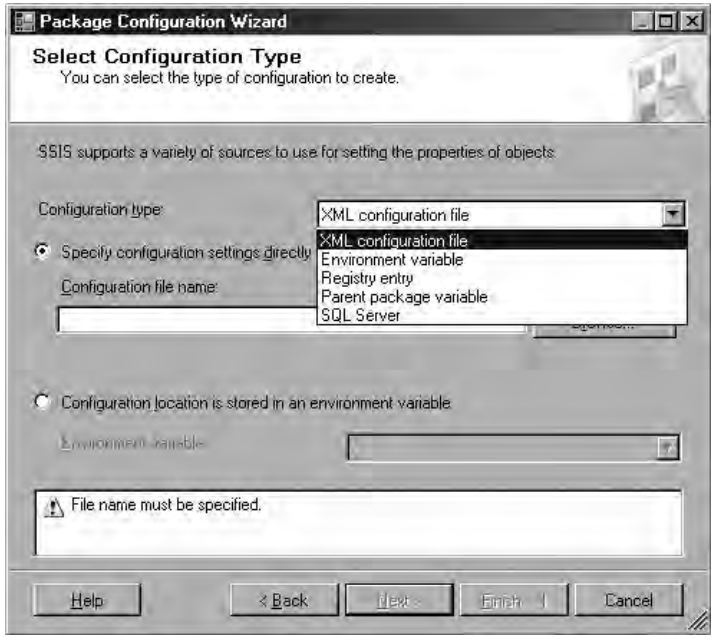


Figure 13-1 *Choosing configuration types for different storage locations*

- Registry entry
- Parent package variable
- SQL Server table

XML Configuration File

When you select XML Configuration File in the Package Configuration Wizard, you must then provide an XML filename, including the path. Here you can create a new configuration file or open an existing file. Then you will be able to select any of the properties or variable to add to the configuration file. You can include multiple property/value pairs of configurations within the same configuration object using an XML configuration file type. Selecting an executable (container) or a task adds all the configurable properties in the scope of that container or task.

If you open the configuration file after it has been created, using any XML editor or Notepad, you will notice that the root element, <DTSConfiguration>, has two child elements, <DTSConfigurationHeading> and <Configuration>. The <DTSConfigurationHeading> element contains details about the file, such as the name

of the login that generated the file, the package name, the ID from where this file was generated, and the generated date; the <Configuration> element contains information about the configurations, such as the property path and the configured value.

Environment Variable

Selecting this option in the Package Configuration Wizard allows you to choose an environment variable from the list provided. Using this type of configuration allows you to update a property directly based on an environment variable.

Registry Entry

Integration Services allows you to store the configuration in a registry key by using the Registry Entry configuration type. When you select this configuration type, you must then specify a registry key and choose a target property that will have its value updated by the registry key.

Parent Package Variable

When you are running a package as a child package using the Package Execute task in the parent package, you may want to use a variable defined in the parent package to set a property in the child package. To do this, you use the Parent Package Variable configuration type in the Package Configuration Wizard. You can then specify a variable name defined in the parent package to store the configuration value used to set a property in the child package.

The package configurations are applied at the package loading time and not at the package execution time, which occurs after the package is loaded and the configurations have been applied. So, when you run a package, the package is loaded as a first step, then package configurations are applied, and then the package is executed, though there is one exception to this: when you use the Parent Package Variable configuration option, the configuration is applied at execution time. This makes perfect logical sense, as the parent package variable needs to be already populated by the time the child package is being executed.

SQL Server

By choosing SQL Server as the configuration type in the Package Configuration Wizard, you can store the package configurations in an SQL Server database table. Using the SQL Server configuration type, you can store multiple property/value pairs of configurations within the same configuration object. You can use the SQL Server configuration type to store all the configurations for your project in the same table and use it as a central store to keep configurations.

The connection to the database that you select to host the table can be created as a new OLE DB Connection Manager, or you can specify an already-existing connection manager. The SQL Server configuration creates ConfigurationFilter, ConfiguredValue, PackagePath, and ConfiguredValueType columns in the new table you create.

Hands-On: Applying Configurations to Contacting Opportunities

In this Hands-On exercise, you will use package configurations to change e-mail addresses and demonstrate how package configurations can affect the working of a package. For demonstration purposes, you will use the Mailing Opportunities.dtsx package you built in Chapter 4.

Method

As you will be modifying the existing Mailing Opportunities.dtsx package, to avoid any confusion with package names, you will create a new project in which you will add the Mailing Opportunities.dtsx package with a new name: Mailing Opportunities with Configurations.dtsx. You will then open the newly added package, Mailing Opportunities with Configurations.dtsx, and enable package configurations. The ToLine property of the Mailing Opportunities task determines who will receive the e-mail message. You will change this property in the package configuration file to demonstrate how the properties of a package can be modified at run time.

Exercise (Enable and Add Package Configurations)

You will enable configurations using the Package Configurations Organizer utility provided in BIDS. Then you will add a ToLine configuration in the package.

1. Start BIDS and create a new Integration Services package with the following details:

Name	Contacting Opportunities with Configurations
Location	C:\SSIS\Projects

2. Add an existing package from the File System using C:\SSIS\Projects\Contacting Opportunities\Mailing Opportunities.dtsx as the Package Path. When the package is added, rename this package as **Mailing Opportunities with Configurations.dtsx**. Then delete the Package.dtsx package.
3. Double-click the Mailing Opportunities with Configurations.dtsx package to load it. When the package is loaded, right-click anywhere on the blank surface of the Designer and choose Package Configurations from the context menu. This will open the Package Configurations Organizer dialog box. You can also open it by choosing SSIS | Package Configurations.

4. Select the Enable Package Configurations check box. This setting is package specific and will not affect any configurations defined in other packages in a multipackage project.
5. Now you can add configurations in this package. Click Add to start the Package Configuration Wizard. Click the Next button on the Welcome screen.
6. Leave the XML Configuration File option selected in the Configuration Type field.
7. Next, choose to specify a configuration directly or indirectly by using an environmental variable to pass the value. In this exercise, you will use the direct package configuration. In the Specify Configuration Settings Directly field, type the filename and path for the XML configuration file as shown in Figure 13-2—**C:\SSIS\Projects\Contacting Opportunities with Configurations\conoppo .dtsConfig**. Note that the package configuration file has a .dtsConfig extension when saved as an XML file. Click Next.
8. In the Select Properties To Export page, you can select to export the variables, properties of the package, properties of the connection managers, or properties of any of the executables or tasks used in the package. Under the Objects pane, expand the Executables folder in the Mailing Opportunities package. Under the Executables folder, expand Iterating October Opportunities followed by the Executables subfolder used in this package. Again under Executables, expand the Mailing Opportunities subfolder, followed by Properties, and then check in the Subject

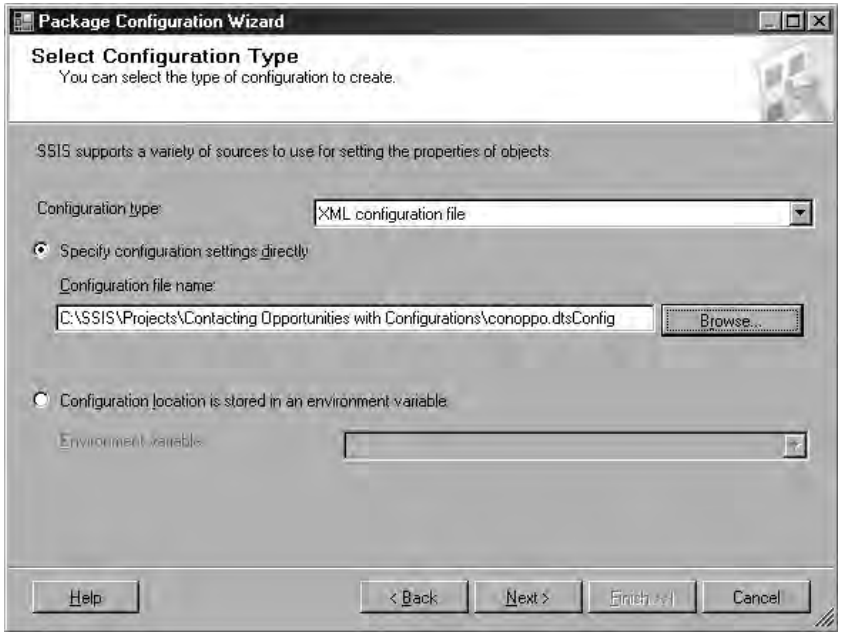


Figure 13-2 *Specifying configurations directly*

and ToLine properties to select them, as shown in Figure 13-3. After selecting these properties, click Next to move on to the Completing The Wizard window.

9. Type **Config_emails** in the Configuration name field and review the options before clicking Finish.
10. When you return to the Package Configurations Organizer dialog box, you will see the configuration you just added listed there. You can return to this dialog box later to add more configurations, edit already-defined configurations, or remove configurations from the package. The configuration we added updates multiple properties (Subject and ToLine properties). You can add multiple configurations in the way you've added the Config_emails configuration into your packages. The configurations are applied in the order they appear in this list. The configurations are applied at loading time, so when multiple configurations update the same property, the value updated last is used at run time. You can control this order by using the direction arrow buttons provided on the right side of the dialog box (shown in Figure 13-4) to move configurations up or down.
11. Pay some attention to the columns that appear under the Configurations area. The configuration name you defined is shown under Configuration Name; Configuration Type shows the type of configuration you have chosen; Configuration String shows the path where the configuration is located. You can create multiple configurations and group them logically for a package in the

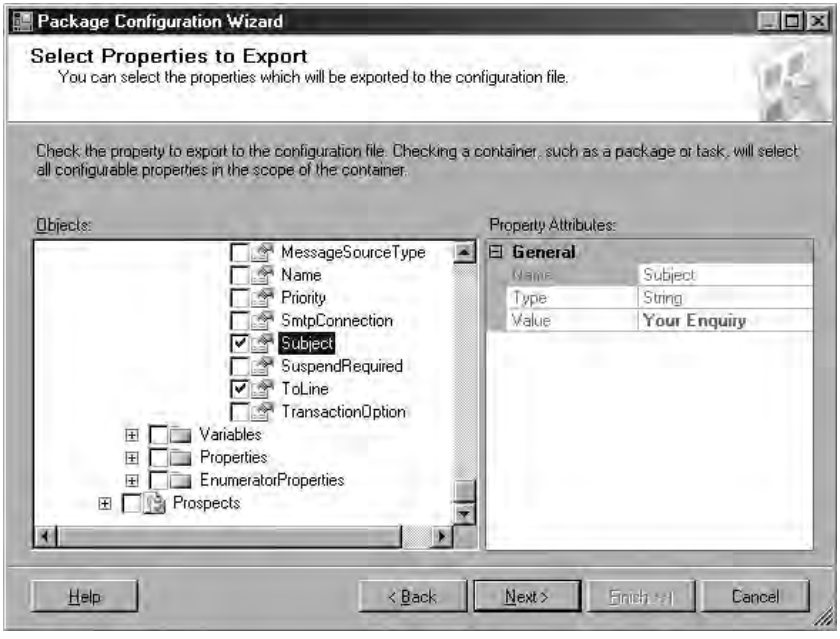


Figure 13-3 *Selecting properties to export for creating configurations*

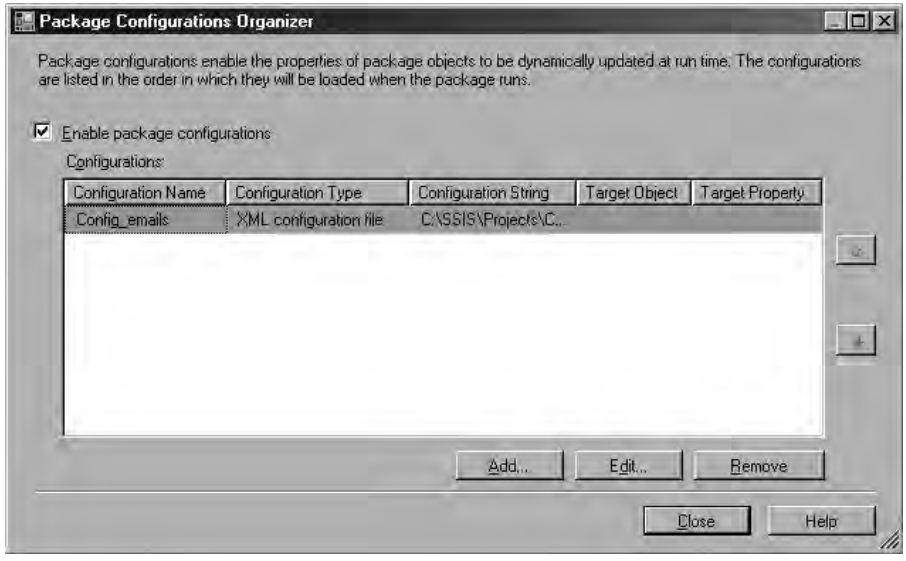


Figure 13-4 *Package Configurations Organizer dialog box showing a configuration*

Configuration Name column. In this exercise, we have modified two properties under one configuration name, Config_emails. Two more columns, Target Object and Target Property, refer to the name of a target object and that of a target property, respectively. The use of an XML configuration file renders the Target Object and Target Property columns blank, as it can update multiple objects. Click Close to close the dialog box.

Exercise (Assign a Value to the Package Configurations and Debug)

In this part, you will assign an e-mail address to the Subject and ToLine configurations and see how they are used by the package during run time to update the values specified (hard-coded) in the package.

12. So far, you have defined a package configuration for the ToLine property. The next step is to assign a value to this configuration. Choose File | Open | File to open a file. In the Open File dialog box, open the conoppo.dtsConfig file by exploring to the C:\SSIS\Project\Contacting Opportunities with Configurations folder and clicking Open.
13. When the conoppo.dtsConfig file opens in a new tab, click the Edit | Advanced | Format Document menu bar command to format the XML layout. Locate

the <ConfiguredValue> element for the ToLine property, after the following configuration:

```
<Configuration ConfiguredType="Property" Path="\Package\Iterating October
Opportunities\Mailing Opportunities.Properties[ToLine]" ValueType="String">
```

Type your alternative e-mail address between <ConfiguredValue> and </ConfiguredValue>. Similarly change the ConfiguredValue of the Subject property to **Your first enquiry** (see Figure 13-5). You don't need BIDS to edit configurations file, you could have done this using any text editor such as Notepad.

14. Now you'll test your configurations. Execute the package by pressing the F5 function key to start debugging the package.
15. You will see the tasks turning yellow and then green to indicate the successful completion of the package. Stop the debugging when all the tasks have completed.
16. Switch to your e-mail application and check the inbox of the alternative e-mail address you specified in the configuration file. You will see e-mails from your package.

Review

You have successfully created a configuration for two of the properties exposed by the Send Mail Task in your package and have assigned values to the exposed properties. Upon executing the package, you saw that the values configured in the package were modified to use the values specified for the configuration. The value can be assigned to the configuration either directly, as you did in this Hands-On exercise, or by

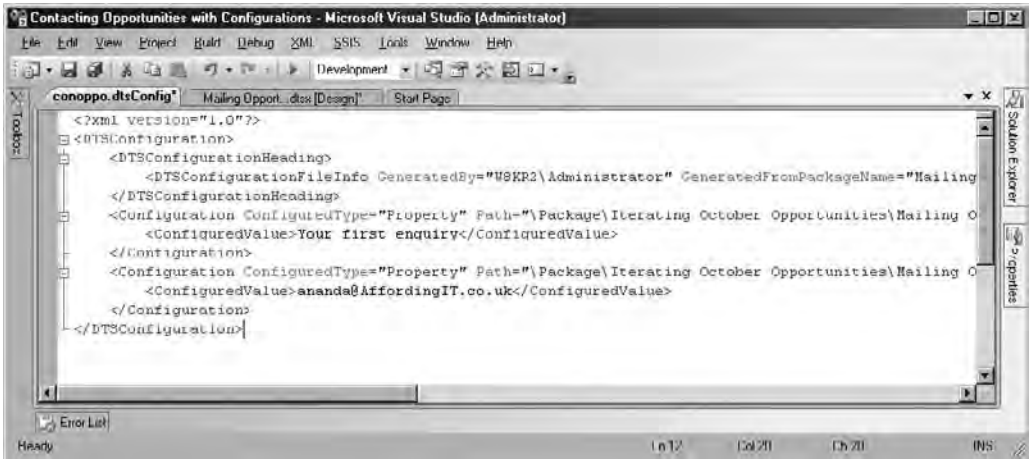


Figure 13-5 *Assigning values to exported properties in package configurations*

directing it to read from an environment variable. In situations when you are sure that the path used by the configurations will be the same for all deployments, use of direct configurations offers a simple solution.

Direct and Indirect Configurations

You can specify the package configurations in two ways—direct and indirect.

In direct configuration, you specify the configuration using any of the five configuration types mentioned earlier. For example, you specify a configuration for a property in an XML configuration file and then assign a value to that configuration. As you deploy the package, this configuration gets deployed with the package. You get an opportunity to specify the configuration file path and modify the configuration value during deployment.

If you decide to deploy the configuration file in such a way that in different environments you keep the same configuration file path and the value of the variable, you can hard-code the configuration values. In this type of deployment, Integration Services reads the configuration file from the path specified during deployment and links directly the configured property in the package and the value assigned to it in the configuration. For simple environments—i.e., those in which you can specify the deployment path and the configuration values during deployment—direct configurations are a better choice because they offer flexibility of making changes during deployment.

However, bear in mind that the configurations become part of the package and are deployed with the package. When your package needs to be deployed to various environments on many servers, you may encounter different configurations for different machines and the opportunity of modifying package configurations during deployment in such situation becomes a hassle. This can cause some management and deployment issues—for instance, if you have special environment needs for a package, then deploying such packages to computers with different environments may call for a lot more work at deployment time and cause some grief.

The solution to this problem lies in using indirect configurations, in which you create a configuration file for each computer once and then use an environment variable to access the computer-specific file. This computer-specific file doesn't flow with the package during deployment; instead, it lies at the deployment computer all the time and becomes responsible for creating an environment into which the package will be deployed. Unlike direct configuration, in this case the package will access this computer-specific configuration file indirectly using an environment variable to free you up from specifying the configuration path each time you deploy the package.

For indirect configurations, you create the configuration file and copy it to a folder on the computer where you want to deploy the package, create an environment variable that points to this file, and then on the development computer point your package to

read the configuration file path from an environment variable. So when you deploy the package, the configuration file is available to the package via the environment variable.

When you create an indirect configuration for an XML configuration file, instead of specifying the file path directly, you point your package to read the file path for the XML configuration file from an environment variable. The computer-specific XML configuration file defines the environment on the deployment computer. Once you have created such XML configuration files for each computer, the package deployment gets reduced to copying your package to these computers.

For still more complex environments, you can update the variable values in the package during run time using configurations, and the variables in turn modify properties of the package through the use of property expressions. In this way, you can have a totally disjointed configuration for your package with different bits playing their roles independently of each other, yet in a coordinated way.

The concept of indirect configurations may seem complex, but in reality is simple to adapt and use; quite interestingly, we already use this concept in real life. Think of your car as a package that has properties such as steering wheel position, pressure on the brake pedal, and so on. These properties change the behavior of the package, the car in our analogy. You, the driver, are modifying these properties that are analogous to property expressions. To modify the properties of car, you are guided by the road conditions (variable values in package analogy), such as traffic lights or traffic signs. And, above all, these road conditions are regulated and applied by a traffic regulating agency, which is the configuration file in our package. So the package configuration sets the environment (traffic lights), the environment changes the variable values (traffic light turning green), and using those modified values, property expressions modify the behavior of the package (applying the brake to stop the car).

Hands-On: Using Indirect Configurations

You will be creating an indirect package configuration for downloading the zipped files package that you created in Chapter 5.

Method

For indirect configurations, you create a configuration file on the computer to which you want to deploy the package. This configuration defines the environment for the computer, which is used by the package to modify its properties. The following steps will be used for this exercise:

- Create a computer-specific package configuration.
- Create an environment variable.

- Use property expressions to update the ConnectionString property.
- Create an indirect package configuration.
- Deploy and execute the package.

A couple of things to note here: If you want to use the Downloading zipped files package that has been provided with this book, you will receive an error when opening the package. When you click OK for the error dialog, the package will load properly sans the connection string in the FTP task. This is because, by default, the sensitive information (passwords, connection strings, and so on) in the packages get encrypted using the user key (my user key in this case), and when someone else tries to open the package, an error will occur and sensitive information will be removed from the package. However, if you open the Downloading zipped files package that you developed yourself in Chapter 5, you will not get such an error.

This package requires a connection to an FTP server. If you skipped building this package in Chapter 5, I would recommend that you find an FTP server (many FTP sites allow you to download files that you can use to build this package—e.g., FTP sites for antivirus updates) and build the package to work through this Hands-On exercise. The provided package may not be of much help, as it points to my home computer, which is obviously not accessible to you.

Exercise (Create a Computer-Specific Package Configuration)

To deploy the package to multiple computers, you need to develop a configuration for each computer. You may do this at the development computer or at the deployment computer, depending upon the available resources. Once the configuration has been created, you then deploy the configuration file on the deployment computer. In this exercise, you will be using a development computer to create this configuration file.

1. Start BIDS and create a new Integration Services project with the following details:

Name	Downloading zipped files with Indirect Configurations
Location	C:\SSIS\Projects

2. After a blank project is created, add the existing package Downloading zipped files.dtsx from C:\SSIS\Projects\Control Flow Tasks\ folder. Also, delete the Package.dtsx blank package. Open the Downloading zipped files.dtsx package by double-clicking it.
3. Verify that the variable fpath exists in the package with the details shown next. Otherwise, add this variable by right-clicking anywhere on the blank surface of

the Designer and choose Variables from the context menu. Add a variable with the following details:

Name	fpath
Data Type	String

4. Now right-click anywhere on the blank surface of the Designer and choose Package Configurations from the context menu. Select the Enable Package Configurations check box.
5. Click Add to start the Package Configuration Wizard. Click Next on the first screen, and on the Select Configuration Type screen, choose the Configuration Type as XML Configuration file.
6. Select the Specify Configuration Settings Directly radio button. This will highlight the Configuration File Name field. Type **C:\SSIS\Projects\Downloading zipped files with Indirect Configurations\Configurations for DZF.dtsConfig** as the filename (see Figure 13-6). Click Next to move to the Select Properties To Export screen. This file path is not of much relevance, as we will finally move this file to the computers where the package will be deployed.

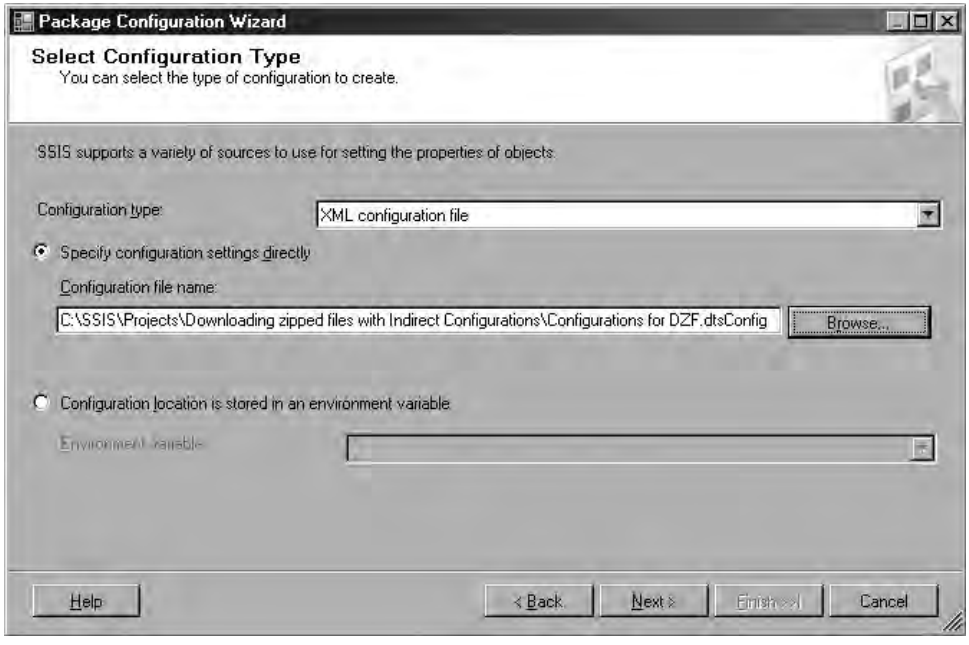


Figure 13-6 *Creating configurations for the deployment computer*

7. In the Objects pane, under the Downloading zipped files, expand the Variables\fpath\Properties. Check the Value property, as shown in Figure 13-7. Click Next to go to the next screen.
8. Type **Downloaded files path** in the Configuration Name field on the Completing The Wizard screen. Note that the Preview section shows the Type as Configuration File and tells you that a new configuration file will be created. Also note that the property \Package.Variables[User::fpath].Properties[Value] has been structured in a particular way. The basic rule here is that executables (containers or tasks) will start with a backslash (\); variables, properties, and connection managers will start with a dot (.); and an item in the collection will be enclosed within square brackets ([ ]). Starting with \Package, the variables collection has been started with a dot, which is then followed by the [User::fpath] variable enclosed in square brackets because it is one in the collection of variables. This is then followed by Properties, starting with a dot and the [Value] item in the properties collection enclosed within square brackets.

   Click Finish to complete the wizard and return to the Package Configurations Organizer dialog box. You will see that the newly created configuration has been added to the Configurations section. Click Close to return to the Designer surface.

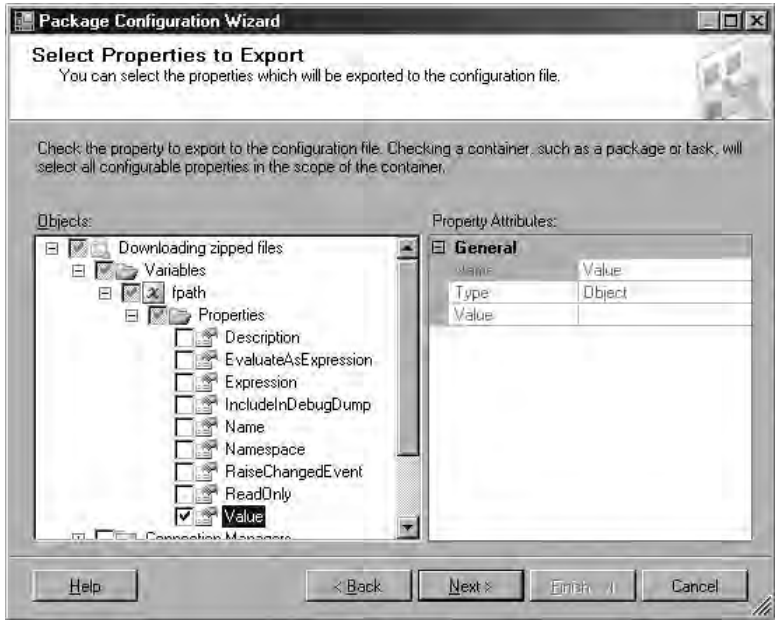


Figure 13-7 *Selecting properties to export*

9. Open the newly created configuration file Configurations for DZF.dtsConfig from the Designer menu: choose File | Open File and provide the path specified in Step 6. This will open a new tab on the Designer surface showing the XML configuration file. Locate the <ConfiguredValue> element and specify **C:\SSISDeploy** after the <ConfiguredValue> and before </ConfiguredValue> element. Save and close this file.
10. Choose File | Save All; then close the project and exit BIDS.

Exercise (Create an Environment Variable)

Let's now move the configuration file to the deployment computer and set up an environment variable on the deployment computer to point to the configuration file. In real life, you will be setting up an environment variable on a development computer, a UAT server, a staging server, a production server, and others, but the path value defined for the environment variable on the deployment computer will differ from computer to computer, depending on your requirements.

11. Create a folder on the deployment computer named **C:\SSISDeploy**, and move the newly created configuration file Configurations for DZF.dtsConfig from your development computer to this folder on the deployment computer.
12. Create a new environment variable on the deployment computer. Usually, the Environment Variables dialog box is found in the advanced system settings of My Computer Properties.
13. In the Environment Variables window, click the New button in the System Variables section and click the OK button in the pop-up window after typing in the following details:

Variable Name	DZFConfig
Variable Value	C:\SSISDeploy\Configurations for DZF.dtsConfig

14. You can scroll through the list of system variables to see this variable added to the list (Figure 13-8).
15. Repeat Steps 11 through 13 on the development computer so that you have access to the run-time environment on the development computer as well as for designing and testing purposes.

Exercise (Use Property Expressions to Update Exported Properties)

Now you'll use property expressions to update the ConnectionString property of the Downloads Connection Manager so that the downloaded zipped files can be saved to a folder specified by the fpath variable. BIDS requires a restart to load new environment

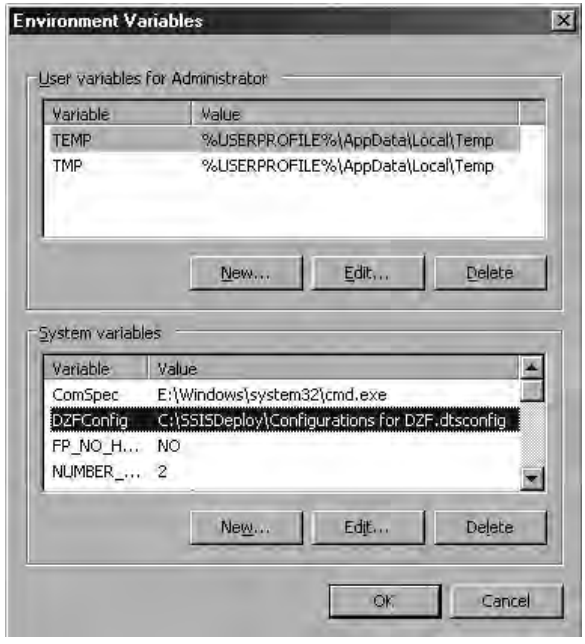


Figure 13-8 *Creating an environment variable for indirect configuration*

settings. So if BIDS is already open, exit from it and then start it again to load the newly created the environment variable.

16. Open the Downloading zipped files with Indirect Configurations project using BIDS on the development computer.
17. Right-click the Downloads Connection Manager in the Connection Managers area and choose Properties from the context menu. In the Properties window, click in the Expressions field and then click the ellipsis button to open the Property Expressions Editor. Click in the field under the Property column and select ConnectionString from the drop-down list. Click the ellipsis button next to the Expressions field to build an expression.

    In the Expression Builder dialog box, expand Variables and then drag the User::fpath variable to drop in the Expression box. Click OK twice to return to the Properties window, shown in Figure 13-9.

Exercise (Create Indirect Package Configuration)

Till now you have created a configuration file to update the value of the fpath variable and the property expression to use fpath to update the ConnectionString property of the Downloads Connection Manager. Also, you have moved the configuration file to

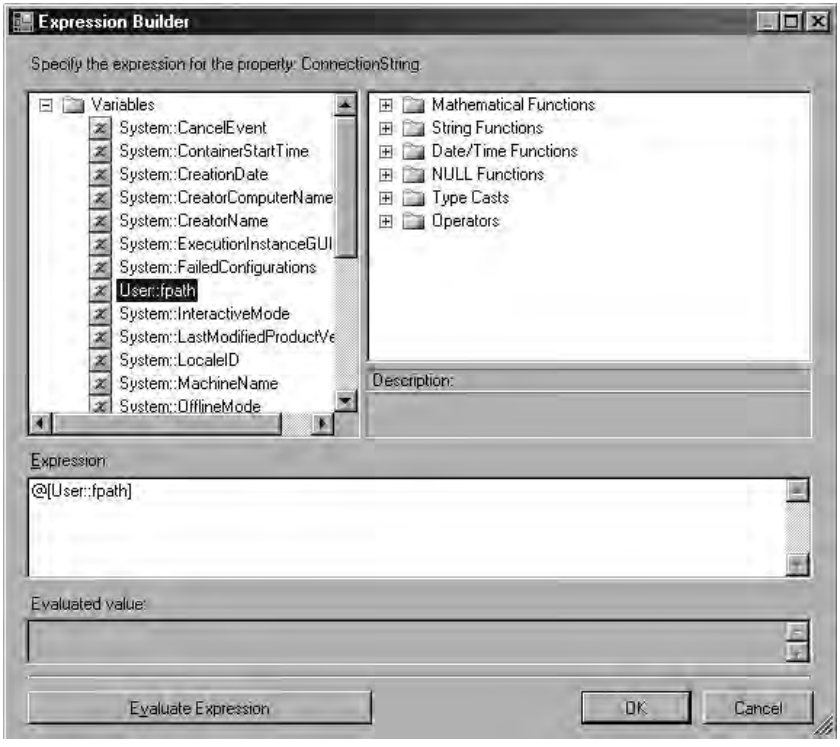


Figure 13-9 *Creating a property expression for ConnectionString property*

a different computer, which is now not accessible to your package. You have created an environment variable that points to the already-created configuration file. In this part you will point your package to read the configuration file from the location specified by the environment variable you created earlier. To do this, you will use an indirect configuration.

18. In BIDS, right-click anywhere on the blank surface of the Designer and choose Package Configurations from the context menu. In the Configurations list box, you will see the previously configured Downloaded files path configuration listed there. Click Remove to delete this configuration.
19. Click Add to create a new configuration. Click Next on the Welcome screen of the Package Configuration Wizard.
20. In the Select Configuration Type screen, choose XML Configuration File in the Configuration Type field. Next, click the radio button for "Configuration location is stored in an environment variable." So instead of specifying an XML

filename, you will be using an environment variable here. Select DZFConfig from the drop-down list in the Environment Variable field (see Figure 13-10). You are telling your package that you are using an XML configuration file to define your configurations whose location information is located in the environment variable DZFConfig. Click Next.

21. Type **DZFConfig file connection** in the Configuration Name field on the Completing The Wizard screen. Note that the Preview section shows the Type as Indirect Configuration File and the Environment Variable Name as DZFConfig. Click Finish to complete the wizard and return to the Package Configurations Organizer. The newly created configuration gets added to the Configurations list.
22. Click Close to return to the Designer surface. With the creation of this configuration, you have completed all the configuration steps required for indirect package configurations. Pointing the package to read information from an environment variable doesn't result in creation of any external file; rather, this information is embedded in the package itself. This enables you to copy your package to any computer and the package will look to the environment on the computer to get the location information for the configuration file from environment variable. Save all the files and exit from BIDS.

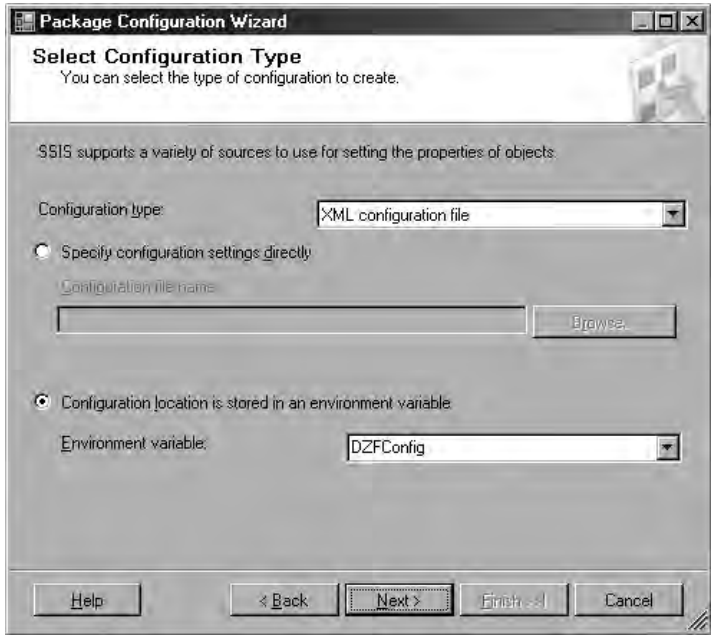


Figure 13-10 *Using an environment variable to create indirect configurations*

Exercise (Deploy and Execute the Package)

In the final exercise, you will deploy your package to the computer where you want to run it, and execute it to see how the package behaves with the indirect package configurations.

23. Copy the Downloading zipped files.dtsx file from the C:\SSIS\Projects\Downloading zipped files with Indirect Configurations folder on the development computer to the root folder of the C: drive on the deployment computer. (Actually, you can choose any folder on the deployment computer to host your package file.)
24. Open the command prompt window and type the following command:

    ```
    dtexec /file "Downloading zipped files.dtsx"
    ```

    You will see the package executed successfully with a DTSER_SUCCESS message, as shown in Figure 13-11.
25. Go to C:\SSISDeploy folder and notice that the two zipped files have been downloaded in that folder.

Review

You have learned how to use indirect package configurations to simplify the deployment or redeployment process to a mere copy operation in situations for which you must deploy to many machines or you must deal with complex requirements of applying configurations. During package loading time, as the configurations have been enabled, the package knows that it has to load configurations from an XML configuration file, which can be accessed from the path specified by the environment variable. So, the package goes to the environment variable DZFConfig, which directs it to the C:\SSISDeploy\Configurations

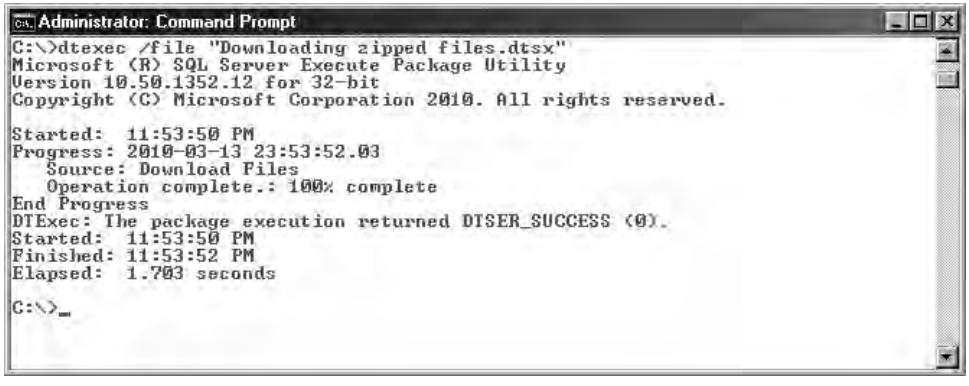


Figure 13-11 *Using the dtexec utility to execute the package on the deployment computer*

for DZF.dtsConfig file. On reading this file, the package sets the fpath variable value to C:\SSISDeploy. And during the package execution time, the property expressions read the value of the User::fpath variable and update the ConnectionString property of the Downloads Connection Manager and hence copy the zipped files to C:\SSISDeploy folder. As a last point on running the package on the deployment computers, you can actually run SSIS packages on a computer that doesn't necessarily have SQL Server installed. All you need to run SSIS packages on a computer is the .NET framework and SSIS run-time environment installed.

Deployment Utility

After you have created package configurations for the package, you are ready to deploy your package to the various computers that are going to do the work. You might not understand the difference between copying a package and deploying a package. Truth is, there's no difference if your package is simple enough that it has no configuration file or miscellaneous files that have to be taken care of. Deploying a package using the tools provided with SQL Server takes into consideration the package configuration file and any other miscellaneous files when pushing the package to the deployment computer. The following methods can be used to deploy a package to a computer:

- Using the import and export facility of Integration Services within SQL Server Management Studio
- Saving a package to the file system or to the SQL Server store
- Copying or moving packages using the dtutil utility
- Using a package deployment utility
- Using a custom deployment tool or home-grown scripts to deploy packages

You have used the first three methods in Chapter 6 when you were learning about administration of Integration Services. This approach is suitable for simple single-package projects or when you develop a package on the machine where you want to use it. This approach is not much help when you are dealing with some or all of the following situations:

- The project contains several small packages as modules of a parent package.
- You need to deploy Integration Services projects to several computers.
- You will use package configurations to set the properties.
- You will store miscellaneous files within the package.

Under these conditions, you will be using either a package deployment utility, which can deploy multiple-package projects along with package configurations and miscellaneous files or custom-developed scripts that could be simple windows batch files to do the trick. A sample of a simple batch file will be discussed later in the chapter.

A package deployment utility is used after you've created a package and package configurations, and are ready to deploy. To create a deployment utility, you need to configure the Deployment Utility section of the project properties before building the project. Three properties here can be configured to control deployment of your project. Once you configure these properties, you can create a package deployment utility by building a project on the computer where the Integration Services project is stored. The build process automatically includes the packages, package configurations, and miscellaneous files in the project and adds a manifestation file that is gracefully called the Deployment Utility.

AllowConfigurationChanges

You can assign a True or False value to this property. The True value lets you update configurations during deployment of the package; False lets you omit the update option during deployment. You can use this feature when you have to distribute a deployment utility to remote administrators for deploying the project, but do not want them to be able to make changes to configurations. This is also useful when you are redeploying packages after making a few changes, but do not want configurations to be changed.

CreateDeploymentUtility

This property also has True or False value options. A True value lets the build process create a deployment utility for the Integration Services project; a False value prohibits the creation of deployment utility. You need to specify a True value if you want to create a deployment utility, as the default value of this property is False.

DeploymentOutputPath

When you create a new project using BIDS, a bin subfolder is also created along with other files within the project folder. The default path shown in this property—i.e., bin\Deployment—refers to the same bin folder under the project. When you build the project, the packages, the package configurations, miscellaneous files, and the deployment utility file are added in this folder. You can change this path to create your deployment utility at a central location within the organization.

Deploying Integration Services Projects

The last step in the Integration Services project deployment journey is the deployment of the project. After having created packages for the project, package configurations, and the deployment utility, there isn't much left other than actually running the deployment utility to deploy the Integration Services project.

As you create the deployment utility, a manifestation file with the extension SSISDeploymentManifest is created along with packages, configurations, and miscellaneous files in the folder specified in the DeploymentOutputPath property of Integration Services project. You need to copy this deployment folder to the deployment computer and execute the SSISDeploymentManifest file to install the Integration Services project. Executing the SSISDeploymentManifest file starts the Package Installation Wizard, which then guides you through the process of package installation.

Next you have to decide where you want to install the package. If you choose to install your package in the file system, the package and other files such as configurations and miscellaneous files will be saved to the specified folder on the file system. However, if you choose to install the Integration Services package in the SQL Server, the package will be saved in the sysssispackages table in the MSDB system database and the other files will still be saved to the file system in the folder specified.

During the life cycle of a package, you may need to modify the package either to add functionality or to accommodate changes happening somewhere else in the environment. In such cases, you may be making changes to your project in the development environment and then testing in the test environment. Once those changes have been tested successfully, you are again facing the task of deploying your project. This redeployment differs slightly from the first deployment of the project. You may want to change the deployment process during redeployment—e.g., you may not want to make any change to package configurations during deployment—and so you will set the AllowConfigurationChanges property to False. Second, if you've made changes only to some of the packages and do not want to redeploy all the packages within a project, you can create a new Integration Services project and add only the packages to the project that you want to redeploy. When you add a package to an Integration Services project, the configurations associated with the package are automatically added to the new project and you don't have to worry about them.

The following Hands-On exercise lets you create a deployment utility and then install the Integration Services package to the deployment computer.

Hands-On: Deploying an Integration Services Project

Create a package deployment utility for the Contacting Opportunities with Configurations project and finally deploy the package.

Method

Deploying a project involves going through three simple steps:

- Configure the Integration Services project properties.
- Build an Integration Services project.
- Install the Integration Service project.

Exercise (Configure Project Properties)

Before you can create a deployment utility, you must configure the project properties. These properties provide control over the deployment process.

1. Open Contacting Opportunities with Configurations project using BIDS.
2. In the Solution Explorer, right-click the Contacting Opportunities with Configurations project and choose Properties from the context menu.
3. In the property pages, click the Deployment Utility under Configuration Properties on the left pane. Specify True in the AllowConfigurationChanges and CreateDeploymentUtility properties (see Figure 13-12). Leave the default bin\Deployment path in the DeploymentOuputPath property. Click OK to close the property pages.

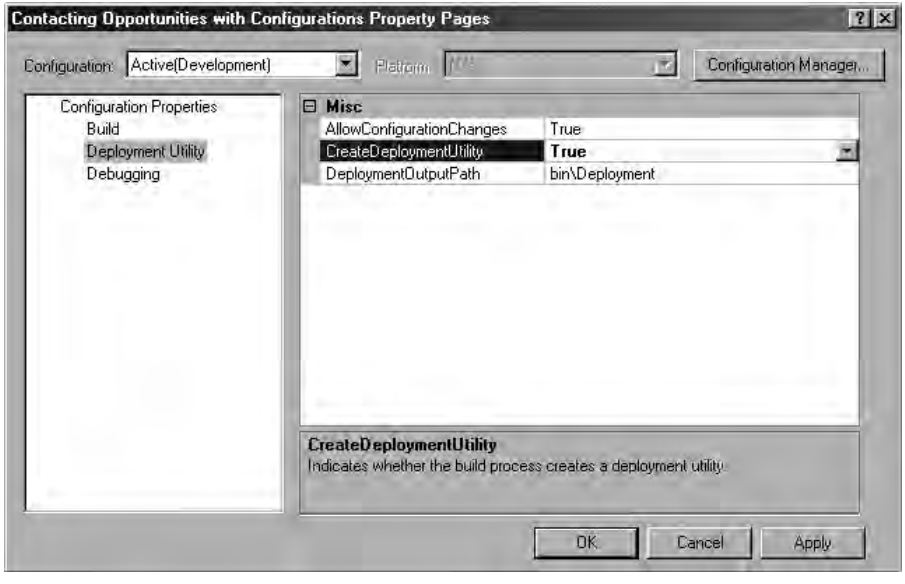


Figure 13-12 *Configuring Project properties*

Exercise (Build the Project)

Building an Integration Services project is quite easy using the Solution Explorer in BIDS.

4. In the Solution Explorer, right-click the Contacting Opportunities with Configurations project again and choose Build from the context menu. Alternatively, you can also do this from the Build menu.
5. You will see a success message in the bottom-left corner in BIDS as well as in the Output window in BIDS.
6. Using Windows Explorer, you can see that the deployment files have been added in the folder C:\SSIS\Projects\Contacting Opportunities with Configurations\bin\Deployment. These files consist of conoppo.dtsConfig, the package configurations file; Mailing Opportunities with Configurations.dtsx, the package XML file; and Contacting Opportunities with Configurations.SSISDeploymentManifest, the package deployment utility. Copy the Deployment folder to the deployment computer in the C:\Temp folder.

Exercise (Install and Run the Package)

After creating the deployment utility, you can install your package using the DTSInstall .exe command-line tool or by double-clicking the SSISDeploymentManifest file. If you want to use the DTSInstall.exe tool, you have to specify the complete path and filename for the manifest file while invoking Package Installation Wizard.

7. On the deployment computer, go to the C:\Temp\Deployment folder using Windows Explorer and double-click the Contacting Opportunities with Configurations.SSISDeploymentManifest file. This will start the Package Installation Wizard after a few seconds. Click Next on the Welcome screen.
8. On the Deploy SSIS Packages screen, you can choose between File System Deployment and SQL Server Deployment of the package. Select the File System Deployment radio button. Click Next.
9. Click Browse and browse to C:\SSISDeploy folder, and then click Make New Folder to create a Contacting Opportunities with Configurations folder. Select this folder and click OK to return and see C:\SSISDeploy\Contacting Opportunities with Configurations listed in the Folder field. Click Next to go to the next screen. Click Next again in the Confirm Installation screen.
10. In the Configure Packages screen, expand Property under the Configurations section. You can see the two properties, Subject and ToLine, listed there. You can modify the value of any configuration here by clicking in the Value field of the configuration. Click Next when you're done.

11. In the Finish the Package Installation Wizard screen, review the summary that tells you about the package to be deployed, the configuration file to be used, the folder where it will be installed, and the log filename. Click Finish to deploy the package.
12. Browse to the C:\SSISDeploy\Contacting Opportunities with Configurations folder using Windows Explorer. You will see the Mailing Opportunities with Configurations.dtsx file in the folder.
13. Open the command window, go to C:\SSISDeploy\Contacting Opportunities with Configurations directory, and type the following at the command prompt:

    ```
    dtexec /file "Mailing Opportunities With Configurations.dtsx"
    ```

    You will see the various steps being completed and scroll through the window, finally returning a DTSER_SUCCESS message.

Review

To deploy a project, you need to configure its deployment properties and make sure that the CreateDeploymentUtility property is set to True. Once the deployment utility is created, you can install the package by running the deployment manifest file. You can also invoke the Package Installation Wizard using the DTSInstall.exe tool. You have also seen that if the AllowConfigurationChanges property is set to True during deployment utility creation, you get an opportunity to modify package configurations during deployment of the package. Make sure you use this aspect when you are deploying to several computers with different environments, if you are not using indirect package configurations.

One of the benefits of using configurations is the ability to pass them on the cmd line when executing a package. This is very handy and practical, especially when you are executing a package on the same server that is acting both as a test and development server with different configurations. In this way, you can choose which configuration file to pick up, as environment variables will not work in this case, due to their inability to point to both versions of the configuration file.

Custom Deployment

Sometimes the deployment procedures are guided by already-adopted deployment standards that might be different from what SSIS deployment tools can provide. Also, in the bigger picture, you might need to deploy lots of other stuff along with packages and you find that the available tools don't provide you an ability to deploy all of the changes in one go. In such instances, you might prefer to adopt custom-developed scripts to deploy SSIS packages and the other changes together. Typically, your deployment stuff will be made up of the following entities:

- SSIS package files
- SSIS configuration files
- SQL scripts to populate configurations if you are using SQL Server–based configurations
- SQL scripts to create or apply changes to SQL Server databases, tables, or other objects
- SQL scripts to create SQL Server Agent jobs
- Rollback scripts

One simple example could be batch file scripts that could be used to deploy all of these entities. A sample of various parts of such a batch file is listed here:

```
REM*****************************************************************
REM Set Variables for Control file
REM*****************************************************************
SET DB_SERVER = DatabaseServerName
SET SSIS_SERVER = SSISServerName
SET CONTROL_DATABASE = ControlDatabaseName
SET ENVIRONMENT = PRODUCTION

REM*****************************************************************
REM Use the following to copy SSIS packages from a Network Share on to the
Production SSIS Server
REM*****************************************************************
XCOPY "\\ServerName\DeploymentShareName\%Environment%\SSISPackages\*.*" "\\%SSIS_
SERVER%\F$\DeploymentFolder\SSISPackages\" /Y

REM*****************************************************************
REM Use the following to deploy an SSIS package from a folder to the MSDB database
on the SSIS Server. You do not need to use this step if you are using File System
deployment of packages.
REM*****************************************************************
dtutil /Q /FILE "\\%SSIS_SERVER%\F$\DeploymentFolder\SSISPackages\Package1.dtsx" /
DESTS %SSIS_SERVER% /COPY SQL;"/SSISPackages/Package1"

REM*****************************************************************
REM Use the following to run a TSQL script on the database Server to populates
configurations to an SQL Server table, Create SQL Server database objects or an
SQL Server Agent job.
REM*****************************************************************
SET INPUT_FILE= \\%DB_SERVER%\F$\DeploymentFolder\SQLScripts\
SetSSISConfigurations.sql
SET OUTPUT_FILE= \\%DB_SERVER%\F$\DeploymentFolder\SQLScripts\LOG_OutPut.txt
SQLCMD -S %DB_SERVER% -d %CONTROL_DATABASE% -E -i %INPUT_FILE% -o %OUTPUT_FILE%
```

So, using this or probably extending the previous example, you can carve out a reasonably simple solution that deploys the changes both in SSIS and in SQL Server databases all in one go.

Summary

You have completed an interesting journey to deploy Integration Services packages. You have created package configurations, used direct and indirect methods to specify configurations for a package, created a deployment utility, and deployed a package using the Package Installation Wizard. You now understand the difference between copying a package and deploying a package, and you're now able to create indirect configurations to make the deployment process as easy as a copy operation.

Chapter 14

Migrating to Integration Services 2008

In This Chapter

- Upgrade Advisor
- Migrating Data Transformation Services Packages
- Upgrading Integration Services 2005
- Summary

As you have started this chapter, I can safely assume that you have worked with DTS 2000 and perhaps are looking to migrate to Integration Services. And if you have completed previous chapters and have worked with various components of Integration Services such as Control Flow tasks, Data Flow transformations, connection managers, variables, property expressions, and package configurations, you can very well appreciate that Integration Services is totally different from Data Transformation Services of SQL Server 2000. In fact, Integration Services is not an upgrade of DTS; rather, it is altogether a new tool. This chapter will cover migration of both DTS 2000 and Integration Services 2005 to Integration Services 2008.

For one thing, Data Transformation services have been deprecated in SQL Server 2008, though 32-bit management, run-time, and design-time support for DTS packages have still been provided in SQL Server 2008. With that in mind, you may be wondering what exactly you can do to your existing DTS 2000 packages. Integration Services provides options that you can use to migrate DTS packages into SSIS. Later, in the exercises in this chapter, you will use a DTS 2000 package provided in the RawFiles folder. Before you start checking out the options, you'll need to analyze your packages to determine whether they can be migrated; if they can be, you need to be aware of issues that might come up during migration. You can analyze your DTS 2000 packages using the Upgrade Advisor. Let's start the chapter with a discussion of the Upgrade Advisor, and then we will look at the migration options.

Upgrade Advisor

As a DBA of SQL Server 2000 or SQL Server 2005, very soon you might find yourself face to face with having to upgrade to SQL Server 2008, if you have not upgraded already. The reason could be an end of support for legacy software, or you may want to use the enhanced performance and new features provided in SQL Server 2008. Microsoft provides a starting point for help in the form of an Upgrade Advisor and recommends that you run it to analyze how your upgrade project will go before you get started on any project upgrades.

The Upgrade Advisor is used to analyze the configurations of SQL Server 2005 or SQL Server 2000 components and identifies issues that you must address to ensure a successful upgrade. The reports generated by this tool let you plan how to deal with issues during an upgrade and the migration process. You can install the Upgrade Advisor from the SQL Server 2008 product media, or you can download it from Microsoft's downloads center. Be sure to go through the system requirements for installing the Microsoft SQL Server 2008 Upgrade Advisor.

In the following Hands-On exercise, you will install the Upgrade Advisor and then run it to analyze your DTS 2000 packages.

Hands-On: Analyzing DTS 2000 Packages with SQL Server 2008 Upgrade Advisor

You need to analyze your existing DTS 2000 packages before upgrading them to Integration Services format to learn about any issues that may come up during migration.

Method

In this Hands-On exercise, you will be using the Upgrade Advisor to analyze an existing package, Importing Contacts.dts, provided in C:\SSIS\RawFiles. This exercise has two main parts:

- Installing the Upgrade Advisor
- Running Upgrade Advisor and viewing an analysis report

Exercise (Install Upgrade Advisor)

You will use SQL Server 2008 product media to install the required software in the following steps. However, if you don't have the product media, they can be freely downloaded from Microsoft Download Center.

1. Run setup from the SQL Server 2008 media and click the Install Upgrade Advisor link in the Planning page. The installation is straightforward and simple (Figure 14-1).

Exercise (Run the Upgrade Advisor to Analyze DTS 2000 Packages)

You will run the Upgrade Advisor to analyze the Importing Contacts.dts package and will see the report.

2. Choose Start | SQL Server 2008 R2 Programs Group and start the SQL Server 2008 R2 Upgrade Advisor. You'll see the Welcome page, which provides links to documentation, the latest updates, and two links for Launch Upgrade Advisor Analysis Wizard and Launch Upgrade Advisor Report Viewer. These two components are the main components of the Upgrade Advisor.
3. Click the Launch Upgrade Advisor Analysis Wizard and click Next on the Welcome screen of the Analysis Wizard.
4. Select the components you want to analyze in the SQL Server Components screen. For this exercise, make sure you select the Data Transformation Services check box as shown in Figure 14-2. You can also use the Detect button to let Upgrade Advisor detect and select the services installed on the specified computer. Click Next to move on.

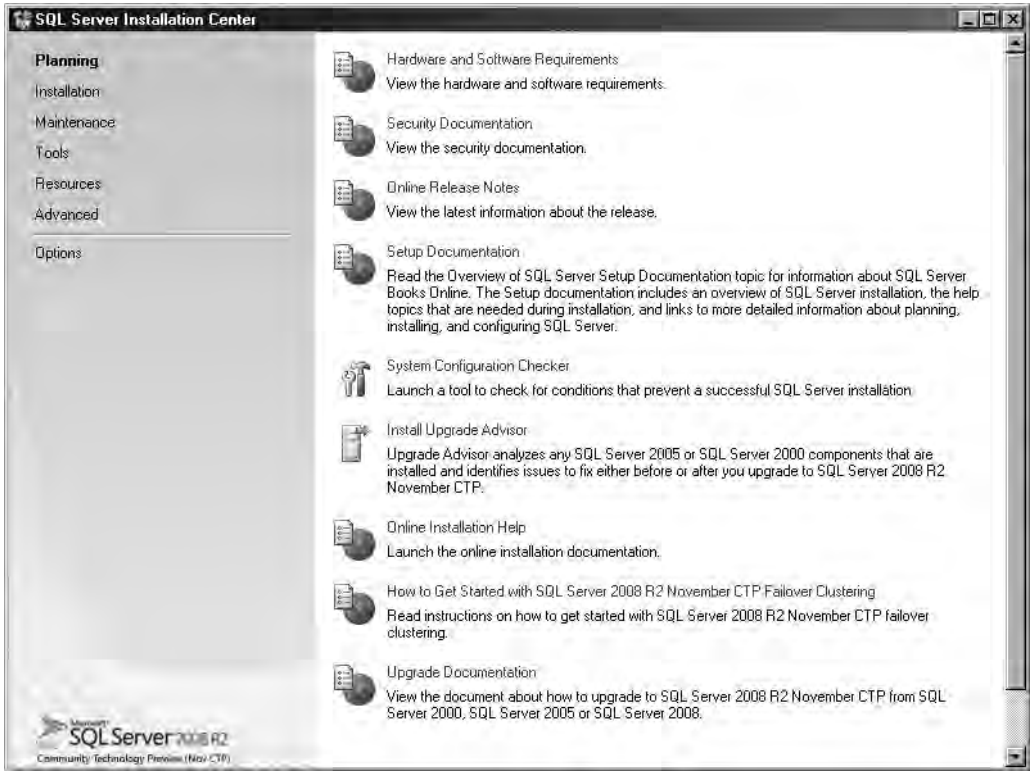


Figure 14-1 *Starting installation of the Upgrade Advisor*

5. Type or select an Instance Name from the drop-down list box in the Connection Parameters screen. Specify the authentication details for the selected instance and click Next when ready. Don't get confused between versions of SQL Server. The Upgrade Advisor connects to SQL Server 2000 to analyze and report, so specify an instance of SQL Server 2000 here.
6. On the DTS Parameters screen, you can select whether you want to analyze DTS 2000 packages that are saved in the SQL Server or the packages saved on the file system. Select the Analyze DTS Package Files radio button and then type **C:\SSIS\RawFiles** in the Path To DTS Packages field. Click Next to move on.
7. Review the settings in the Confirm Upgrade Advisor Settings screen. Click Run to analyze the selected package.
8. In the next screen, the Upgrade Advisor will analyze the package; when it completes analysis, you can see the report by clicking Launch Report. The

Figure 14-2 *Selecting Data Transformation Services for analysis*

analysis report will be displayed in the Upgrade Advisor Report Viewer. The key messages that you can spot in the report among other messages are

- That you need to install SQL Server 2000 DTS Designer Components to edit the packages in Integration Services.
- Packages stored in Meta Data Services are not supported.
- SQL Server 2000 Data Transformation Services has been deprecated.

Review

Depending on the complexity and the components used in the package, you will see different messages for your packages. But you will find that the previously mentioned three key messages are common to all types of packages, as these are generic messages representing major differences between DTS and SSIS.

Migrating Data Transformation Services Packages

In this part of the chapter you will study the options available to you to migrate your existing DTS 2000 packages to Integration Services 2008. You will work with a test package that has been developed in Data Transformation Services (DTS) and will understand the requirements of different methods and the end results achieved that can affect your choice of the option. But first of all, let's start with the options you have for DTS 2000 packages.

Migration Options

After having run Upgrade Advisor for DTS packages, you can view the report that will tell you whether your package can or cannot be upgraded, given the complexity and the components used in the package. At that time, you need to consider the options available for the DTS 2000 packages that are still in use. Some preconditions will affect the option you choose.

While considering crossing the migration bridge, one thing you need always to bear in mind is that Integration Services is new software that has been developed from the ground up. Though it has adopted most of the DTS 2000 concepts, it is totally different in usability and operation; above all, not all DTS 2000 tasks are available in Integration Services. Some of them have been deprecated, and some have been removed. But the good news is that Integration Services can coexist with DTS and also provides necessary tools for you to migrate.

You also need to be aware of the fact that not all versions of SQL Server 2008 include Integration Services. SQL Server 2008 Standard Edition has basic Integration Services transformations, whereas the Developer, Enterprise, and Datacenter Editions have advanced transformations built into them. The remaining versions (Express, Workgroup, and Web Editions) do not have Integration Services, though the Express Edition has an Import and Export Wizard utility (still, it is not the Integration Services service). While discussing your options, we will focus only on the editions that have Integration Services built into them.

Another consideration is the storage location of your package. You were able to store your DTS 2000 packages in SQL Server, Structured Storage File, or in Meta Data services, also known as the repository. Integration Services can access DTS 2000 packages that were saved in SQL Server 2000 or Structured Storage File for executing or migrating, but Integration Services does not support Meta Data services. If you have DTS 2000 packages saved to Meta Data services, you can export the packages from Meta Data services to SQL Server or Structured Storage File for Integration Services easy

access. However, if you cannot do this, you will have to install SQL Server 2000, SQL Server 2000 tools, or repository redistributable files on your local computer to enable Integration Services to scan and migrate the packages saved in Meta Data services.

At the time of upgrading your database server, you might have to migrate all your packages and test them on the SSIS environment before you can commission an SQL Server 2008 migration project. This alone can be a deterrent to the upgrade project and can cause long delays. The following options are available while moving to Integration Services:

- Running DTS 2000 packages as-is with run-time support
- Embedding DTS 2000 packages in Integration Services packages using a wrapper task
- Migrating DTS 2000 packages to Integration Services using the Package Migration Wizard

Installing DTS 2000 Support Components

Before we get into exploring the options available for handling DTS 2000 packages, let's install the components you will require to work with legacy packages. You can install DTS 2000 run-time and design-time support components on 32-bit systems, whereas 64-bit systems have limited support. There is no 64-bit design-time or run-time support for DTS packages. However, if you install the 32-bit version of dtexec utility, you can run DTS 2000 packages in 32-bit mode on a 64-bit computer. On Itanium-based operating systems, however, you cannot even do that, as there is no support for DTS packages. Hence you cannot create, view, modify, or run DTS packages on Itanium-based operating systems. If you need to modify DTS 2000 packages, you should be using 32-bit systems. The details and exercises covered in this chapter refer to 32-bit systems. You need to install multiple components to provide support to DTS 2000 packages in Integration Service. The following are the steps involved in installing a complete support for Data Transformation Packages:

- **Basic Installation** When you choose Integration Services in the Feature Selection page during installation, the ActiveX Script task and the DTS Package Migration Wizard are installed. As this has been already installed in our test environment, you don't need to do anything for this step. ActiveX task is really provided to support existing scripts that have been developed in DTS. You should not develop new scripts using this task, as it is marked deprecated and will be removed in future releases.

- **Clients Tools Backwards Compatibility** When you choose Clients Tools Backwards Compatibility feature in the Feature Selection page during installation, Execute DTS 2000 Package task is installed. However, you also need to install the DTS run time in order to use this component. Run SQL Server 2008 installation and choose the Client Tools Backwards Compatibility feature in the Feature Selection page as shown in Figure 14-3.
- **DTS run time** To install DTS run-time support, search for the sqlserver2005_bc.msi file in the SQL Server installation media. On the media I'm using, the file exists under 1033_enu_lp\x86\setup\x86 folder. Double-click this file to install SQL Server 2005 Backward Compatibility software, which also installs the DTS run-time engine (refer to Figure 14-4). Complete the installation wizard.
- **DTS Designer Components** SQL Server 2000 DTS Designer Components allow you to edit and maintain DTS 2000 packages until they can be migrated to Integration Services. Download SQL Server 2000 DTS Designer Components

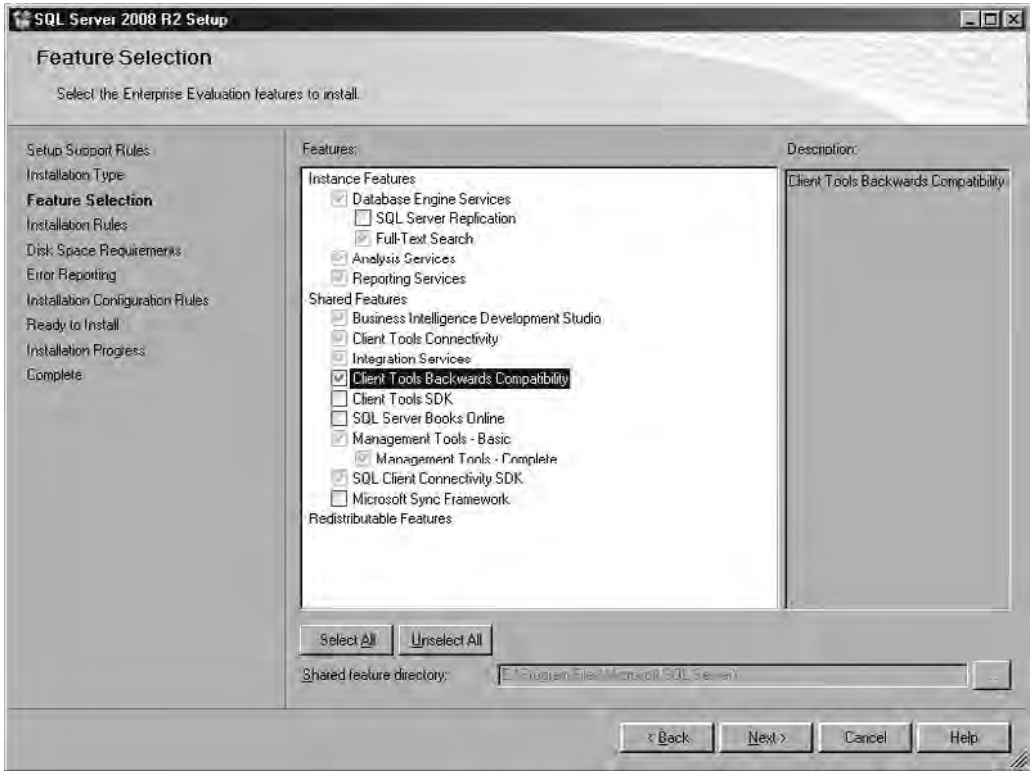


Figure 14-3 *Installing the Client Tools Backwards Compatibility feature*

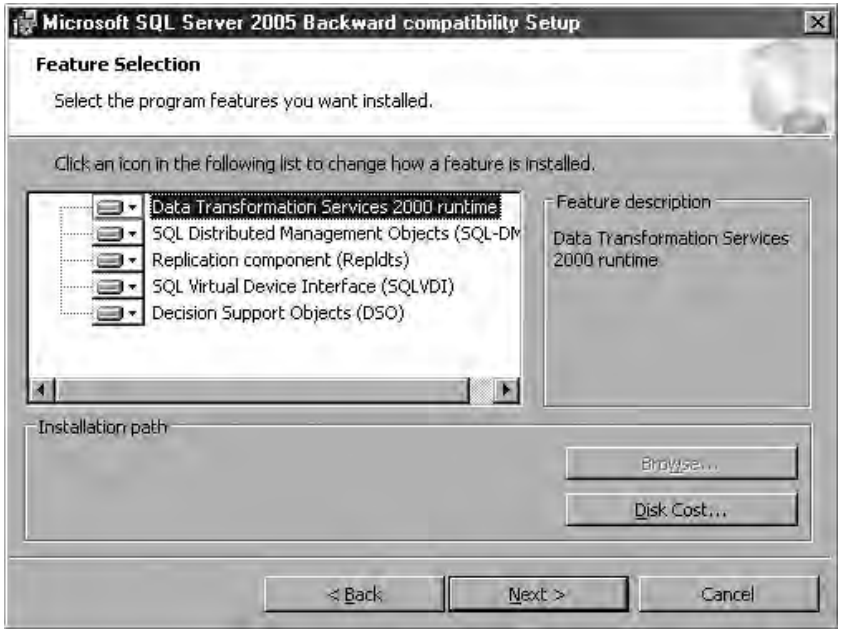


Figure 14-4 *Installing Data Transformation Services 2000 run time*

from the Microsoft Download Center. This software has been released with the SQL Server 2005 Feature pack. From the Feature Pack for SQL Server 2005 page, locate the Microsoft SQL Server 2000 DTS Designer Components section and download the SQLServer2005_DTS.msi file. Once the download is complete, double-click the SQLServer2005_DTS.msi file to start the installation wizard (see Figure 14-5) for DTS Designer Components. Click Next on the Welcome screen. Installation is straightforward. Go ahead and complete the installation.

It is worth mentioning couple of issues with these legacy software components. As various versions of SQL Server and the underlying operating systems have been released, the components also needed to be revised time to time, though they are still released with the SQL Server 2005 feature pack. You need to install the latest version available at the time. I've used the ones released in December 2008 that have worked on Windows 2008 with SQL Server 2008 R2. Earlier versions didn't work with Windows 2008 due to Data Execution Prevention (DEP) technology. Both the latest run time and the designer components have been included in the code files of this chapter. Another issue was observed during installation the path to the sqlgui.dll library. For DTS you need to use the legacy file that resides in C:\Program Files\Microsoft SQL Server\80\Tools Binn\ and not the one that resides in C:\Program Files\Microsoft

Figure 14-5 *Installation Wizard for SQL Server 2000 DTS Designer Components*

SQL Server\100\Tools Binn\. To avoid using new sqlgui.dll, you need to modify the path system variable in the advanced properties of the computer and move the former listed path before the later in the path variable value.

After installation of these components, you are now ready to explore the earlier discussed options in detail and see how well they fit with your scenario.

Running DTS 2000 Packages As-Is with Run-Time Support

If you are busy dealing with other stuff while migrating to SQL Server 2008, you may want to leave your DTS packages running as is and focus on migration later. You can run the DTS 2000 run-time environment and Integration Services on the same computer. However, some of the databases to which DTS 2000 package connects might get upgraded to SQL Server 2008, leaving DTS 2000 unable to access them. To deal with this issue, the run-time software has been enhanced to an updated version of the DTS 2000 run-time environment that is capable of accessing SQL Server 2005/2008 databases.

In the following Hands-On exercise, you will discover how you can manage DTS 2000 packages in SQL Server 2008.

Hands-On: Executing a DTS 2000 Package

The scenario is that you have upgraded your SQL Server 2000 instance to SQL Server 2008 and have installed Integration Services and the DTS 2000 run time. Here you will learn how you can manage and run DTS 2000 packages.

Method

A DTS 2000 package, Importing Contacts, is available in a C:\SSIS\RawFiles folder saved in a structured storage file. This package imports records from the C:\SSIS\RawFiles\Contacts.txt flat file to the Contacts table of the Campaign database. Contacts.txt is a fixed-width file, and the Importing Contacts package uses a flat file source, an OLE DB provider for SQL Server, and a Transform Data task to import data into SQL Server.

Starting with a structured storage file, you will investigate how the package can be managed and modified in SQL Server Management Studio based on the various storage locations. The steps involved are as follows:

- Enumerate and import the package to SQL Server using SQL Server Management Studio.
- Export and execute the DTS 2000 package.

Exercise (Enumerate and Import the DTS 2000 Package in SQL Server Management Studio)

When you save DTS 2000 packages to SQL Server 2000, you know they are saved to the sysdtspackages table of the MSDB database. These packages are visible in the Local Packages node under Data Transformation Services in Enterprise Manager. In this part, you will explore how SQL Server Management Studio enumerates legacy packages.

1. Start SQL Server Management Studio and connect to the local SQL Server 2008 database engine.
2. In the Object Explorer window, expand Management | Legacy | Data Transformation Services. Your legacy packages—i.e., DTS 2000 packages—will show up under this node.
3. Right-click the Data Transformation Services and choose Import Package File from the context menu, as shown in Figure 14-6.

   In the Import DTS 2000 Package dialog box, choose Importing Contacts.dts Package from C:\SSIS\RawFiles and click Open.

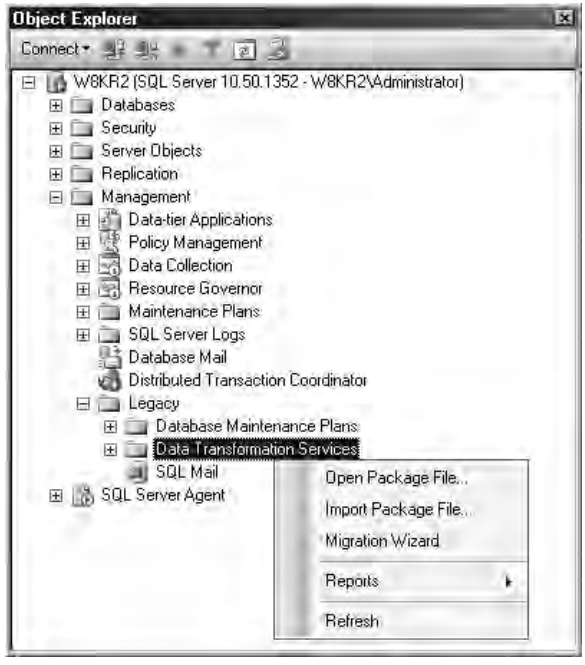


Figure 14-6 *Importing a DTS 2000 package in SQL Server Management Studio*

4. Because a structured storage file can have more than one package and package version, you are asked to choose the package you want to import in the Select Package dialog box. Expand the Importing Contacts package and choose any of the versions; then click OK to select your package. These packages are the same, just saved a few seconds apart.
5. Go back to Data Transformation Services node to see that the Importing Contacts package has been added there.
6. Click New Query and run the following query in the New Query tab:

   ```
   SELECT * FROM msdb.dbo.sysdtspackages
   ```

   You will see the Importing Contacts package listed in the output results along with any other packages that you might have in sysdtspackages table. This means that all the DTS 2000 packages are still saved in the msdb.dbo.sysdtspackages table and can be added to this table using the import facility provided by SQL Server Management Studio.

   You can enumerate DTS 2000 packages with SQL Server Management Studio and import and export DTS 2000 packages from the sysdtspackages table. However, if you need to modify the DTS 2000 packages, you must have SQL Server 2000 tools on the computer or take them away to a computer where SQL

Server 2000 is installed. This is not a very helpful situation. To overcome such issues, Microsoft has provided SQL Server 2000 DTS Designer Components so that you can modify DTS 2000 packages from SQL Server Management Studio. You have already installed these components earlier; however, if you have skipped that step, then this is the time to install them.

Exercise (Edit and Execute a DTS 2000 Package)

You will use the SQL Server 2000 DTS Designer components to edit the Importing Contacts package and then run the package.

7. In the SQL Server Management Studio, go to the Data Transformation Services node and right-click the Importing Contacts package. Select Open from the context menu to open the Importing Contacts package in the DTS 2000 Package Designer, as shown in Figure 14-7.
8. In the DTS 2000 Package Designer, double-click the Campaign to open the Microsoft OLE DB Provider for SQL Server Connection Properties and provide the name of your server in the Server field or specify (local) to refer to the local server. The provided name SARTH is the name of computer used in setup lab for this book. Change it to your computer name. In the Database field, select Campaign (Figure 14-8). Click OK to close this dialog box. Do not select the check box in the Task References dialog box, and click OK to close it.

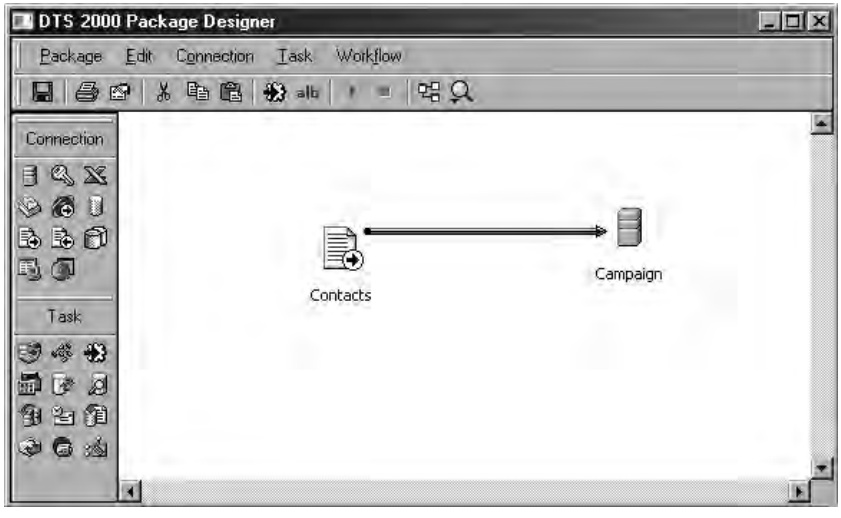


Figure 14-7 *The Importing Contacts package in the DTS 2000 Package Designer*

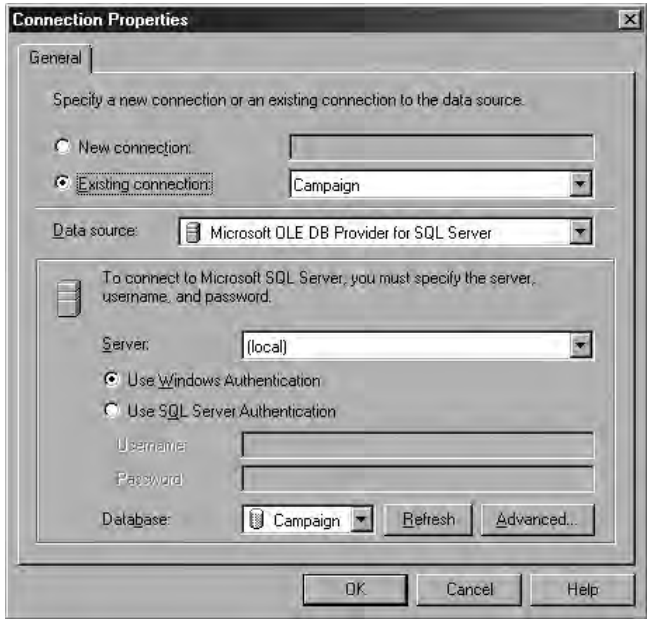


Figure 14-8 *Connection Properties for the Campaign database*

9. You can also execute your package in this Designer in the usual way by clicking Execute. Save your package and close the Designer.
10. Using Windows Explorer, browse to the C:\SSIS\RawFiles folder and delete the Importing Contacts.dts package.
11. Now, go back to SQL Server Management Studio and right-click the Importing Contacts and choose Export from the context menu.
12. Specify C:\SSIS\RawFiles as the path to export Importing Contacts package.
13. Choose Start | Run; type **cmd**; and click OK. At the command prompt, type the following:

    ```
    C:\>dtsrun /F "C:\SSIS\RawFiles\Importing Contacts.dts"
    ```

    You will see that the package executes successfully and 49 rows are added to the Contacts table, as shown in Figure 14-9.

    Integration Services lets you run DTS 2000 packages as-is using the DTSRun.exe utility. For those who haven't used Data Transformation Services of SQL Server 2000 and are facing migration challenges now, DTSRun.exe is the command-line utility used to run DTS 2000 packages. The jobs created in the SQL Server Agent to run DTS 2000 packages also use DTSRun to execute the jobs. This also means that all the jobs that were created in the SQL Server Agent will keep on

```
Administrator: Command Prompt

C:\>dtsrun /F "C:\SSIS\RawFiles\Importing Contacts.dts"
DTSRun:  Loading...
DTSRun:  Executing...
DTSRun OnStart:  DTSStep_DTSDataPumpTask_1
DTSRun OnProgress:  DTSStep_DTSDataPumpTask_1; 49 Rows have been transformed or
copied.; PercentComplete = 0; ProgressCount = 49
DTSRun OnFinish:  DTSStep_DTSDataPumpTask_1
DTSRun:  Package execution complete.

C:\>
```

Figure 14-9 *Running a DTS 2000 package*

running as is. If you who have extensively used DTSRun.exe and DTSRunui.exe, note that Integration Services provides equivalent, rather better utilities called dtexec and dtexecui. The dtexecui utility can be used to create commands for use with dtexec. Refer back to Chapter 6, where these utilities are covered in detail.

Review

In this exercise, you used SQL Server Management Studio to enumerate the packages stored in the sysdtspackages table of the MSDB database. You also used the import and export facility provided by SQL Server Management Studio and explored other available options. The important thing you learned is that Microsoft has provided a DTS 2000 Designer package as a separate download in the Feature Pack for SQL Server 2005 that you can install and use to edit existing DTS 2000 packages. This makes life much easier if you need to manage migration of DTS packages to Integration Services. In the next part of this chapter, you will learn how to include your existing DTS 2000 packages in Integration Services projects.

Embedding DTS 2000 Packages in Integration Services Packages

Integration Services lets you to create business solutions using a modular design so that you can build your package as a small unit of work to provide a piece of functionality that can be repeatedly used in various enterprise solutions. You can integrate these smaller units of work to form a complete solution using the Execute Package task. You learned about the Execute Package task in Chapter 5 and used it to build a solution in one of the Hands-On exercises as well. This is relevant if you are developing a new piece of functionality. What about the work that has been developed already using DTS 2000 packages? Do you have to develop that functionality in Integration Services also?

Various scenarios may guide you to adopt a particular solution, but if you want to use your DTS 2000 package as a module in your Integration Services package, Integration Services provides a wrapper task that lets you do this, and that wrapper task is called the Execute DTS 2000 Package task.

When you installed the Clients Tools Backwards Compatibility feature earlier, the Execute DTS 2000 Package task was also installed, though you may not have realized, it as it is not added in the Toolbox by default. Later in the following exercise you will add this task to the Toolbox in BIDS.

Execute DTS 2000 Package Task

Using the Execute DTS 2000 Package task, you can include a DTS 2000 package in your Integration Services project and run it using the DTS run-time engine. This task can be considered similar to the Execute Package task with the differences that this task runs DTS 2000 packages instead of SSIS packages and uses DTS run-time engine instead of SSIS run-time engine. However, both tasks are used to in the same way for the running child packages inside a parent package and even can be used together in one package without worrying about run-time conflicts, as both use different run-time engines that can coexist. Other benefits of using this task are modularity, reusability, and security.

Let's work through a quick Hands-On exercise to demonstrate usage of the Execute DTS 2000 Package task.

Hands-On: Executing Importing Contacts Using the Execute DTS 2000 Package Task

In this exercise, you will run a DTS 2000 package—the Importing Contacts.dts package—that is saved to the file system in the C:\SSIS\RawFiles folder using the Execute DTS 2000 Package task.

Exercise (Configure the Execute DTS 2000 Package Task)

1. Start BIDS and create a new Integration Services project with the following details:

Name	Executing Importing Contacts
Location	C:\SSIS\Projects

2. When the new project has been created, go to Solution Explorer and rename the Package.dtsx package as **Importing Contacts.dtsx**. Click Yes to accept to rename.

3. Right-click in the Toolbox on any task and select Choose Items from the context menu. This opens up a Choose Toolbox Items dialog box where you can add or remove components in to Visual Studio. Click SSIS Control Flow Items tab and select Execute DTS 2000 Package task as shown in Figure 14-10. Click OK to add this task in to the Toolbox.
4. From the Toolbox, drop the Execute DTS 2000 Package task on to the Control Flow Designer surface.
5. Double-click the task to open the Execute DTS 2000 Package Task Editor.
6. In the General page, click in the StorageLocation field under the Location section to enable the drop-down arrow. Note that the drop-down list shows three options: SQL Server, Structured Storage File, and Embedded in Task, as shown in Figure 14-11.

   Depending on the storage location of the DTS 2000 package, choose the relevant option, and based on the option you choose, the editor's interface changes to show only the relevant fields. When you choose the SQL Server option, the task will access packages stored in the sysdtspackages table of the MSDB database of the SQL Server you specify in the SQLServer field. If you choose the Structured Storage File option, the package will access the packages from the file stored on the file system. You can specify the path for this file in the File field. The third option, Embedded In Task, means the DTS 2000 package has been saved

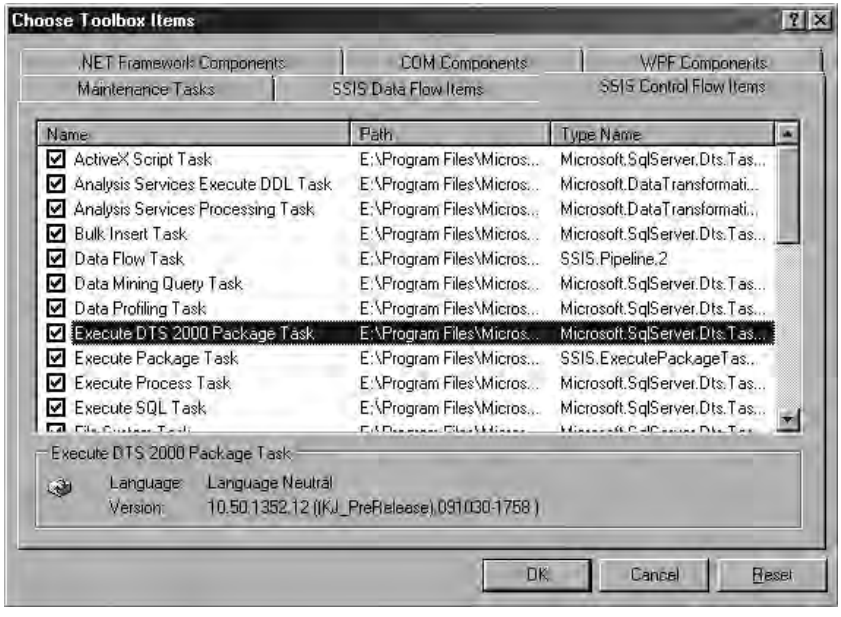


Figure 14-10 *Adding the Execute DTS 2000 Package task into the Toolbox*

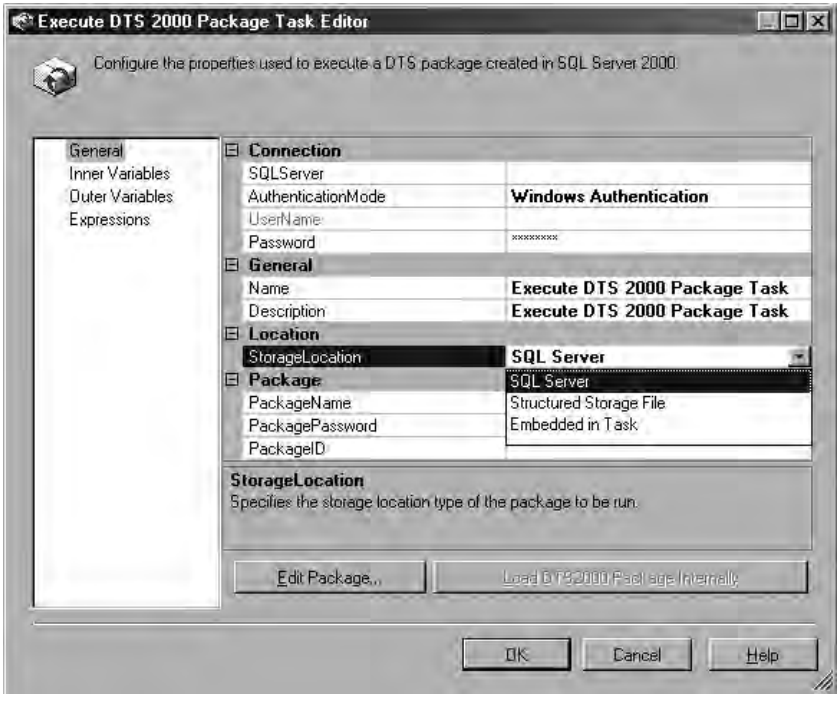


Figure 14-11 *Storage locations supported by Execute DTS 2000 Package task*

internally in the Execute DTS 2000 Package task and you don't have to specify the connection information of file or SQL Server for the DTS 2000 package. You can load the DTS 2000 package in this task by clicking Load DTS2000 Package Internally. To load the package, you first select either SQL Server or Structured Storage File location in the Location field and specify the relevant connection information in the Connection area. After choosing the PackageName, you can then click Load DTS2000 Package Internally and the package will be loaded in the task. When you save this project, the DTS 2000 package is saved within the task. This task will hold all the information required for executing the DTS 2000 package after that and removing the DTS 2000 package from original location wouldn't affect the execution of this task.

7. For this exercise, select Structured Storage File in the Location field.
8. Specify C:\SSIS\RawFiles\Importing Contacts.dts in the File field.
9. Type the following in the General section:

Name	Importing Contacts
Description	This task is used to execute the named DTS 2000 package.

10. Click in the PackageName field and then on the ellipsis button to open the Select Package dialog box. As the structured storage file can contain multiple packages and package versions, you are asked to choose the package you want this task to execute. Select a version for the Importing Contacts.dts package and click OK. The PackageID field will show the unique identifier for the selected version of the package.
11. As the Importing Contacts package doesn't have a password, leave the PackagePassword field as is. The task should look like Figure 14-12.
12. You can edit the specified package by clicking Edit Package. The DTS 2000 Package Designer will open, where you can edit the package using legacy tools. If you haven't installed the SQL Server 2000 DTS Designer components earlier, you will get an alert message asking you to install the Designer components. Close the DTS 2000 Package Designer window.
13. Go to Inner Variables page. Here you can specify and configure the inner variables that the DTS 2000 package uses. The variables and their values specified will be passed to the global variables in the DTS 2000 package. You can use the New button to add a new variable for which you can specify a type and can assign a value as well. This value will be used by global variables as the updated value at run time.

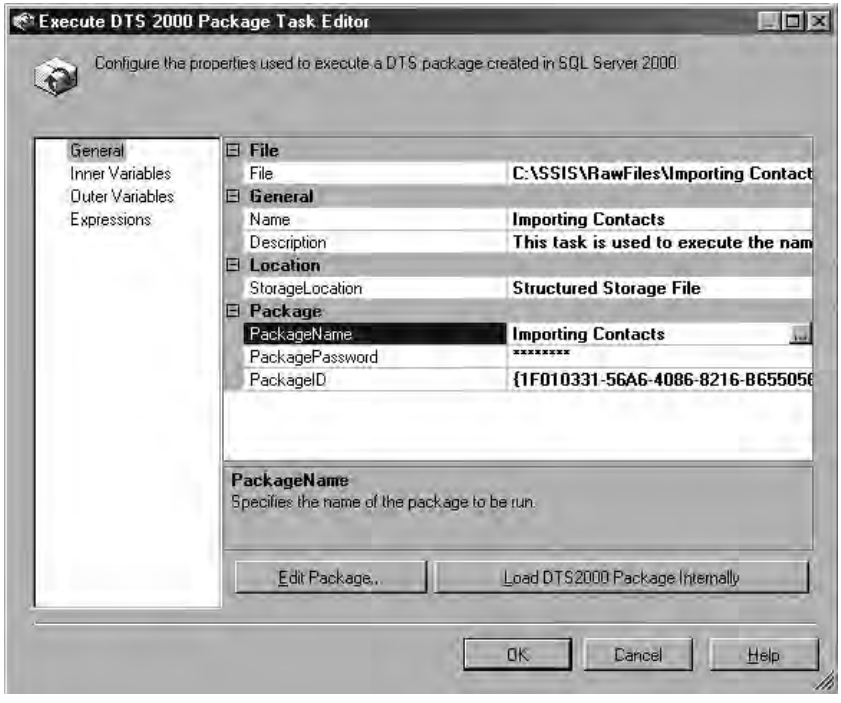


Figure 14-12 *Configurations for the Execute DTS 2000 Package task*

14. Go to Outer Variables page, where you can add an outer variable to the DTS 2000 package. Outer variables are used in the parent package—i.e., in an Integration Services package. When you click New, a blank row is added under the Name field, in which you can select a variable from the drop-down list of system and user-defined variables in the parent package.
15. Using the Expressions page, you can write an expression for any of the property for this task to update dynamically at run time.
16. Click OK to close this task and press SHIFT-CTRL-S to save the project.
17. Press F5 to execute the package. The task will change color to green and you will receive a success message in the Output window.
18. Press SHIFT-F5 to stop debugging and close the project.

Review

In this exercise, you learned another way of executing your DTS 2000 package using the Integration Services Execute DTS 2000 Package task. You can also include your legacy package as a child package in a parent SSIS package using this task. You also learned that you can use inner and outer variables to pass updated variables to the legacy package and can save DTS 2000 packages internally in the Execute DTS 2000 Package task. The option of loading and saving DTS 2000 packages internally is useful during migration of DTS to SSIS when dealing with custom tasks developed in DTS 2000. As the DTS 2000 custom tasks will not have a directly mapped task in Integration Services, the migration process encapsulates such tasks inside the Execute DTS 2000 Package task using the Embedded in Task feature and adds it in the migrated package.

Migrating DTS 2000 Packages to Integration Services

Once you understand the issues involved in migrating a DTS package to Integration Services, you can begin migrating most of your packages. To make life easier, Integration Services provides the Package Migration Wizard to help you migrate packages to the SSIS format.

Some of the Data Transformation Services tasks didn't fit well in the design and development of Integration Services software. This led to the removal of those tasks from Integration Services. Other tasks were developed into Integration Services. These tasks map directly and are converted to Integration Services tasks during migration, while custom DTS tasks or tasks that have been removed from Integration Services are either encapsulated in the Execute DTS 2000 Package task or replaced with a placeholder task. The following discussion tells you how the various components of SQL Server 2000 Data Transformation Services are mapped to Integration Services components and what happen when they are migrated to Integration Services.

- **ActiveX Script task** The SSIS version of this task is able to run most of the DTS ActiveX scripts; however, the scripts that reference DTS package objects may not successfully run on migration and you may have to edit the code manually. This task has been marked deprecated in SSIS and is provided to support existing DTS scripts.
- **Analysis Services task** Though Integration Services has two built-in tasks, namely the Analysis Services Execute DDL task and the Analysis Services Processing task, neither of them directly map to the DTS 2000 Analysis Services task. This is because Integration Services tasks do not work with SQL Server 2000 Analysis Services. The migrated package contains the encapsulated functionality that must be manually migrated to Integration Services.
- **Bulk Insert task** Directly maps to Bulk Insert task of Integration Services.
- **Copy SQL Server Objects task** Migrates to Transfer SQL Server Objects task.
- **Data Driven Query task** This task does not get migrated straight away, as it has no directly mapped task in Integration Services. The migrated package contains the encapsulated functionality of this task, which you have to migrate manually using Data Flow components based on the functionality provided by this task.
- **Data Mining Prediction task** Directly maps to Data Mining Query task; however, in some cases you may find that the Data Mining Prediction task of SQL Server 2000 gets encapsulated in a placeholder task which you have to convert to a Data Mining Query task manually.
- **Dynamic Properties task** There is no direct mapping task in SSIS to the DTS 2000 Dynamic Properties task that is used to modify the DTS components dynamically at run time. This functionality has been achieved in Integration Services by the use of package configurations and property expressions. You learned about property expressions in Chapter 3 and package configuration in Chapter 13. On migration, this task is replaced by a placeholder task, which you have to convert manually using package configurations, property expressions, and variables in Integration Services.
- **Execute Package task** This is migrated to the Execute DTS 2000 Package task in Integration Services.
- **Execute Process task** This is converted to the Execute Process task in Integration Services after migration.
- **Execute SQL task** Maps directly to the Execute SQL task in Integration Services.
- **File Transfer Protocol task** Maps directly to the FTP task in Integration Services.

- **Message Queue task** This task maps directly to the Message Queue task in Integration Services.
- **Send Mail task** Gets converted to Send Mail task of Integration Services.
- **Transform Data task** This task has no direct mapping in Integration Services and gets encapsulated in the Execute DTS 2000 Package task on migration. The functionality provided by this task has been distributed among various data flow components in Integration Services. You must manually convert this functionality to the Integration Services format.
- **Various transfer tasks** The SQL Server transfer tasks get converted to various transfer tasks in Integration Services. These tasks have direct mapping tasks.
- **Passwords** As the Integration Services has a different security architecture than Data Transformation Services, the passwords used to protect DTS packages don't get migrated. However, you can configure the package protection level after the package has been migrated to Integration Services. For more details on security features, refer to Chapter 7.
- **Variable** DTS 2000 packages contain only global variables, whereas Integration Services has a much enhanced variables architecture. For example, in an SSIS package you can use system variables, create user-defined variables, assign scopes and namespaces to variables, attach property expressions to variables to update their values dynamically at run time, and configure them to raise an event when their values are updated. On migration, the DTS 2000 package variables get migrated to the package scope in the user namespace. You can enhance the use of variables in your package after migration.
- **Connections** The connections in the DTS 2000 package get migrated to connection managers in Integration Services for the tasks that get directly migrated. For the tasks that use connections and cannot be directly migrated, such as the Transform Data task, the connection stays the part of the intermediate package that encapsulates DTS 2000 task.

It should be clear to see that the migration from DTS 2000 to the SSIS package may involve some development work due to the tasks that did not find their place in Integration Services. Removing these tasks from Integration Services does not mean that the functionality has been lost; rather, the functionality provided by these tasks has been enhanced. But the enhanced features have been distributed among various other components that must be used together to create the required functionality.

Depending on the complexity of your package and the tasks it contain, your package may get migrated without any additional effort or may migrate some of the tasks while encapsulating the others in the Execute DTS 2000 Package task, requiring only a bit of effort on your part to migrate completely; or they may not get migrated at all, and you have to then decide whether to use them with the DTS run-time support or rewrite the packages.

In the following section, you will work with the Package Migration Wizard, the tool that you can use to migrate an SQL Server 2000 Data Transformation Services package to an Integration Services package.

Package Migration Wizard

Integration Services provides this tool that guides you step-by-step to migrate packages stored in SQL Server or a structured storage file. The Package Migration Wizard reads the DTS package and creates a new copy of the migrated package in Integration Services. It does not alter the source package, which stays available as is, if the migration process fails. The Package Migration Wizard can be started using one of the following methods:

- From SQL Server Management Studio, right-click the Data Transformation Services node under the Legacy node of Management in the Object Explorer (see Figure 14-6) and choose Migration Wizard from the context menu.
- From BIDS, right-click the SSIS Packages node in the Solution Explorer and select Migrate DTS 2000 Package from the drop-down list box.
- From the command prompt, go to the folder C:\Program Files\Microsoft SQL Server\90\DTS\Binn and run the executable DTSMigrationWizard.exe.

Let's do a short Hands-On exercise on using the Package Migration Wizard.

Hands-On: Migrating Importing Contacts to Integration Services

Migrate the DTS 2000 package Importing Contacts into Integration Services format.

Method

You will use the Package Migration Wizard to migrate the package, and after migration is completed, you will execute the package to see the results.

Exercise (Use the Package Migration Wizard)

In the first part of this Hands-on, you will create a new project and run the Package Migration Wizard by right clicking the SSIS Packages node. This will not only migrate the package but will add it in the project as well.

1. Start BIDS and create a new Integration Services project with the following details:

Name	Migrating Importing Contacts
Location	C:\SSIS\Projects

2. When BIDS loads the blank project, go to the Solution Explorer window, right-click the SSIS Packages node, and choose Migrate DTS 2000 Package from the context menu (see Figure 14-13).
3. This will invoke the Package Migration Wizard, and you will see the first screen of this wizard. Click Next to move on.
4. The Choose Source Location screen allows you to select the location where your package is stored. Click the down arrow in the Source field to see the list of source locations. The Package Migration Wizard can read DTS 2000 packages stored in Microsoft SQL Server or in a structured storage file. As mentioned, SQL Server 2005 does not support Meta Data services (also called the repository). If your DTS 2000 package is stored in Meta Data services, you may prefer to save those packages to the locations supported by the Package Migration Wizard. In case you cannot do that, the Package Migration Wizard provides a way to read packages from Meta Data services when SQL Server 2000, SQL Server 2000 tools, or the repository redistributable files are installed on the local computer. For this exercise, select Structured Storage File in the Source field.

Figure 14-13 *Starting the Package Migration Wizard*

5. Specify C:\SSIS\RawFiles\Importing Contacts.dts in the File Name field as shown in Figure 14-14. Alternatively, you can choose this file by clicking Browse. Click Next to move on.
6. In the Choose Destination Location screen, specify the folder where you want to save the migrated package. The migrated package is created as a new DTSX file, and your DTS 2000 package is left as is without any changes. In the Folder Name field, specify C:\SSIS\Projects\Migrating Importing Contacts as the folder where you want to save the migrated package (see Figure 14-15). You can also click Browse to choose this folder, which also provides you an opportunity to create a new folder if you need to. Click Next.
7. As a structured storage file can have multiple packages and package versions, you can select the packages and their versions you want to migrate in the List Packages screen. You can also modify the name of the migrated package by editing the Destination Package field for the selected package. Select the Importing Contacts package and change the Destination Package to **Importing Contacts (Migrated)** by directly typing it into the field (Figure 14-16). Click Next.

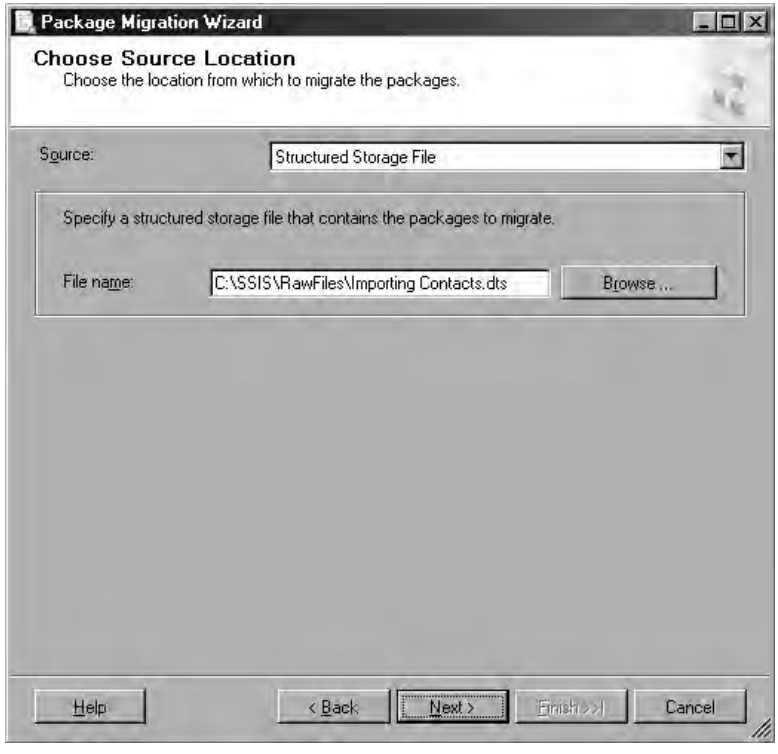


Figure 14-14 *Specifying the DTS 2000 package location*

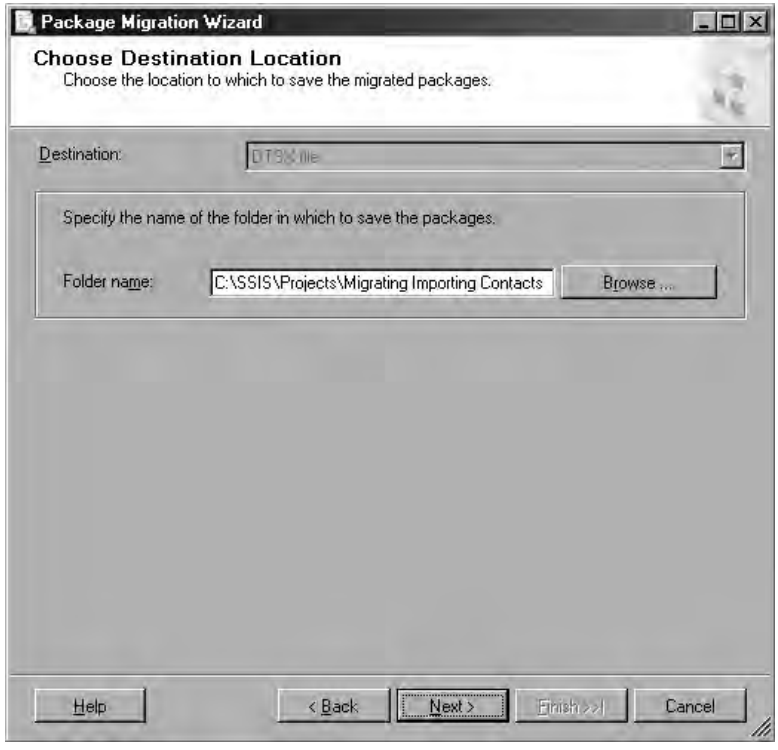


Figure 14-15 *Specifying a destination folder for the migrated package*

8. In the Specify A Log File screen, click Browse and specify Migrating Importing Contacts.Log in the File Name field after selecting the C:\SSIS\RawFiles folder and click Save. Select Yes to create this file. Then click Next.
9. The Complete The Wizard screen shows you summary information for all the options that you have selected in the previous screens. Note that this screen also tells you the number of packages that will be migrated and the number of packages that will not be migrated. Review the information and click Finish when ready.
10. You will see activities occurring while the package is being migrated, component by component in the Migrating The Packages screen. You can interrupt this process by clicking Stop if necessary. Once the package is migrated completely, you will see a success status for the package. At this stage you can see the report by clicking Report. Once you're happy, click Close to close the Package Migration Wizard.
11. As the Package Migration Wizard is closed, you will notice that a package by the name Importing Contacts (Migrated).dtsx has been added in the SSIS Packages

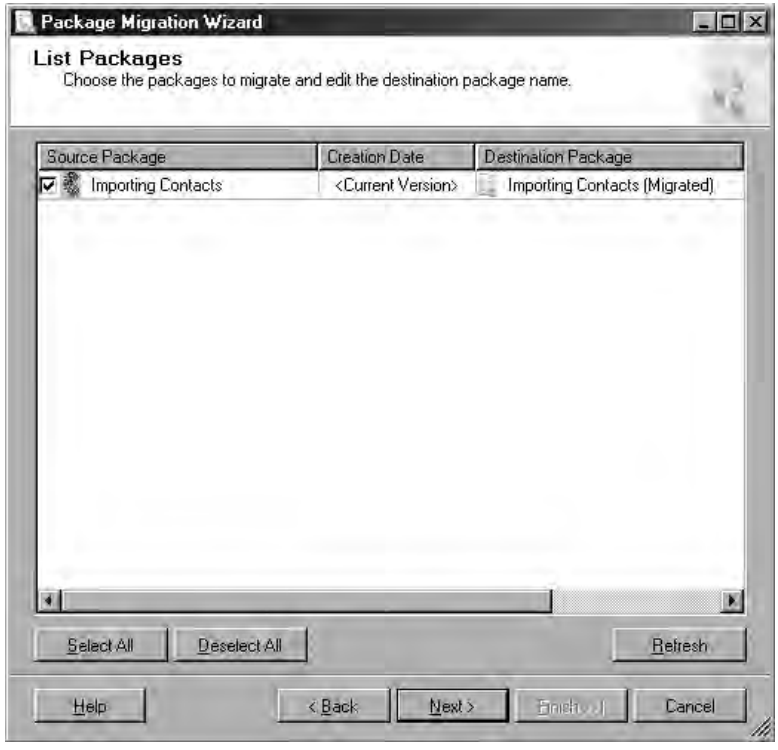


Figure 14-16 *Selecting DTS 2000 packages that you want to migrate*

node in the Solution Explorer window. You can delete the blank package Package.dtsx, as it is not required in this exercise.

12. Explore to the C:\SSIS\RawFiles folder and open the Migrating Importing Contacts.Log file using Notepad. This shows you details of how the migration progressed.

Exercise (Execute the Migrated Package)

In this final part, you will execute the migrated package.

13. Double-click the Importing Contacts (Migrated).dtsx package to open it. You will see a Data Flow task named DTSTask_DTSDataPumpTask_1 on the Control Flow Designer surface. Notice that this is the DTS 2000 data pump task name. Also, two connection managers were added: the OLE DB Connection Manager for connecting to the Campaign database, and a Flat File Connection Manager for connecting to the Contacts.txt file. Double-click the Data Flow task to open the Data Flow panel.

14. As the DTS 2000 package was quite simple, you will see a flat file source and an OLE DB destination adapters connected by a data flow path with a data conversion task in between in the Data Flow Designer surface. Press F5 to execute the package and check that it has been migrated successfully.
15. Press SHIFT-F5 to stop debugging the package. Go to the Data Flow tab and double-click the OLE DB Destination to open the editor. Review the settings in the Connection Manager page. Go to the Mappings page and note that though the input column names do not match with the destination columns, they have been mapped correctly. When you're done, click OK to close the editor.

Review

You've used the Package Migration Wizard to migrate a DTS 2000 package to Integration Services. You selected a DTS 2000 package stored on the file system in a structured storage file and saved the destination package in the DTSX format on the file system. In the end, you observed that though the package was migrated successfully and no error was reported, the settings do seem in need of tuning. The point to be noted is that the Package Migration Wizard can migrate most of the components successfully, yet it needs to be tested out and the migrated package might need configurations.

Upgrading Integration Services 2005

If you are facing the task to upgrade a previous version of Integration Services, you will find this section useful as you will learn about various options available to you when upgrading from Integration Services 2005. Depending upon how the server is currently configured and what you want to achieve, you will choose one of the various options available to you that are explained next.

Before we go ahead and look at pros and cons of each approach, let us understand what key changes have been implemented in SQL Server 2008 from the point of view of Integration Services 2008. If you save your packages to SQL Server, they get stored into the MSDB database. However, they are stored in different tables, depending on which version of the SQL Server or Integration Services you use. As you know, SQL Server 2000 uses the sysdtspackages table to store DTS packages and Integration Services 2005 uses the sysdtspackages90 table to store SSIS packages in the SQL Server 2005 MSDB database. With the SQL Server 2008 release, the word "dts" has been replaced in these tables with the more relevant word "ssis" and the table Integration Services 2008 uses to store packages is named as sysssispackages. Not only the storage table but other relevant tables used by Integration Services have undergone this change too—e.g., sysssispackagefolders and sysssislog tables. Understanding this difference alone will answer most of your questions on differences in behavior between the two versions of Integration Services.

The other key difference is made in BIDS, which uses Microsoft Visual Studio Tools for Applications (VSTA), replacing the Microsoft Visual Studio for Applications (VSA) environment. This will mean that all your scripts you have developed in SSIS 2005 need to be upgraded to the VSTA environment. You do not need to worry here, as VSTA converts VSA scripts for you. Having understood these changes, let us explore the options described.

Same-Server Installation

In this case both the SQL Server Database Engine and Integration Services are running on the same server currently and you wish to upgrade them. In this case you might choose from one of the following two options:

- Running side-by-side
- Upgrading both SQL Server and Integration Services

Running Side-by-Side

First of all, the good news is that Integration Services 2008 can coexist with Integration Services 2005 as it does with Data Transformation Services. If you install Integration Services 2008 side-by-side with its 2005 predecessor, you can run both versions of packages with their respective development environments. Considering the changes in Integration Services storage tables and the changes in BIDS as explained earlier, the following issues will come up:

- BIDS 2005 can maintain and develop SSIS 2005 packages, while BIDS 2008 can maintain and develop SSIS 2008 packages.
- BIDS 2008 can open and load SSIS 2005 packages, while BIDS 2005 cannot open SSIS 2008 packages. When opening an SSIS 2005 package in BIDS 2008, the SSIS 2005 package gets converted at loading stage into SSIS 2008 format. BIDS then works on this package format, and once this loaded and converted package is saved, it can't be opened using BIDS 2005.
- Much as described in the preceding point, when you try to run an SSIS 2005 package using the SSIS 2008 dtexec utility, the utility temporarily converts the package into SSIS 2008 version before running it. This happens only in memory and the original version is not changed.
- Due to differences in storage locations, Integration Services cannot use different versions of SQL Server to store packages, as the tables and metadata won't exist. That is, Integration Services 2008 needs SQL Server 2008 and Integration

Services 2005 needs SQL Server 2005 to store packages. This also means that you cannot connect to an SSIS 2005 instance from SSMS 2008. However, you can modify the configuration file so that you can manage SSIS 2005 packages from SSMS 2008. This also enables you to import packages into SQL Server 2008 from an instance of SSIS 2005, but not the other way around: using SQL Server 2005 Management Studio doesn't allow you to manage or connect to a higher version of Integration Services, nor does it allow you to import or export packages to SSIS 2008.

Upgrading Both SQL Server and Integration Services

Your next option is to upgrade both SQL Server and Integration Services, especially if they exist on the same server. As discussed earlier in the chapter, you can use the Upgrade Advisor to find out any issues with the upgrade and can fix them before you begin the upgrading process. When you run the upgrade process and choose to upgrade both SQL Server 2005 and Integration Services 2005 to their respective 2008 releases, the upgrade process performs a series of tasks that are important to understand.

- The upgrade process replaces the SSIS 2005 files with SSIS 2008 files and configures Integration Services to point to the new instance of SQL Server 2008.
- It creates new metadata, tables, and stored procedures for Integration Services in the MSDB database, removing the old msdb.sysdts*90 system tables and the system stored procedures used by SSIS 2005. The new tables, named as msdb.sysssis*, contain the required metadata for SSIS 2008.
- It creates three new fixed database-level roles: db_ssisadmin, db_ssisltduser, and db_ssisoperator for access control to SSIS packages. The SSIS 2005 roles of db_dtsadmin, db_dtsltduser, and db_dtsoperator are added as members to the corresponding new roles.
- The most important task it does from the user perspective is to move SSIS packages from old store locations to new store locations. Bear in mind that it can do this only for the locations it is aware of. So, it moves packages from the sysdtspackages90 system table to the sysssispackages system table for the packages that are stored in the MSDB database. And it moves packages stored in SSIS 2005 default SSIS package store, that is, the …\SQL Server\90 folder, to the new default location under …\SQL Server\100. Along with moving packages, it also moves the folder structure you have created in SSIS 2005 by moving folder metadata from the sysdtsfolders90 system table to the sysssispackagefolders system table. All other storage locations used to store SSIS 2005 packages remain as-is and no change happens there, as the upgrade process is not aware of them.

- While it moves packages to new storage locations, it does not migrate them to 2008 format. You will need to migrate them separately using the SSIS Package Upgrade Wizard. The sysssispackages table shows values in the packageformat column to indicate the version of the package. The packages that are still in the SSIS 2005 version will have a value of 2, whereas the packages that are in SSIS 2008 will have a value of 3. You can use the packageformat column to identify the current version of the packages when looking to upgrade existing packages.
- Last, the SQL Server Agent jobs that use dtexec utility directly will not run due to a change in the path of the dtexec utility in the 2008 release. You will need to modify these jobs yourself.

Different Server Installation

In this installation, you have a SQL Server database engine installed on one computer while the Integration Services package that connects to this database instance is installed on another computer. You might choose one of the following two options in such a scenario:

- Upgrading SQL Server Database Engine only
- Upgrading Integration Services only

Upgrading SQL Server Database Engine Only

In this scenario you upgrade only the SQL Server database engine that is used by Integration Services to store packages. The Integration Services process could be on a different server and remains on 2005 version. This shows your intention to keep using SSIS 2005 without upgrading packages even after an upgrade of SQL Server. Here are the limitations that apply in this case:

- You can still use BIDS 2005 to develop and modify existing Integration Services packages that are stored in the file system.
- The system tables used by Integration Services in the MSDB database get migrated to 2008 format as has been discussed in the preceding section, and the SSIS 2005 packages are moved to the new system table. This change renders Integration Services 2005 unable to open the packages stored in SQL Server. Actually, not only Integration Services but the other SQL Server 2005 tools on other computers, such as BIDS, SQL Server Management Studio and SQL Server 2005 Agent Jobs, cannot discover the packages on the upgraded instance of the database engine. That is, the packages are not available on the upgraded

instance of SQL Server from the perspective of the SSIS 2005 toolset. You should keep a copy of packages on the file system before upgrading the database engine if you intend to keep using SSIS 2005 even after an upgrade of the SQL Server database engine.

Upgrading Integration Services Only

In this scenario you upgrade only the Integration Services while the SQL Server database engine that is used to store packages is on another computer and continues to be the 2005 version. You intend to upgrade packages after the upgrade, though the SQL Server is not upgraded. Here are the limitations that apply in this case:

- The packages stored in the file system are available to Integration Services 2008 and can be easily upgraded using the SSIS Package Upgrade Wizard. However, the upgraded packages or the new SSIS 2008 packages cannot be saved to SQL Server 2005, as BIDS 2008 looks for the 2008 versions of system tables in the MSDB database.
- The packages stored in SQL Server 2005 cannot be accessed by BIDS 2008 due to different system tables. However, these packages can still be executed by SQL Server Agent jobs that run on SQL Server 2005, as no change has happened on the SQL Server.
- Later, if you want to upgrade the packages that are stored in SQL Server 2005, you will have to export them to a file system so as to enable BIDS 2008 to access them.

Upgrading SSIS 2005 Packages

By now you understand that the upgrade process moves the existing SSIS 2005 packages to new system tables or to the new SSIS Package Store, but does not upgrade them to 2008 format. You will have to manually upgrade these packages. Following are the tools that you can use and the consideration you need to be aware of while upgrading SSIS 2005 packages to SSIS 2008 format.

- If you open an SSIS 2005 solution or project in BIDS 2008, the Visual Studio Conversion Wizard starts automatically (Figure 14-17) and converts the solution or the project created in the previous version to the current version. This is in-place conversion, so you should take a backup or keep a copy in another location before proceeding further. If the solution or project is under source control, it

Figure 14-17 *Visual Studio Conversion Wizard*

will automatically be checked out during the conversion. Once the Visual Studio conversion completes, the SSIS Package Upgrade Wizard starts automatically.

- You can use the SSIS Package Upgrade Wizard to upgrade one or many SSIS 2005 packages (see Figure 14-18) together to SSIS 2008 format. This tool is installed when you install Integration Services 2008 and is available only in Standard, Enterprise, and Developer Editions of SQL Server. This wizard can also be started manually from one of the following places:
  - By right-clicking the SSIS Packages node in the Solution Explorer and selecting the Upgrade All Packages option
  - While connected to Integration Services in SSMS, by expanding the Stored Packages node and then right-clicking the File System or MSDB node and selecting the Upgrade Packages option
  - From Installation Center | Tools | Upgrade Integration Services packages
  - By running the SSISUpgrade.exe file from the command prompt after going to the C:\Program Files\Microsoft SQL Server\100\DTS\Binn folder

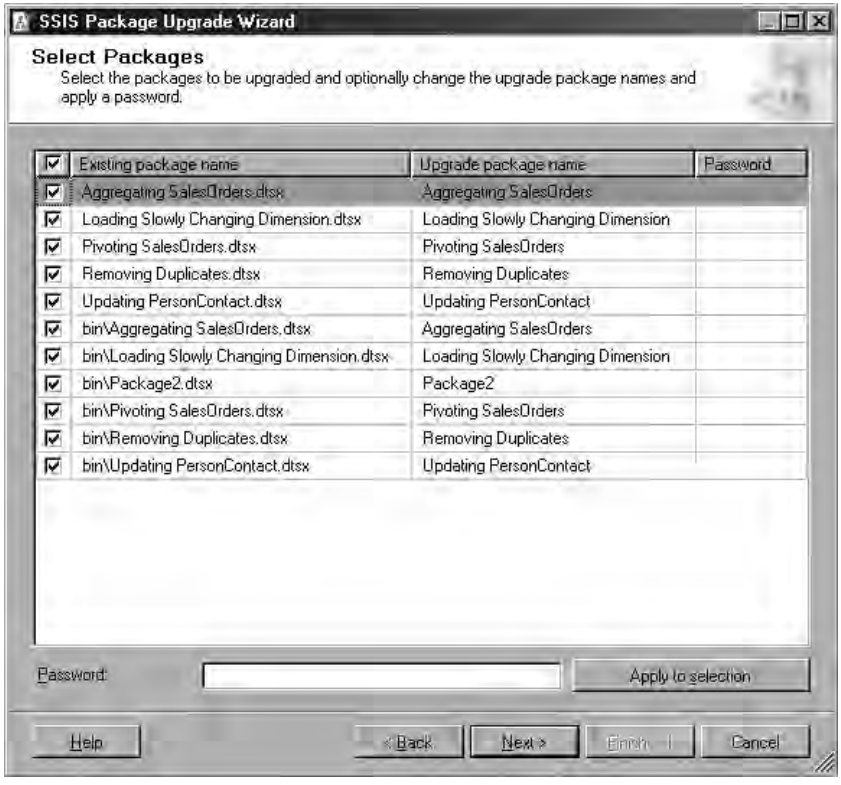


Figure 14-18 *Upgrading multiple packages with SSIS Package Upgrade Wizard*

This wizard offers the following Package Management Options during upgrade process:

- **Update connection strings to use new provider names** Updates the connection strings to use the SQL Server 2008 names for the OLE DB Provider for Analysis Services and the SQL Server Native Client provider. Note that the SSIS Package Upgrade Wizard updates only the connection strings that are stored in connection managers and not the ones that are constructed dynamically at run time.
- **Validate upgrade packages** Validates the upgraded packages and saves only those upgrade packages that pass the validation.
- **Create new package ID** Creates new package IDs for the upgraded packages.
- **Continue upgrade process when a package upgrade fails** Continues to upgrade the remaining packages when a package cannot be upgraded.

- **Backup original packages** When you are upgrading packages that are saved in the file system, it allows you to create backups of original packages during the upgrade process (Figure 14-19) so that you can continue to use existing packages if the upgrade process doesn't complete successfully. The original packages are backed up in a subfolder called SSISBackupFolder created by this upgrade wizard.

  After completing the upgrade wizard, the packages get migrated to new version and can no longer be converted back to SSIS 2005. If you need to use the older version, you will have to get a copy from the backup.

- Irrespective of the tool or the method you use, the upgrade process migrates the existing VSA compatible scripts in any Script task and Script component to Microsoft Visual Studio Tools for Applications (VSTA).

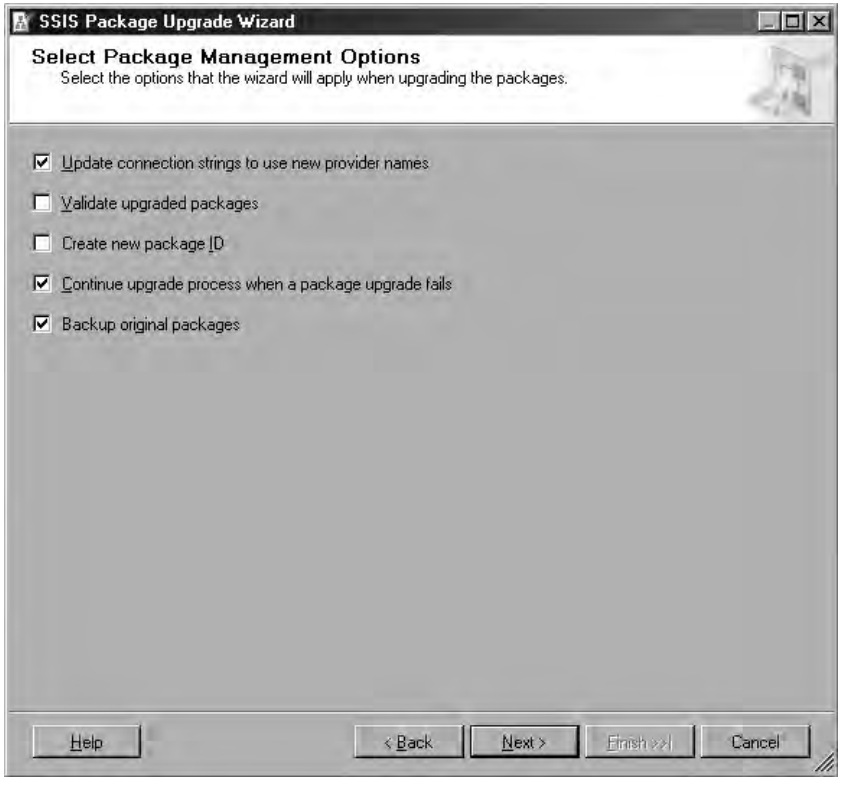


Figure 14-19 *Package Management Option with SSIS Package Upgrade Wizard*

- When you execute an SSIS 2005 package using SSIS 2008 dtexec utility, the utility temporarily migrates the package to SSIS 2008 format before execution. It does not save the upgraded package, however. If there is any issue with migration, the package will not be executed.
- When you open and load an SSIS 2005 package after adding it to a project in BIDS 2008, it automatically gets converted to SSIS 2008 format. If you save this converted package, the upgrade becomes permanent and cannot be reverted back.
- Most of the SSIS 2005 components seamlessly migrate to SSIS 2008 format, while some might have issues and could generate warning messages. For example, certain connection strings could generate warning messages while the Lookup transformation will migrate automatically, though you may prefer to re-configure the new Lookup transformation to take advantage of additional features.
- No custom components or third-party components will be migrated. You will have to manually recompile these components to enable them to work with SQL Server 2008 Integration Services.

Summary

Integration Services gives you the option of running your DTS 2000 package without making any changes using the enhanced run-time support. However, you may find yourself at a crossroads in deciding when to upgrade your packages, as Data Transformation Services has been marked deprecated and will not be supported in the future versions. You've also learned about the provision of the SQL Server 2000 DTS Designer components, the method to include or run your DTS 2000 packages in Integration Services projects using the Execute DTS 2000 Package task, and you have used the Package Migration Wizard to migrate a DTS 2000 package to Integration Services. One important thing you learned about migrating your DTS 2000 packages is that there is no straightforward method, though the Package Migration Wizard makes it easy to do; still, you may have to mend some configurations yourself. In the second part of the chapter you learned about migration options from SSIS 2005 and found that it is much easier compared to migrating from DTS 2000.

Chapter 15

Troubleshooting and Performance Enhancements

In This Chapter

- Troubleshooting Integration Services Packages
- Performance Enhancements
- Performance Monitoring Tools
- Using Parallel Processing
- Summary

You've come a long way with Integration Services and have worked with its various aspects to configure and develop SSIS packages. By now, not only you can develop workflow and data transformations in your packages but can also deploy them to an enterprise infrastructure securely with complete administration and control over the storage locations. In this chapter, you will learn the skills necessary for debugging your packages, in general and at the component level. You will then learn about the facilities that you can use to determine what happens at package run time to fix these issues accordingly. You must have come across various alerts and warning messages raised by Integration Services throughout your experience with package development. In this chapter you will explore how Integration Services validates component configurations in your packages and handles errors.

In the second half of the chapter, you will learn to configure your packages performing at peak levels. You'll explore the performance counters provided by Integration Services, custom events of the Data Flow task, execution trees and execution plans, among other options to help you keep memory usage in control. This chapter also offers a high-level discussion on package design techniques to optimize the pipeline and exploit the parallelism features. Let's start the last leg of our journey with Integration Services by learning to troubleshoot SSIS packages.

Troubleshooting Integration Services Packages

During package development you might have seen various alerts and warnings appear on the screen. For example, when you drop the Execute SQL task on the Control Flow surface, you see a crossed red glyph on the task indicating "No connection manager is specified." These validation alerts are available by default in Business Intelligence Development Studio (BIDS). Some features are available in Integration Services by default, so you don't need to set any configurations; other debugging features, such as breakpoints and logging, must be configured and you must interpret the results. You can broadly lump these features into two categories:

- Debugging features that are available by default
- Debugging tools that are required to be configured

The following section covers the features that are available to you by default.

Debugging Features Available by Default

Integration Services provides several features that help you in debugging packages by alerting and providing useful information that you might have overlooked while configuring components in your package. Various windows display variable values, expression results, alerts, and other messages. Following are brief descriptions of some of these windows.

Error List Window

You can launch this window from the View menu in BIDS. It lists errors and warning messages during design time, such as validation errors. The messages are generally descriptive, stating the type of error, source of the error, and description of the error. And if you find it a bit difficult to link the error message to the source task in a complex package, you can double-click the error message to open the editor for that particular task.

Locals Window

This window is quite helpful in debugging by displaying local variables in the current scope. The Locals window is available only when you've suspended execution in your package using breakpoints. While execution is suspended at breakpoints, you can launch this window by choosing Debug | Windows to see the state or current values of local variables at that time.

Watch Window

This window is a great debugging facility that lets you watch for specific variables during package execution. As with the Locals window, you can launch this window by choosing Debug | Windows when your package has suspended execution after encountering a breakpoint. You can directly type in a variable to see its value or set a watch for that variable. You can also assign a new value to the variable manually and inject this value into the package execution to see the results or modify the execution process. You will use this feature to modify the execution of the package in the following exercise.

Output Window

This window is also available from the View menu. It shows results of execution, like the Progress tab described next, and hence includes errors that occur during run time. You will be using this window quite a lot to check status or error messages, more often than the Progress tab, which is a lot more verbose such that it becomes difficult to look for desired messages in it.

Progress Tab

The Designer includes a Progress tab during package execution. It displays lot of progress information about the package and its components. You can find start and stop times, validation errors or warnings, and other execution errors or warnings about the components of the package here. When package execution is finished and you switch back to design mode, this tab changes to the Execution Results tab and shows the messages of the last execution of the package.

Debugging Tools Requiring Configuration

The debugging tools such as breakpoints, logging, precedence constraints, data flow paths, and data viewers must be configured in order to use them.

Breakpoints

As SSIS Designer is based in the Visual Studio environment, it can leverage some of the functionality provided by this programming environment. You can set a breakpoint in your Integration Services package, for example, to help you debug the control flow in your package at run time. Breakpoints suspend the execution of the package and let you review run-time environment conditions, such as variable values and the state of your package at that particular point in execution. You can set breakpoints at the package level, on containers, and on Control Flow tasks.

You can also use breakpoints in your scripts while working with the Script task and break your code at execution time. The VSTA environment is used in Integration Services to develop and debug scripts. This enables you to apply breakpoints to your scripts to pause execution during debugging. You can apply breakpoints in one of the following ways:

- Right-click at the left of the row and choose Breakpoint | Insert Breakpoint option from the context menu.
- Select the line where you want to insert the breakpoint and choose Debug | Toggle Breakpoint, or press the F9 key.

When you set a breakpoint on a line, it is highlighted in red and a red circle appears to the left of the line. The run-time handling of breakpoints in the scripts is the same as handling them while running the package in the Designer. You can step into, step over, or step out of the breakpoint using the provided options from Debug menu. VSTA provides a debugging window that you can use with breakpoints to get information about what's happening with script execution. You can print variable values, execute procedures, or evaluate expressions using this window. This is equivalent to the Locals and Watch windows provided in the SSIS Designer.

Let's work through a Hands-On exercise to see how breakpoints can be set within your packages.

Hands-On: Setting Breakpoints to See Variables Values

You will be setting breakpoints at various levels in an Integration Services package to see how variables change values during execution.

Method

In this exercise, you will be using the Contacting Opportunities with Property Expressions Integration Services project you developed in Chapter 8. This project creates personalized e-mail messages. The package contains a Foreach Loop container that executes seven times, once for each segmented record; reads the October Prospects contact details such as first name, last name, and e-mail address; and passes this information on to the Send Mail task that creates personalized e-mail using this information. The Foreach Loop container passes this information to the Send Mail task using variables. You've seen the package executing but did not see how the variables change the values. In this exercise, you'll be using breakpoints to see exactly how the variables are getting updated.

Exercise (Set Breakpoints)

Here, you will continue working with the Mailing Opportunities.dtsx package and set the breakpoints to see the runtime values.

1. Open the Contacting Opportunities with Property Expressions project in BIDS and then open the Mailing Opportunities.dtsx package in the Designer.
2. Right-click the Iterating October Opportunities Foreach Loop Container and choose Edit Breakpoints from the context menu. The Set Breakpoints dialog box will open.
3. In this dialog box, you can set breakpoints using various break conditions. Take a moment to read through the conditions that are available here. For this exercise, you will be using the Break at the beginning of every iteration of the loop break condition, the last in the list. Select the check box at the left of this break condition. You will see the Hit Count Type column displaying Always for this break condition. Click this and then click the down arrow to see the following four options in the Hit Count Type column:
   - **Always** Execution will always be suspended.
   - **Hit count equals** Specify a value in the Hit Count column when you want to skip some execution cycles of the component—such as when your container iterates over a record set that contains thousands of records and you want to see the variable values after a few thousand iterations. The breakpoint will suspend execution after the number of breakpoint occurrences has reached the value specified in the Hit Count column.
   - **Hit count greater than or equal to** Similar to Hit count equals option, except the execution is suspended when the number of times the breakpoint has occurred is equal to or greater than the value specified in the Hit Count column. The execution will be suspended each time the breakpoint is hit after

the Hit Count value is reached, whereas Hit count equals breaks the execution only once when the Hit Count value is equal to the number of times the breakpoint has occurred.

- **Hit count multiple** You may be interested in seeing variable values at intervals to test that the package is working as expected. You can use this option and specify a value in the Hit Count column to break the execution only when the breakpoint occurrence is a multiple of this value. For example, you may set a value of 100 in the Hit Count column with this option and the execution will be suspended every 100th time the breakpoint occurs.

For this exercise, leave Always selected, and you don't have to specify any value in the Hit Count column (see Figure 15-1). Click OK to close the Set Breakpoints dialog box.

4. You will see a red circle on the Iterating October Opportunities Foreach Loop container, indicating that a breakpoint has been set on this component. Press F5 to execute the package.
5. You will see that the October Opportunities task has been successfully completed and the execution is stopped at Iterating October Opportunities with this container appearing in yellow, and the breakpoint red circle has a yellow arrow inside it. The execution has been suspended at the first hit of the breakpoint. Choose Debug | Windows | Locals to open the Locals window. Expand the

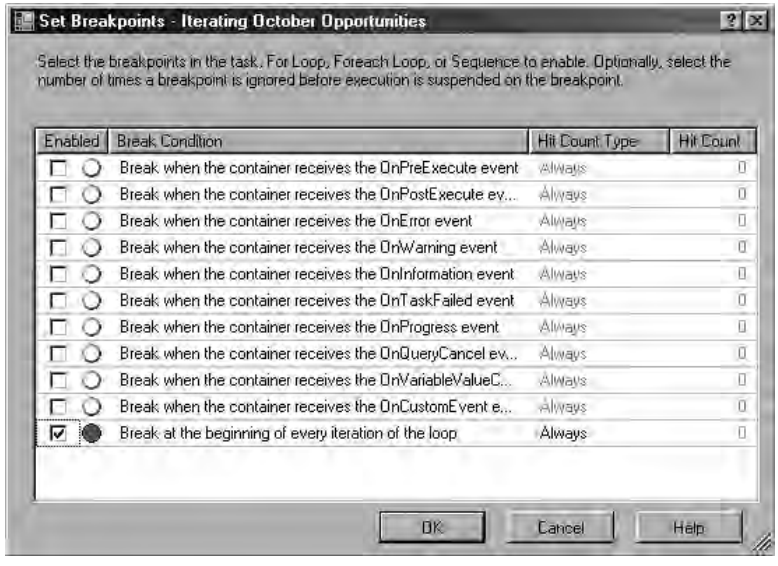


Figure 15-1 *Setting a breakpoint on a Foreach Loop container*

Variables node to see the list of system as well as user variables. Note the User::fname and User::lname variable values.

6. Click the Continue button (green triangle) or press F5 to continue execution. This time you will see that the Mailing Opportunities task has been executed successfully and the package execution is stopped again at the Iterating October Opportunities as it hits the breakpoint a second time. Note the User::fname and User::lname variable values, which have been updated with the values of second row in the data set.
7. Go to the Progress tab in the SSIS Designer, and locate the Task Mailing Opportunities section. Note the two rows showing initialization and completion of the task. Continue the package execution by clicking Continue or pressing F5.
8. The package executes one more iteration of the Iterating October Opportunities Foreach Loop Container and stops as it hits the breakpoint a third time. Note the User::fname and User::lname variable values, which have been updated with the values of the third row in the data set. Go to the Progress tab and check out the Task Mailing Opportunities section. Note that this time it shows ⇒ *Start (2)* at the beginning and ⇐*Stop (2)* at the end of the section, indicating that this task has completed two execution cycles.
9. Go to the Control Flow tab, right-click Iterating October Opportunities, and select Edit Breakpoints from the context menu. For the selected Break Condition, change the Hit Counter Type from Always to Hit Count Equals and specify 2 in the Hit Count column. Click OK to close the dialog box and press F5 to continue the execution of the package.
10. This time you will notice that the Mailing Opportunities task has executed a number of cycles—exactly four—and the execution stops again. You can verify this by going to the Progress tab and checking out the Task Mailing Opportunities section, which shows that the task has been executed six times.
11. Choose Debug | Windows | Watch | Watch 1 to add a watch window. In the Watch 1 window, click in the Name column and type **User::email** and then press the ENTER key. You will see that this variable is added in the Watch 1 window and displays its current value. Expand the User::email variable and type in an alternative e-mail address in the Value field; then press the ENTER key on the keyboard. You will see the value of User::email is changed to the one you've specified. Figure 15-2 shows the various values in the Progress tab, Watch 1 window, and Locals window.
12. Press F5 to continue execution. This time, the package completes execution successfully and stops. Press SHIFT-F5 to switch to design mode. Check your inbox and the alternate e-mail address inbox. You will receive six e-mails in your inbox and one e-mail at your alternative e-mail address.
13. Remove the breakpoint from Iterating October Opportunities, save all the files, and close the project.

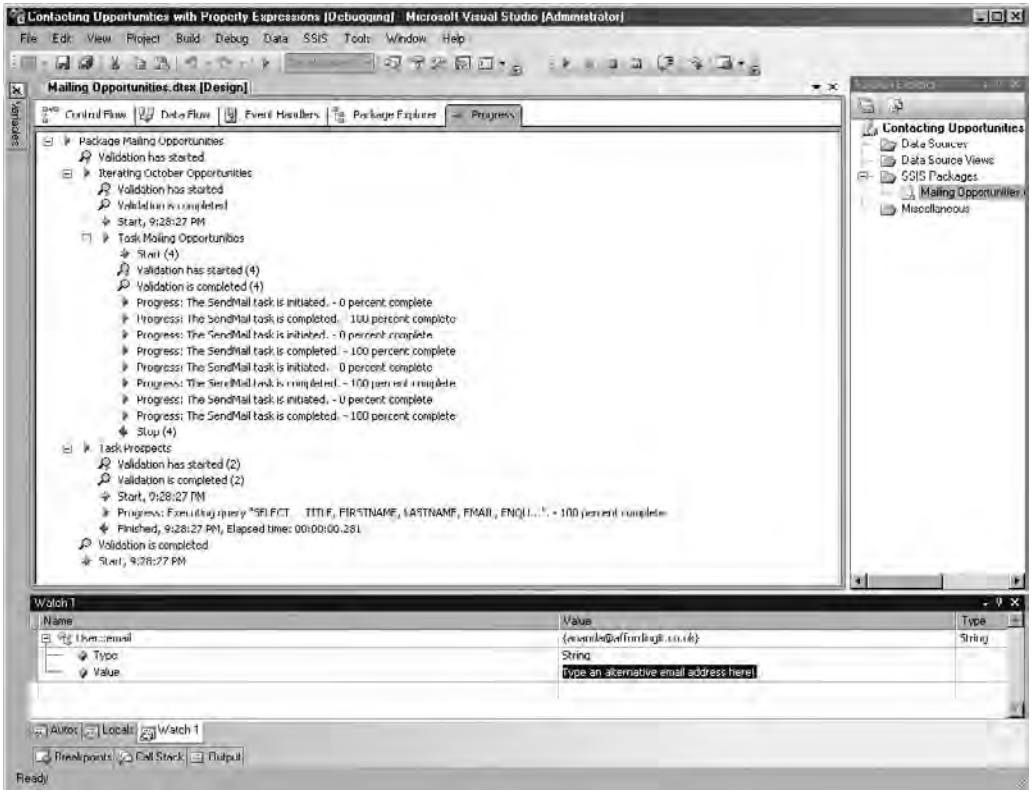


Figure 15-2 *Changing variable value at run time using the Watch 1 window*

Review

In this Hands-On exercise, you've seen how the Progress tab shows the number of executions of a task along with other information. This tab also displays alert, warning, and error messages while the package executes. These messages are still available after the package completes execution when the Progress tab becomes the Execution Results tab. You've also used the Locals window to see the values of variables that get updated with each iteration of the Foreach Loop container. Then you changed the breakpoint setting and jumped four steps at a time. This feature helps a lot while debugging packages with large data sets and you need to check the multiple points of failure. The most interesting thing you've learned is how to change the variable value on the fly and inject that value into the package execution. You can easily simulate various values and see how the package responds to these values using this feature.

Logging Feature

The logging feature of Integration Services can be used to debug task failures and monitor performance issues at run time. A properly configured log schema to log run-time events can be an effective tool in your debugging toolkit. Logging has been discussed in detail in Chapter 8 along with a discussion on various log providers. Refer to that chapter for more details on how to configure logging in your package.

Precedence Constraints and Data Flow Paths

You've used these two connecting elements in package development, so now you understand that precedence constraints are used to connect control flow tasks whereas data flow paths are used to connect data flow components. You can use precedence constraints to control the workflow in a package by configuring the evaluation operation to a constraint or an expression, or a combination of both. Using a constraint as the evaluation operation, you can configure the connected task to run on the basis of success, failure, or completion of the initial task. If you use an expression as an evaluation operation, you can then specify a Boolean expression whose evaluation result of True will let the connected task be executed. You can also choose to use both a constraint and an expression with an AND operator or an OR operator to specify when the next task in the workflow can be executed. You can also set multiple precedence constraints to one task when multiple tasks are connected to this task. You can set it to run when either of the constraint evaluates to True or leave it to default setting of executing the connected task when all the constraints must evaluate to True. By configuring precedence constraints properly, you can control the order of execution of tasks in the workflow to the granular level.

As with precedence constraints, you use data flow paths in the pipeline to connect components. However, data flow paths do not apply any constraints on the data flow as precedence constraints apply to control flow tasks. Several of the data flow components support error outputs that are represented by red-colored data flow paths. You can configure the error outputs of components to handle the rows that do not pass through the main output successfully and can cause truncation of data or failure of the component. These failing rows can be passed on to error outputs to be treated separately from the main pipeline. You can then log these rows to be dealt with later, or you can be more productive by configuring an alternate data flow to fix the errors in the data for such failing rows and put these rows back to the main data flow. This ability to fix errors in the data while processing the bulk of it is extremely powerful and easy to use. So, in the simplistic way, you can use error output to handle errors or the failing rows in the data flow component, and in a slightly modified way, you can use them to deploy alternate pipeline paths to create more productive, more resilient, and more reliable data manipulation packages.

Data Viewers

Data viewers are excellent debugging tools when used in a pipeline—just like oscilloscopes that are used in electric circuits. You see traces of the waves and pulses of current flowing in the circuit using an oscilloscope, whereas you use data viewers to see the data flowing from one component to the other in the pipeline. There is a difference between the two, though. Oscilloscopes do not tend to affect the flow of current, whereas data viewers stop the execution of the data flow engine and require you to click Continue to proceed. Data viewers are attached to the path connecting the two data flow components, and at run time, these data viewers pop open to show you the data in one of four formats: grid, histogram, scatter plot, or chart format.

You were introduced to data viewers in Chapter 9 and used them extensively in Chapter 10. While working with them, you attached the data viewers to the path and saw the data in the grid format popping on the Designer surface. If you run a package that has data viewers attached in the data flow using any method other than inside BIDS, the data viewers do not show up—i.e., data viewers work only when the package is run inside the BIDS environment.

Performance Enhancements

The next step to improve your package development skills is to consider the proper use of resources. Many scenarios occur in development, staging, and production environments that can affect how packages run. For example, your server may be running other processes in parallel that are affected when an Integration Services package starts execution and puts heavy demands on server resources, or your package may use sorts and aggregations that require lots of memory that may not be available to you, or else the tables required by an Integration Services package may be locked by SQL Server to serve queries from other users.

In this section, you will learn skills to design your package and its deployment so that you can manage resources on the server for optimal utilization without affecting other services provided by the server. You will study about optimization techniques to keep your packages running at peak performance levels. As with most database applications, you can enhance the performance of Integration Services by managing memory allocations to various components properly. Integration Services packages can also be configured to work in parallel, as is discussed later in the chapter.

It's All About Memory

One common misconception is to assume that the memory management of Integration Services either is handled by the SQL Server 2005 database engine or can be managed in a similar fashion. However, Integration Services is a totally independent application

that is packaged with SQL Server, but runs exactly the way any other application would run. It has nothing to do with memory management of the SQL Server database engine. It is particularly important for you to understand that Integration Services works like any other application on Windows that is not aware of Advanced Windowing Extensions (AWE) memory. Hence, SSIS can use only virtual address space memory of 2GB or 3GB if the /3GB switch is used on 32-bit systems per process. Here a *process* means a package—that is, if you have spare memory installed on the server that you want to use, you have to distribute your work among multiple packages and run them in parallel to enable these packages to use more memory at run time. This goes back to best practices of package design that advocates modular design for development of more complex requirements. The package modules can be combined using the Execute Package task to form a more complex parent package.

Also, you will need to run child packages out of process from the parent package to enable them to reserve their own memory pool during execution. You can do this by setting the ExecuteOutOfProcess property to True on the Execute Package task. Refer to Chapter 5 for more details on how this property affects running packages. If a package cannot be divided into child packages and requires more memory to run, you need to consider moving to 64-bit systems that can allocate large virtual memory to each process. As SSIS has no AWE memory support, so adding more AWE memory on a 32-bit system is irrelevant when executing a large SSIS package.

64-Bit Is Here

With 64-bit chips becoming more affordable and multicore CPUs readily available, performance issues are beginning to disappear—at least for the time being. Utilizing 64-bit computer systems in the online data processing, analytical systems, and reporting systems makes sense. If you are up against a small time window and your data processing needs are still growing, you need to consider moving to 64-bit technology. The benefits of this environment not only outweigh the cost of moving to 64-bit, but it may actually turn out to be the cheaper option when you're dealing with millions of transactions on several 32-bit systems and need to scale up.

Earlier 64-bit versions in SQL Server 2000 provided limited options, whereas with SQL Server 2005 and later, 64-bit edition provides the same feature set as 32-bit with enhanced performance. All the components of SQL Server 2005 and later versions can be run in native 64-bit mode, thus eliminating the earlier requirement to have a separate 32-bit computer to run tools. In addition, the WOW64 mode feature of Microsoft Windows allows 32-bit applications to run on a 64-bit operating system. This is a cool feature of Microsoft Windows, as it lets third-party software without 64-bit support coexist with SQL Server on the 64-bit server.

While discussing advantages of 64-bit technology, let's quickly go through the following architectural and technical advantages of moving to the 64-bit environment.

Compared to 32-bit systems that are limited to 4GB of address space, 64-bit systems can support up to 1024 gigabytes of both physical and addressable memory. The 32-bit systems must use Address Windowing Extensions (AWE) for accessing memory beyond the 4GB limit, which has its own limitations. The increase in directly addressable memory for 64-bit architecture enables it to perform more complex and resource-intensive queries easily without swapping out to disk. Also, 64-bit processors have larger on-die caches, enabling them to use processor time more efficiently. The transformations that deal with row sets instead of row-by-row operation, such as the Aggregate transformation, and the transformations that cache data to memory for lookup operations, such as the Lookup transformation and the Fuzzy Lookup transformations, are benefited with increased availability of memory.

The improved bus architecture and parallel processing abilities provide almost linear scalability with each additional processor, yielding higher returns per processor when compared to 32-bit systems.

The wider bus architecture of 64-bit processors enables them to move data quicker between the cache and the processor, which results in improved performance.

With more benefits and increased affordability, deployment of 64-bit servers is growing and will eventually replace 32-bit servers. The question you may ask is, what are the optimal criteria to make a switch? As an Integration Services developer, you need to determine when your packages start suffering from performance and start requiring more resources. The following scenarios may help you identifying such situations:

- With the increase in data volumes to be processed, especially where Sort and Aggregate transformations are involved, the pressure on memory also increases, causing large data sets that cannot fit in the memory space to be swapped out to hard disks. Whenever this swapping out of data volumes to hard disks starts occurring, you will see massive performance degradation. You can use Windows performance counters to capture this situation so that when it happens, you can add more memory to the system. However, if you've already run out of full capacity and there is no more room to grow, or if you are experiencing other performance issues with the system, your best bet is to replace it with a newer, faster, and beefier system. Think about 64-bit seriously, analyze cost versus performance issues, and go for it.
- If you are running SSIS packages on a database system that is in use at the time when these packages run, you may encounter performance issues and your packages may take much longer to finish than you would expect. This may be due to data sets being swapped out of memory and also due to processor resource allocations. Such systems will benefit most due to improved parallelization of processes within 64-bit systems.

The next step is to understand the requirements, limitations, and implications associated with use of 64-bit systems. Review the following list before making a final decision:

- Not all utilities are available in 64-bit editions that otherwise are available in 32-bit editions. The only utilities available in 64-bit edition are dtutil.exe, dtexec.exe, and DTSWizard.exe (SQL Server Import and Export Wizard).
- You might have issues connecting to data sources on a 64-bit environment due to lack of providers. For example, when you're populating a database in a 64-bit environment, you must have 64-bit OLE DB providers for all data sources available to you.
- As DTS 2000 components are not available for 64-bit editions of SQL Server 2000, SSIS has no 64-bit design-time or run-time support for DTS packages. Because of this, you also cannot use the Execute DTS 2000 Package task in your packages you intend to run on a 64-bit edition of Integration Services.
- By default, SQL Server 64-bit editions run jobs configured in SQL Server Agent in 64-bit mode. If you want to run an SSIS package in 32-bit mode on a 64-bit edition, you can do so by creating a job with the job step type of operating system and invoking the 32-bit version of dtexec.exe using the command line or a batch file. The SQL Server Agent uses the Registry to identify the correct version of the utility. However, on a 64-bit version of SQL Server Agent if you choose Use 32-Bit Runtime on the Execution Options tab of the New Job Step dialog box, the package will run in 32-bit mode.
- Not all .NET Framework data providers and native OLE DB providers are available in 64-bit editions. You need to check the availability of all the providers you've used in your package before deploying your packages on a 64-bit edition.
- If you need to connect to a 64-bit system from a 32-bit computer where you're building your Integration Services package, you must have a 32-bit provider installed on the local computer along with the 64-bit version, because the 32-bit SSIS Designer displays only 32-bit providers. To use 64-bit provider at run time, you simply make sure that the Run64BitRuntime property of the project is set to True, which is the default.

After you have sorted out the infrastructure requirements of having a 32-bit system or a 64-bit system, your next step toward performance improvement is to design and develop packages that are able to use available resources optimally, without causing issues for other applications running in proximity. In the following sections, you will learn some of the design concepts before learning to monitor performance.

Architecture of the Data Flow

The Data Flow task is a special task in your package design that handles data movement, data transformations, and data loading in the destination store. This is the main component of an Integration Services package that determines how your package is going to perform when deployed on production systems. Typically, your package can have one or more Data Flow tasks, and each Data Flow task can have one or more Data Flow sources to extract data; none, one, or more Data Flow transformations to manipulate data; and one or more Data Flow destinations. All your optimization research rotates around these components, where you will discover the values of properties you need to modify to enhance the performance. To be able to decipher the codes and logs, you need to understand how the data flow engine works and what key terms are used in its design.

When a data flow source extracts the data from the data source, it places that data in chunks in the memory. The memory allocated to these chunks of data is called a *buffer*. A memory buffer is nothing more than an area in memory that holds rows and columns of data. You've used data viewers earlier in various Hands-On exercises. These data viewers show the data stored in the buffer at one time. If data is spread out over more than one buffer, you click Continue on the data viewers to see the data buffer by buffer. In Chapter 9, you saw how data viewers show data in the buffers, and you also explored two other properties on the Data Flow task, which are discussed here as well.

The Data Flow task has a property called DefaultBufferSize whose buffer size is set to 10MB by default (see Figure 15-3). Based on the number of columns—i.e.,

Figure 15-3 *Miscellaneous properties of the Data Flow task*

the row size of pipeline data—and keeping some contingency for performance optimizations (if a column is derived, it could be accommodated in the same buffer), Integration Services calculates the number of rows that can fit in a buffer. However, if your row width is small, that doesn't mean that Integration Services will fit as many rows as can be accommodated in the buffer. Another property of the Data Flow task, DefaultBufferMaxRows, restricts a buffer from including more than a specified number of rows, in case there are too few columns in the data set or the columns are too narrow. This design is meant to maximize the memory utilization but still keep the memory requirements predictable within a package. You can also see another property, EngineThreads, in the same figure that is discussed a bit later in this chapter.

While the data flow is executed, you see the data is extracted by Data Flow sources, is passed on to downstream transformations, and finally lands in the Data Flow destination adapter. By looking at the execution view of a package, you may think that the data buffers move from component to component and data flows from one buffer to another. This is in fact not all correct. Moving data from one buffer to another is quite an expensive process and is avoided by several components of the Integration Services data flow engine. Data Flow transformations are classified into different categories based on their methods of handling and processing data in buffers. Some components actually traverse over the memory buffers and make changes only in the data columns that are required to be changed, while most other data in the memory buffer remains as is, thus saving the costly data movement operation. SSIS has some other components that do require actual movement of data from one buffer to another to perform the required operation. So, it depends on what type of operation is required and what type of transformation is used to achieve the objective. Based on the functions performed by Data Flow transformations, they may use different types of outputs that may or may not be synchronous with the input. This has quite an impact on the requirement of making changes in place and can force data to be moved to new memory buffers. Let's discuss the synchronous and asynchronous nature of Data Flow transformations before discussing their classifications.

Synchronous and Asynchronous Transformations

Data Flow transformations can have either *synchronous* or *asynchronous* outputs. Components that have outputs synchronous to the inputs are called *synchronous components.* These transformations make changes to the incoming rows by adding or modifying columns, but do not add any rows in the data flow—for example, the Derived Column transformation modifies existing rows by adding a new column. If you go to the Input And Output Properties tab in the Advanced Editor for these transformations, you may notice that these transformations do not add all the columns to the Output Columns section, but add only the newly created columns, because all

other columns that are available in the input are, by default, available at the output also. This is because these transformations do not move data from one buffer to another; instead, they make changes to the required columns while keeping data in the same buffer. So the operation of these transformations is quick and they do not place a heavy load on the server. Hence, the transformation with synchronous outputs processes and passes the input rows immediately to the downstream components. One point to note here is that, as these transformations do not move data between buffers, you cannot develop a synchronous transformation for operations that do need to move data between buffers such as sorting and aggregating. Synchronous transformations do not block data buffers, and the outputs of these transformations are readily available to the downstream components. As the data stays in the same buffer within an execution tree (you will learn more about execution trees later in the chapter) where only synchronous transformations are used, any changes you do in the metadata, for instance, addition of a column, get applied at the beginning of the execution tree and not at the time the synchronous component runs in the data flow. For example, if a Derived Column transformation adds a column in the data flow, the column is added at the beginning of the execution tree in which this transformation lies. This helps in calculating the correct memory allocation for the buffer that is created the first time for this execution tree. The column is not visible to you before the Derived Column transformation, but it exists in the buffer.

While the synchronous transformations are fast and lightweight in situations when you are performing derivations and simple calculations on columns, you will need to use transformations that have asynchronous outputs in other situations when you are performing operations on multiple buffers of data such as aggregating or sorting data. The Asynchronous transformations move data from input buffers to new output buffers so that it can perform transformations that are otherwise not possible by processing individual rows. Typically, these transformations add new rows to the data flow—for example, an Aggregate transformation adds new rows and most likely with new metadata containing aggregations of the data columns. The buffers that arrive at the input of an asynchronous transform are not the buffers that leave at the asynchronous output. For example, a Sort transformation collects the rows, decides the order in which to place the rows, and then writes them to new output buffers. These components act as a data source and a data destination at the same time. These transformations provide new rows and columns for the downstream components and hence have to work harder. While developing custom components, you can develop a transform that can have both synchronous and asynchronous outputs, though no such component has been provided in the collection of prebuilt transformations.

If you go to the Input And Output Properties tab in the Advanced Editor for these transformations, you may notice that the transformations define new columns in the Output Columns section. While waiting for the complete data set, these transformations

block the data buffers and slow down the processing. These transformations also put a heavy load or huge memory demands on the server to accommodate the complete data set in the memory. If the server doesn't have enough memory, the data will be cached on to the disk, further degrading the performance of the package. For these reasons, you need to make sure that you use only the minimum required number of asynchronous transformations in your package and that the server has sufficient memory to support the data set that will be fed through the asynchronous transformation. You have used both of these types of transformations that are readily available in the Toolbox in BIDS in Chapter 10 and have also built custom transformations in Chapter 11.

Classifying Data Flow Transformations

Earlier you've seen the transformations grouped together in Chapter 9. Here, you will study the three classifications of Data Flow transformations that are based mainly on performance considerations.

Nonblocking Synchronous Row-Based Transformations

These transformations work on a row-by-row basis and modify the data by changing the data in columns, adding new columns, or removing columns from the data rows, but these components do not add new rows to the data flow. The rows that arrive at the input are those that leave at the output of these transformations. These transformations have synchronous outputs and make data rows available to the downstream components straightaway. In fact, these transformations do not cause data to move from one buffer to another; instead, they traverse over the data buffer and make the changes to the data columns. So these transformations are efficient and lightweight from the process point of view. Their operation is quite visible in BIDS, and you can quickly make out the differences by looking at the execution of the components in run-time mode. However, bear in mind that the run-time display of BIDS is delayed from the actual execution of the package. This delay becomes quite noticeable, especially when you are running a heavy processing package with a large data set and complex transformations. The run-time engine gets busier in actual processing that it prefers than to report back to BIDS, so you can't rely totally on what you see in BIDS. However, for our test cases, you don't need to worry about these delays.

When you execute a package in BIDS, look at the data flow execution of the package. You may see certain components processing together with the annotations for row numbers ticking along, though they may be placed one after another in the pipeline; you may also notice that certain transformations do not pass data to the downstream components for some time. The transformations that readily pass the data to the downstream components are classified as Row transformations. These transformations

do not block data. It's bit difficult to envisage the downstream components working on the data set that is still to be passed by the upstream component. However, in reality, data doesn't flow. Data stays in the same buffer and the transformation works on the relevant column. So as the data is not flowing and is remaining static in the memory buffer; a transformation works on a column while the other transformation performs operations on another column. This is how multiple-row transformations can work so fast (almost simultaneously) on a data buffer.

These transformations fall under one execution tree. The following Row transformations are examples of nonblocking synchronous row-based transformations:

- Audit transformation
- Character Map transformation
- Conditional Split transformation
- Copy Column transformation
- Data Conversion transformation
- Derived Column transformation
- Export Column transformation
- Import Column transformation
- Lookup transformation
- Multicast transformation
- OLE DB Command transformation
- Percentage Sampling transformation
- Row Count transformation
- Row Sampling transformation
- Script Component transformation configured as Nonblocking Synchronous Row–based transformation
- Slowly Changing Dimension transformation

Partially Blocking Asynchronous Row-Set-Based Transformations

Integration Services provides some transformations that essentially add new rows in the data flow and hold on to the data buffers before they can perform the transformation. The nature of such transformations is defined as *partially blocking* because these

transformations hold on to data for a while before they start releasing buffers. As these transformations add new rows in the data flow, they are asynchronous in nature. For example, a Merge transformation combines two sorted data sets that may be the same data, but it essentially adds new rows in the data flow. The time taken by this transformation before starting to release the buffers depends on when these components receive matching data from both inputs.

Following is a list of partially blocking asynchronous row set-based transformations:

- Data Mining Query transformation
- Merge Join transformation
- Merge transformation
- Pivot transformation
- Term Lookup transformation
- UnPivot transformation
- Union All transformation
- Script Component transformation configured as Partially Blocking Asynchronous Row-set-based transformation

Blocking Asynchronous Full Row-Set-Based Transformations

These transformations require all the rows to be assembled before they can perform their operation. These transformations can also change the number of rows in the data flow and have asynchronous outputs. For example, the Aggregate transformation needs to see each and every row to perform aggregations such as summation or finding an average. Similarly, the Sort transformation needs to see all the rows to sort them in proper order. This requirement of collecting all rows before performing a transformation operation puts a heavy load on both processor and memory of the server. You can understand from the nature of the function they perform that these transformations block data until they have seen all the rows and do not pass the data to the downstream component until they have finished their operation.

If you are working with a large data set and have an Aggregate or Sort transformation in your package, you can see the memory usage ramp up consistently in the Task Manager when the transformation starts execution. If the server doesn't have enough free memory to support this, you may also see disk activity and virtual memory being used, which affects performance drastically. When all the rows are loaded in memory, which you can confirm by looking at the row number shown in the Designer and after the processing is done, you may see in the Task Manager that all the built-up memory is released immediately.

The following are blocking asynchronous full row-set-based transformations:

- Aggregate transformation
- Fuzzy Grouping transformation
- Fuzzy Lookup transformation
- Sort transformation
- Term Extraction transformation

Optimization Techniques

The techniques used for optimizing your Integration Services packages are quite simple and similar to any other application built around database operations. Here we will discuss the basic principles that help you optimize packages by applying common-sense techniques. For some of us, it is easiest to spend more money to get the fastest 64-bit machines with multicore CPUs with several gigs of memory and consider optimization done. It does help to throw more hardware and more processing power at performance issues, but if you don't debug the underlying performance bottlenecks, they come back to haunt you a few months after you've upgraded to newer, beefier machines.

From the architecture point of view, you also need to decide whether you'd like to go with Integration Services or any other third-party application, or maybe write the T-SQL scripts directly in SQL Server to perform the work. A bit of analysis needs to be done before you make a decision. The choice to use a particular application may be based on its architectural advantages. For example, using custom-built scripts directly on large, properly indexed databases may prove to be more efficient for batch transformations compared to Integration Services, which is more suitable for complex row-by-row transformations. Most of the transformations of Integration Services are row-based and are quite lightweight and efficient for performing transformations compared to row-set-based transformations. If you need to use lots of row-set-based transformations that move buffers around in the memory, you need to consider whether that can be done using direct SQL. This requires testing and analysis, which you must do to optimize your data transformation operations. On the other hand, if you are working with different data sources such as SQL Server, DB2, Oracle Server, flat files, and probably a piece of XML feed, you may find it very difficult to write a query to join these sources together and then optimize it. In this case, you may find SSIS is simpler and better performing. Monitoring, measuring, testing, and improving are vital to any optimization plan, and you cannot avoid this process with Integration Services, either.

Once you've decided to use Integration Services and are building your package, you can apply the following simple considerations to design your package to perform optimally.

Choose the Right Operation

You need to have a critic's eye when designing an Integration Services package, and whenever you choose a specific operation in your package, keep asking yourself the following:

- Why am I using this operation?
- Can I avoid using this operation?
- Can I share this operation?

It is easy to get carried away when considering what you want to achieve with the various requirements of data flow components. Not all the operations you do in an SSIS package are functional requirements; some of them are forced on to you because of environmental conditions. For example, you have to convert the text strings coming out of a flat file into the proper data types such as integer and datetime. Though you cannot avoid such a conversion, you can still decide where to perform this conversion—i.e., at a source or a destination, or maybe somewhere in between. Second, you need to consider whether you're using any redundant operations in the package that can be safely removed. For example, you may be converting a text column to a datetime column when you extract data from a flat file and later converting the same column into a date type column because of data store requirements. You can avoid one operation in this case by directly converting the column to the data type required by destination store. Similarly, if you are not using all the columns from the flat file, are you still parsing and converting those columns? When it comes to performance troubleshooting, remember *every little bit helps*! Third, consider whether you can distribute processing on multiple machines. Some other specific and thought-provoking components-based examples are discussed in the following paragraphs.

While designing your package, you should always think of better alternatives for performing a unit of work. This calls for testing different components for performance comparisons within Integration Services or using alternative options provided by SQL Server. For example, if you are loading data in to the local SQL Server where your SSIS package is running, it is better to use an SQL Server destination than an OLE DB destination. An SQL Server destination uses memory interfaces to transfer data to SQL Server at a faster rate, but requires that the package be run on the local server. Also, keep in mind that T-SQL can be used to import small sets of data faster than using the Integration Services Bulk Insert task or SQL Server destination because of the extra effort of loading the Integration Services run-time environment.

When you use a Lookup transformation to integrate data from two different sources, think about whether this can be replaced with a Merge Join transformation, which

sometimes could perform better than the Lookup transformation for a large data set. Try using either of them and test the performance difference. Before you go ahead with your test, read on as there are issues with the way you can configure Merge Join transformation and you need to configure your package in a better way to extract the best performance.

From the earlier discussion, you know that a Lookup transformation is a nonblocking synchronous component that wouldn't require data to be moved to new buffers, whereas a Merge Join transformation is a row-set-based partially blocking asynchronous transformation that will move data to new buffers. However, moving data to new buffers may call for a new execution tree, which may also get an additional processing thread and increase the number of CPUs used.

Second, a Merge Join transformation requires data inputs to be sorted in the same order for the join columns. You may think that adding a Sort transformation in the pipeline would be expensive, as it is a blocking asynchronous full row-set-based transformation that needs to cache all the data before sorting it. There is a better alternative to Sort transformation: if your data is coming from an RDBMS where it is indexed on the join column, you can use an ORDER BY clause in T-SQL in the Data Flow Source adapter to sort the data in the RDMS, or if the data is being read from a flat file and the flat file is already sorted in the required order, you can avoid using a Sort transformation by telling the Merge Join transformation that the incoming data is already sorted. The output of data flow source has a property called IsSorted that you can use to specify that the data being output by this component is in sorted order. You need to set this property to True, as shown in Figure 15-4.

After setting the IsSorted property, you still need to specify the columns that have been sorted and the order in which they have been sorted. This works exactly like the ORDER BY clause in which you specify column names and then the order of the sort by using the ASC or DESC keywords. To set this property, expand the Output Columns node, choose the column, and assign a value of *1* to the SortKeyPosition property for specifying that the column is the first column that is sorted in an ascending order (see Figure 15-5), or use a value of *–1* to specify that the first column is sorted in descending order. For specifying second column in the sort order, set the SortKeyPosition value for that column to either *2* or *–2* to indicate ascending or descending order, respectively. By understanding how you can use a Merge Join transformation with already-sorted data, you can determine whether you can extract better performance with a Merge Join transform or by using a Lookup transformation.

The next thing is to keep the data buffers in memory. If you do not have enough memory available to SSIS, partition your data. Smaller batches do not put heavy demands on memory, and if properly configured, values can be fit in the memory available on the server and the process will complete in a much shorter time. I experienced an incident you might find interesting. One of my clients asked me to optimize their DTS package,

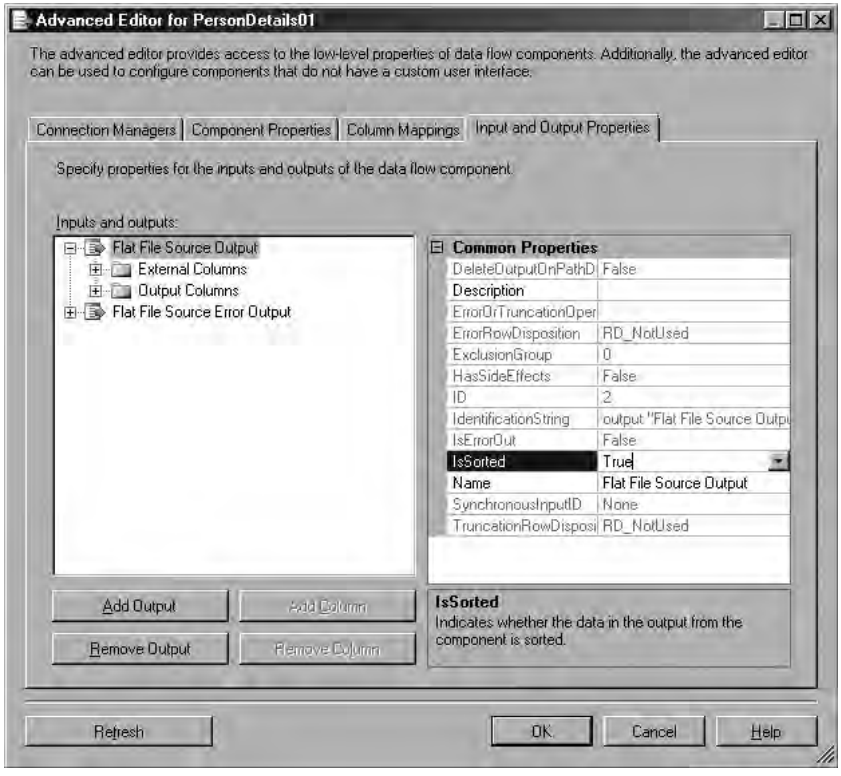


Figure 15-4 *Indicating that the data is sorted using the IsSorted property*

which was loading about 12 million rows into SQL Server. The package was simple but was taking about one and a half days to complete processing of all the files. I started checking the usual things—that files were copied to a local server before the process, that the database had enough free space, and that the disks were configured in RAID 10. The package code was relatively simple. I started believing (against my gut feeling) that this was probably just the time it should take to complete the process. I asked for a baseline. The DBA who implemented the package had done a good job at deployment time by providing the baseline for 100,000 records. The DBA had moved on after implementation of this package, and the data grew massively after that. I quickly did the calculations and realized that something was wrong—the package should finish quicker. Further research uncovered the issue: the commit transaction setting was left blank, which meant that all the rows were being committed together in a single batch. DTS was being required to keep 12 million rows in the memory, so it was no surprise that the server was running out of memory and swapping data out to disks and the processes were taking so much time to complete. I reduced the batch size to 10,000 rows and the

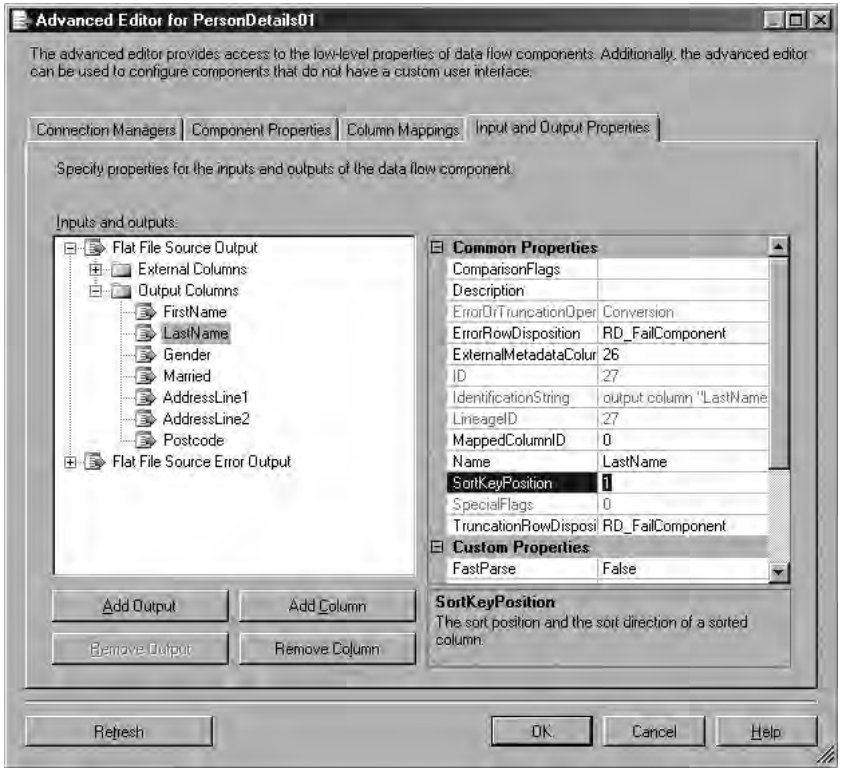


Figure 15-5 *Specifying sort order on the columns*

package completed in less than 4 hours. So, keep the commit size smaller if you need not or cannot use bigger batches. Figure 15-6 shows two properties of the OLE DB destination: Rows Per Batch and Maximum Insert Commit Size, which you can use to specify the number of rows in a batch and the maximum size that can be committed together. However, if you are more concerned about the failures of this task and want to the failing restart package, you need to use this property carefully or consider other options that can help you design your package in such a way. Refer to Chapter 8 to know more about restart options.

Do Only the Work You Need to Do

You should consider this tip seriously, as this is generally an overlooked problem that causes performance issues. Always ask yourself these questions:

- Am I doing too much?
- Can I reduce the amount of work I'm doing?

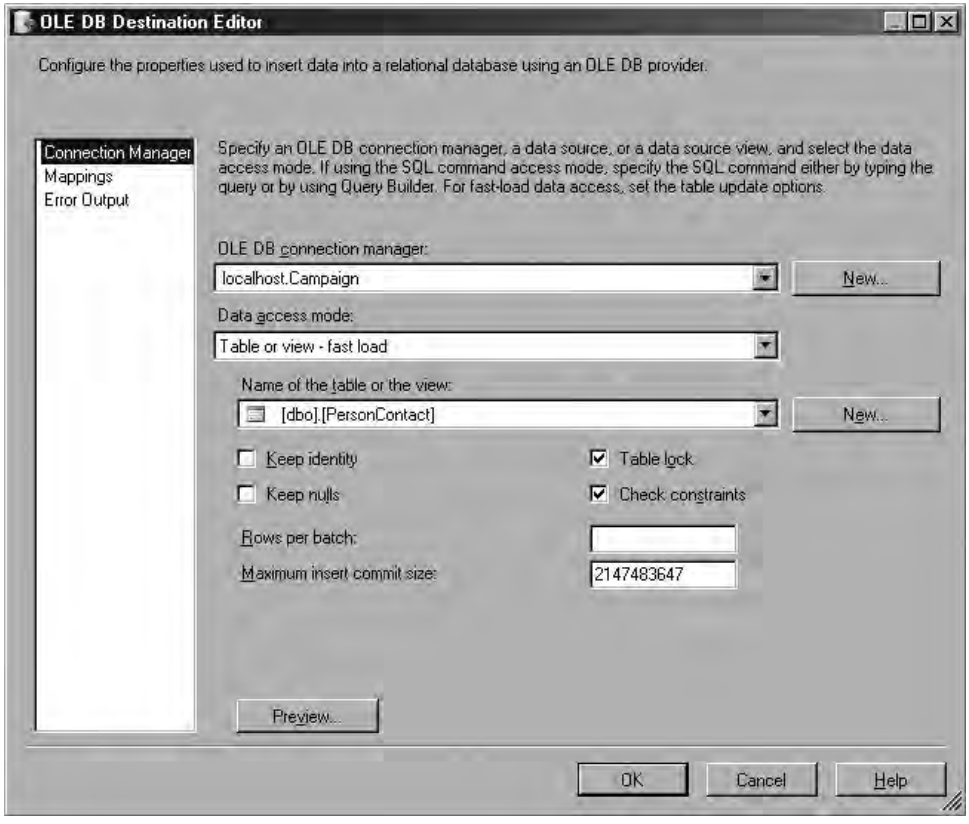


Figure 15-6 *OLE DB destination can control the commit size of a batch.*

One common mistake developers generally make is to start with a larger data set than required in order to build the required functionality quicker, especially when developing a proof of concept (POC). Once the package is created, the data set is reduced by fine-tuning the data flow sources or other components. For example, you may have imported a complete table in the pipeline using a data flow source with a Table or View Data access option, whereas you might have wanted only part of the rows. You can do that while trying to build your package as a proof of concept, but preferably you should use the SQL Command data access mode in the data flow sources to specify the precise number of rows you want to import in the pipeline. Also, remember not to use the `Select * from TableName` query with this option, which is equally bad. You can also reduce the number of data columns from the data set. If you're importing more columns than are actually required and you parse and convert these columns before you decide not to use them in the data flow, this is unnecessary processing that can be avoided by using the SQL command properly to

access the precise data that you want to use in the pipeline. Consider whether you need to use a complete data set in your package or whether a much-reduced data set in rows or columns that consist of only incremental data would be sufficient. You should select only the columns that are required not only in data sources but in the Lookup transformation reference data set as well. Write an SQL select statement with the minimum required data instead of using a table or view to keep the reference data set small that stays in the memory; it will perform better.

Run an SSIS Package at the Most Suitable Location

Choosing a location for running your package is also important when a data source or destination is remote. If you are accessing data over the network or writing to a remote destination, package performance will be affected. Determine whether you can transfer a file to a local server or run a package locally on a machine where the data is stored. Avoid transferring data over a network if possible, as network connectivity issues and traffic conditions may unnecessarily affect performance of your package. Among other factors, the choice of running your package may be guided by whether you are exploding your data using transformations or reducing the data. For example, if you are doing aggregations and storing the results on a remote SQL Server, you can be better off running it on a computer that is on or near to the data source computer. Alternatively, if you are expanding a data set by copying columns or using data conversions, you may be better off getting the raw data from the data source and running the package on the computer on or near to destination computer.

The choice of data type can affect the bytes handled by the package. For example, if you are using an integer data type to specify an integer, it will be represented by a space of 2 bytes in the memory, whereas using a real data type for this number will use 4 bytes. When you are dealing with a large data set that has, say, 10 million rows, you will be adding an extra 20 million bytes to your package that will further degrade the performance when this data is being transferred over the network.

Run an SSIS Package at the Most Suitable Machine

If your SQL Server is being used heavily or scheduled jobs are running on the SQL Server at the same time that you want to run your SSIS package, you can explore the possibility of using a server other than the SQL Server that may have more resources available at that time. This will ease the pressure of a narrow processing window; however, the down side is that you will have to buy an extra license for SQL Server for the server on which you are running Integration Services.

If you are planning to buy a new machine for SQL Server or wherever you will be running Integration Services, consider buying the server with the fastest possible drives, as the disk read and write operations are affected by disk performance. Configure

RAID on your server with a clear understanding of performance options offered by different types. RAID 10 provides the best read and write performance but uses more disk real estate, whereas RAID 5 offers a balance between the disk read speed and the redundancy cost. Use RAID 10 if your budget can afford it.

Avoid Unnecessary Contention for Resources

When deploying your packages, think of the run-time conditions. Ask yourself these questions:

- Will other users be accessing the server at the same time?
- Will other processes or jobs, such as server maintenance plans or backup jobs, be running at that time?

Integration Services can happily coexist with any other application and run packages with minimal requirements for resources. The amount of data to be processed, the transformations, and the ways you've configured your package to use resources all determine the impact on server performance as a whole. If a server is being used heavily, avoid running your packages on the server, as it may not be able to acquire locks quickly enough and will also affect the users connected to other services on the server. Check out the availability of memory on the server at the time when the package will be run. Memory is the most precious resource for any database application, and any contention for memory will negatively affect performance.

Archive Historical Data

You should consider archiving the data that is no longer required for business analysis or reporting, as this type of data is an unnecessary drag on performance. For example, if the business needs only the last three years of data, then the data older than three years will not only take more hard disk space but will slow down all the data access queries, data loading processes, and data manipulation processes. It is worth considering the archival process and the data retention requirements during the design phase of a project so that you can develop your processes accordingly. Keeping your databases lean with only the needed data will avoid unwanted performance hassles.

Discuss and Optimize

As with database development, you discuss the data modeling with developers and other information analysts to agree on properly optimized yet resilient and compliant rules for database design, and adopt best practices for designs and techniques used within the development team. You will find this is not only encouraging for the team,

but it will provide uniform techniques that can be used throughout the enterprise—and that goes a long way toward developing a culture of adopting best practices.

As mentioned in various Hands-On exercises, make your packages self-explanatory by adding proper descriptions and comments in tasks and annotations. You can annotate your package on the Control Flow surface to explain how the package works, and this helps other developers quickly understand the functionality and will help avoid accidental changes. Document and distribute the adopted naming conventions, auditing, and logging for SSIS packages.

Test, Measure, and Record

Performance tuning is a strenuous process. You must clearly define performance requirements and try to keep your packages performing within that matrix. The packages change execution behavior over time as the data to process grows. When you develop an SSIS package, you should first test and document the performance of the package to develop a baseline to compare with future test results. Having a baseline can help you quantify the performance tuning you need to do to optimize the package.

If at some stage you want to break open the pipe and measure the data pressure, as most plumbers do to clear blocked pipes, you can use a trick explained in the following few lines to get a view of how much performance can be achieved with your pipeline. You can replace the downstream components at any stage in your pipeline with a Row Count transformation that is quick to consume the rows coming to it. You can determine maximum speed at any stage of your package and compare this value with the real-time value—i.e., with the real components in place. This is handy for finding out which component is degrading the performance of your package. It is worth recording the values monitored with this technique for future references as well. Various tools and utilities can be used to measure the baseline parameters, and will study these in the following section.

Performance Monitoring Tools

Integration Services provides a number of performance counters that can help you monitor the run-time workings of a package. You can also use tools such as SQL Server Profiler provided with SQL Server 2008 and Windows Performance counters to get a complete picture of run-time activities. These tools can be useful in understanding the internal workings and identifying which components are acting as bottlenecks in the performance of your package. In addition, you can use the Logging tool provided by Integration Services to develop a performance baseline for your package.

Performance Counters

You can use a set of performance counters provided by Integration Services to track pipeline performance. You can create a log that captures performance counters that are available in the SQLServer:SSISPipeline object. You can access these counters in the Windows Perfmon tool also called Performance Monitor.

These counters provide information about three main types of objects: BLOB data, memory buffers, and the number of rows. Knowing about memory usage is more important, so more counters are provided to track this. The SSIS pipeline uses memory buffers to keep the data and to allocate memory to individual components to meet their processing requirements. The buffers used to hold data are called *flat buffers,* and the buffers allocated to components such as Sort, Aggregate, or Lookup transformations for their internal hashing and calculation purposes are called *private buffers.* Large binary objects can require lot of the memory buffers, so use BLOB counters to check out these values if your data carries BLOB objects. These performance counters are described here:

- **BLOB Bytes Read** Displays total number of BLOB bytes read from all the data sources, including the Import Column transformation.
- **BLOB Bytes Written** Displays the total number of BLOB bytes written to all data destinations, including the Export Column transformation.
- **BLOB Files In Use** Displays the number of BLOB spooling files in use throughout the pipeline.
- **Buffer Memory** Displays the amount of memory buffers allocated to the pipeline at different times during the package execution. Compare this value with the memory available (which you can capture using memory object counters) on the computer to track whether the available memory falls short during any time of the package processing. The Buffer Memory counter value includes both physical and virtual memory used, so if this value is close to physical memory on the computer, you can expect the swapping of memory to disk. This is also indicated by Buffers Spooled counter, as its value starts increasing to indicate a shortage of physical memory. These are important counters to observe to identify slow performance due to memory swapping to disk.
- **Buffers In Use** Displays the number of buffers used from the allocated buffers for the pipeline.
- **Buffers Spooled** This is the most important counter to observe if your package is taking an exceptionally long time to execute. It will help you determine whether at any time during the package execution, Integration Services starts swapping out

buffers to disk. Whenever memory requirements outpace the physical memory available on the computer, you will see that the buffers not currently in use are swapped out to disk for later recovery when needed. This counter tells you the number of buffers being swapped out to disk. This is an important event to watch.

- **Flat Buffer Memory** Flat buffers are used to store data when a package runs. This counter displays the total amount of memory allocated to all the flat buffers. If your package has multiple Data Flow tasks, this counter shows consolidated value used by all the Data Flow tasks.
- **Flat Buffers In Use** Displays the number of flat memory buffers used by data flow engine.
- **Private Buffer Memory** Some transformations such as the Sort transformation and the Aggregate transformation need extra memory buffers to perform the operations on the data in flat buffers. These extra memory buffers are locally allocated to the transformation and are called private buffers. This counter shows the total number of buffers allocated as private buffers in the pipeline.
- **Private Buffers In Use** Displays the number of buffers in use throughout the pipeline.
- **Rows Read** Displays the total number of rows read from all data sources. The rows read by the Lookup transformation for lookup operations are not included in the total.
- **Rows Written** Displays the total number of rows that are written to all the Data Flow destinations.

In addition to these performance counters, SQL Server 2008 provides another counter to monitor the number of package instances currently running. The SSIS Package Instances counter is available under SQL Server:SSIS Service 10.0 Performance object.

SQL Server Profiler

You can use the SQL Server Profiler whenever you're transferring data with SQL Server to determine what's happening inside SQL Server that may be negatively affecting the running of your package. If your package is simple and a light load, you expect it to be running at top speed, but if SQL Server is also running other processes during that time, your package may find it difficult to transfer data. With SQL Server Profiler, you can monitor the SQL Server not only for data access but also for the performance of the query you may be using in a data source to access the data.

Logging

You've already read about and used logging in Integration Services, so it is worth knowing that you can use logging to create a baseline for your package execution as well. This baseline should be revised from time to time as the data grows or whenever the processing design of the package is changed. It is particularly helpful to watch the time taken by different tasks or components to complete, as you can focus on improving this. For example, if a data source takes most of the processing time to extract data from a source, it is not going to benefit much if you're putting efforts into improving transformations.

The Data Flow task also provides some interesting custom log events that are helpful in debugging issues that affect performance of the pipeline. You can view these events in the Log Events window when the package is being executed by selecting the Log Events command from the SSIS menu or by right-clicking the Control Flow surface and choosing Log Events from the context menu. Alternatively, you can log these events by configuring logging for the Data Flow task. Also, other than the following defined logging events, it tells you about the pushback in the engine to save memory.

Following are descriptions of some of the log events available for the Data Flow task. These can be helpful in monitoring performance-related activities:

- **BufferSizeTuning** This event happens whenever the Integration Services pipeline changes the size of a buffer from the default size. This log entry also specifies the reason for changing the buffer size, which is generally about either too many rows to fit in the default buffer size or too few for the given buffer size. It indicates the number of rows that can fit in the new buffer. Refer to the earlier discussion on DefaultBufferSize and DefaultBufferMaxRows for more details on buffer size and rows that can fit in a buffer.
- **PipelineBufferLeak** When the pipeline execution stops, some of the components may hold on to the buffers they used even after the buffer manager has stopped. Thus the memory buffers that are not freed will cause a memory leak and will put extra pressure on memory requirements. You can discover such components using this event log, as it will log the name of the component and ID of the buffer.
- **PipelineComponentTime** Each component in a pipeline undergoes the five major processing steps of Validate, PreExecute, PostExecute, ProcessInput, and PrimeOutput, and this event log reports the number of milliseconds spent by the component in each of these phases. Monitoring this event log helps you understand where the component spent most of the time taken.
- **PipelineExecutionPlan** An SSIS pipeline has an execution plan just as the stored procedures have. This event provides information about how memory buffers are created and allocated to different components. By logging this event

and the PipelineExecutionTrees event, you can track what is happening within the Data Flow task.

- **PipelineExecutionTrees** The pipeline is divided into separate execution trees based on the synchronous relationship among various components of the Data Flow task. When Integration Services starts building an execution plan for the package, it requires information about execution trees, and this information can be logged using this event log.
- **PipelineInitialization** This log event provides in one or more entries the information about directories to use for temporary storage of BLOB data, the default buffer size, and the number of rows in a buffer at the initialization of the Data Flow task.

You will log these events later in a Hands-On exercise to understand them better.

Execution Trees

At run time, the pipeline engine divides the execution of pipeline into discrete paths just like an execution plan for a stored procedure. These discrete paths, called *execution trees* (also called *execution paths* in Integration Services 2008), are allocated their own resources to run the package at optimal levels. The number of execution paths in a pipeline depends on the synchronous relationship among the components and their layout in the package. In simplistic terms, if a package consists of only synchronous row-based components, it will have only one execution path. However, if you introduce a component with asynchronous outputs in the pipeline, it will be executed in two discrete parts and will have two execution paths. The asynchronous output of the component starts a new execution path, whereas its input is included in the upstream execution path. So, from this, you can make out that an execution tree starts at a data flow source or a component with asynchronous outputs and ends at a data flow destination or at an input of the component with asynchronous outputs.

Let's review what happens within an execution tree. From earlier discussions, you already know that the components with synchronous outputs—i.e., row-based components—work on the same data buffers and do not require that data be moved to new buffers. This set of buffers constitutes an execution path. All the components within an execution path operate on the same set of buffers. As the data is not moved, it allows transformations to perform operations at the maximum attainable speed on the data. Addition of an asynchronous component in the pipeline requires data to be moved to new set of buffers, hence a new execution path; however, this also means that the new execution path might get its own worker thread, thus increasing CPU utilization. So, some developers used this trick in earlier versions of Integration Services to break the single thread execution by introducing an asynchronous transformation in

the data flow to use more processors and hence increase performance. However, this trick also has a performance overhead involved in moving data to new buffers. This is no longer required in Integration Services 2008.

Integration Services 2005 had a limitation of assigning generally one worker thread per execution tree. This happened because the thread scheduling was done during the pre-execution phase when the relative amount of work for each execution tree was still not known; this design resulted in poor performance in some cases, especially when using multicast or lookup transformations. Users have experienced that the SSIS package uses relatively few CPUs even though several processors are free on a multiprocessor machine. The pipeline architecture in Integration Services 2008 has been enhanced with improved parallelism and can now allocate multiple worker threads. The worker threads are assigned dynamically at run time to individual components from a common thread pool that results in utilization of more CPUs on a multicore computer. The packages that have high degree of parallelism will benefit most, especially if they contain transformations such as lookup and multicast. The pipeline engine can create subpaths for these transformations and allocate them their own worker threads, thus increasing parallelism. For example, for a multicast transformation all the outputs will now each get separate subpaths and hence their own worker threads, compared with only one execution tree and only one worker thread in the case of SSIS 2005. The ability to allocate multiple processes and create subpaths even in the scope of a set of synchronous transformations enables SSIS 2008 to achieve high performance. This happens automatically in the pipeline engine, requiring little configuration from developers, thus making SSIS 2008 more productive.

Hands-On: Monitoring Log Events in a Pipeline

In this exercise, you will discover the execution trees in the data flow of your package.

Method

You will enable logging in the package and add custom log events on the Data Flow task to log what's happening in the package at run time.

Exercise (Enable Logging on the Data Flow Task)

Here, you will be using the Updating PersonContact package of the Data Flow transformations project you built in Chapter 10.

1. Open the Data Flow transformations project using BIDS and then load the Updating PersonContact.dtsx package on the Designer.
2. Right-click the blank surface of the Control Flow and choose Logging from the context menu.

3. Click the check box to enable logging for Updating PersonContact in the Containers pane.
4. On the right side, in the Providers And Logs tab, select the SSIS log provider for Text files selected in the Provider Type field and click Add to add this provider type. When this provider type has been added, click in the Configuration column, then click the down arrow and select <New Connection…> to add the File Connection Manager.
5. In the File Connection Manager Editor, select Create File in the Usage Type field. Type **C:\SSIS\RawFiles\ExecutionLog.txt** in the File field and click OK.
6. On the left side, click the Data Flow task and then click twice in the check box provided next to it to enable logging for this task. The right pane becomes available. Click to select the SSIS log provider for Text files log.
7. Go to the Details tab, scroll down, and select the custom events BufferSizeTuning, PipelineBufferLeak, PipelineComponentTime, PipelineExecutionPlan, PipelineExecutionTrees, and PipelineInitialization, as shown in Figure 15-7. Click OK to close this dialog box.
8. Go to the Data Flow tab and delete the data viewers attached to all data flow paths, if any.

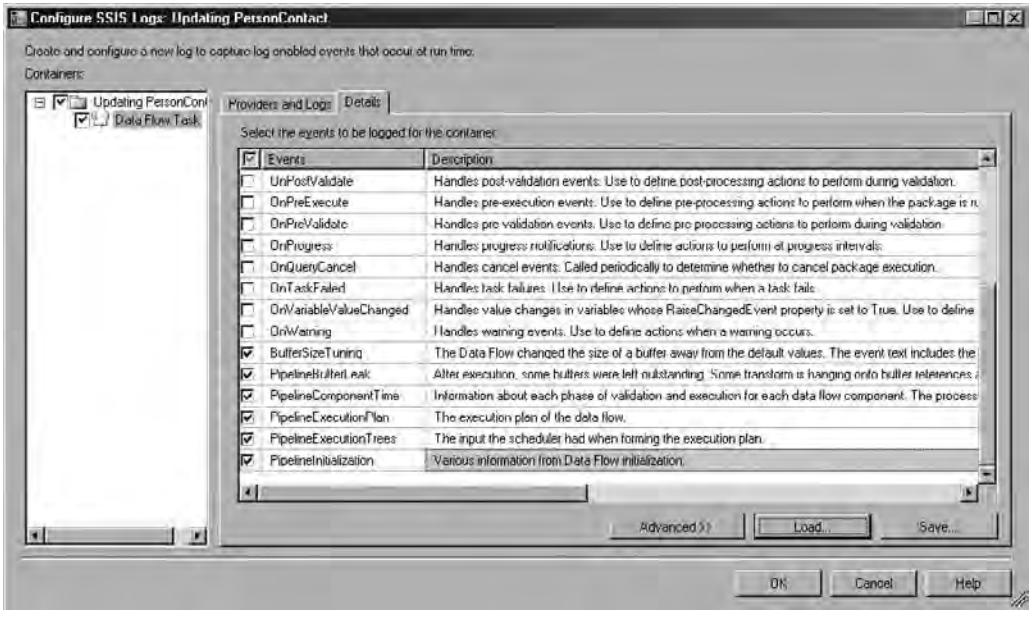


Figure 15-7 *Custom log events provided by the Data Flow task*

9. Right-click the Updating PersonContact.dtsx package in the Solution Explorer window and choose Execute Package from the context menu.
10. When the package has been executed, press SHIFT-F5 to switch back to designer mode.

Exercise (Review the ExecutionLog File)

In this part, you will review the execution log file using Notepad.

11. Explore to the C:\SSIS\RawFiles folder and open the ExecutionLog.txt file using Notepad.
12. Look through the log file for the PipelineComponentTime entries for different components. You will notice that in the beginning of the file (and hence the processing) you have entries for validate events and later, almost at the end, there will be entries for other phases such as the PreExecute, PostExecute, ProcessInput, and PrimeOutput events.
13. After the validation phase, you will see the list of execution trees under the PipelineExecutionTrees log entry. The log is listed here in case you haven't managed to run the package until now:

```
Begin Path 0
   output "Flat File Source Output" (2); component "PersonDetails01" (1)
   input "Union All Input 1" (308); component "Merging PersonDetails01 and
PersonDetails02" (307)
End Path 0

Begin Path 1
   output "Excel Source Output" (17); component "PersonDetails02" (9)
   input "Data Conversion Input" (73); component "Converting PersonDetails02"
(72)
   output "Data Conversion Output" (74); component "Converting
PersonDetails02" (72)
   input "Union All Input 2" (332); component "Merging PersonDetails01 and
PersonDetails02" (307)
End Path 1
Begin Path 2
   output "Union All Output 1" (309); component "Merging PersonDetails01 and
PersonDetails02" (307)
   input "Derived Column Input" (177); component "Deriving Salutation" (176)
   output "Derived Column Output" (178); component "Deriving Salutation"
(176)
   input "Character Map Input" (194); component "Uppercasing Postcode" (193)
   output "Character Map Output" (195); component "Uppercasing Postcode"
(193)
   input "Lookup Input" (203); component "Adding City Column" (202)
   Begin Subpath 0
      output "Lookup Match Output" (204); component "Adding City Column"
(202)
      input "OLE DB Command Input" (254); component "Deleting Duplicates"
(249)
      output "OLE DB Command Output" (255); component "Deleting Duplicates"
```

```
(249)
      input "OLE DB Destination Input" (279); component "PersonContact" (266)
   End Subpath 0
   Begin Subpath 1
      output "Lookup No Match Output" (217); component "Adding City Column"
(202)
      input "Flat File Destination Input" (228); component "No Match Lookups
File" (227)
   End Subpath 1
End Path 2
```

Let's now see how the pipeline engine has created execution paths. The execution paths are numbered beginning with 0, so you have three main execution paths in total. Based on the preceding log events, the execution paths have been marked in the Figure 15-8.

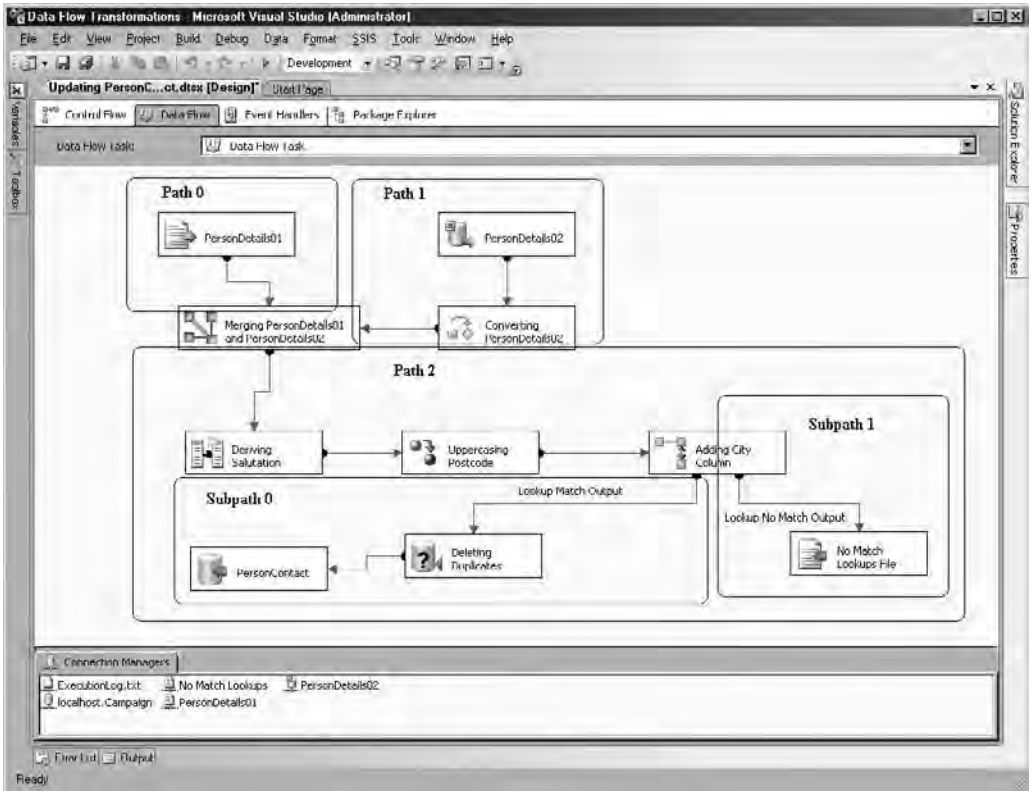


Figure 15-8 *Execution paths in the Updating PersonContact package*

14. The next section of the log shows PipelineExecutionPlan, which is listed here:

```
Begin output plan
   Begin transform plan
     Call PrimeOutput on component "Merging PersonDetails01 and
PersonDetails02" (307)
          for output "Union All Output 1" (309)
   End transform plan
   Begin source plan
     Call PrimeOutput on component "PersonDetails01" (1)
          for output "Flat File Source Output" (2)
     Call PrimeOutput on component "PersonDetails02" (9)
          for output "Excel Source Output" (17)
   End source plan
End output plan
Begin path plan
   Begin Path Plan 0
      Call ProcessInput on component "Merging PersonDetails01 and
PersonDetails02" (307) for input "Union All Input 1" (308)
   End Path Plan 0
   Begin Path Plan 1
      Call ProcessInput on component "Converting PersonDetails02" (72) for
input "Data Conversion Input" (73)
      Create new row view for output "Data Conversion Output" (74)
      Call ProcessInput on component "Merging PersonDetails01 and
PersonDetails02" (307) for input "Union All Input 2" (332)
   End Path Plan 1
   Begin Path Plan 2
      Call ProcessInput on component "Deriving Salutation" (176) for input
"Derived Column Input" (177)
      Create new row view for output "Derived Column Output" (178)
      Call ProcessInput on component "Uppercasing Postcode" (193) for input
"Character Map Input" (194)
      Create new row view for output "Character Map Output" (195)
      Call ProcessInput on component "Adding City Column" (202) for input
"Lookup Input" (203)
      Create new execution item for subpath 0
      Create new execution item for subpath 1
      Begin Subpath Plan 0
         Create new row view for output "Lookup Match Output" (204)
         Call ProcessInput on component "Deleting Duplicates" (249) for input
"OLE DB Command Input" (254)
         Create new row view for output "OLE DB Command Output" (255)
         Call ProcessInput on component "PersonContact" (266) for input "OLE
DB Destination Input" (279)
      End Subpath Plan 0
      Begin Subpath Plan 1
         Create new row view for output "Lookup No Match Output" (217)
         Call ProcessInput on component "No Match Lookups File" (227) for
input "Flat File Destination Input" (228)
      End Subpath Plan 1
   End Path Plan 2
End path plan
```

The PipelineExecutionPlan creates two different plans: the output plan and the path plan. The output plan consists of the source plan and the transform plan. The source plan represents the outputs of data flow sources, while the transform

plan represents the outputs of asynchronous transformations. The path plan creates a plan for each execution plan or subplan. Actually, the pipeline uses two different types of threads, SourceThreads and WorkThreads. The SourceThreads are directly related to the output plans that represent the data flow sources and the asynchronous transformations outputs. On the other hand, WorkThreads are related to the path plans that represent the execution paths and subpaths. Each source thread has to create a buffer and call the PrimeOutput method on the source component, or the component with asynchronous output, so that it can start filling up the output buffer with the data rows. This is where the execution starts and the rest of the work is distributed among WorkThreads. The WorkThreads are also linked to the data flow task property called EngineThreads. Refer to earlier Figure 15-3, which shows the EngineThreads property set to the default value of 10. This EngineThreads property value suggests to the data flow engine the number of threads to use during execution. You will study more about the EngineThreads property later in this chapter.

15. Next, search for the string BufferSizeTuning and read through the messages related to the BufferSizeTuning event. Here is one example:

```
"Rows in buffer type 2 would cause a buffer size greater than the configured
maximum. There will be only 2449 rows in buffers of this type."
```

    This event indicates that the buffer size created will be larger than the default size configured; it also tells you the number of rows that this buffer will be able to hold.

16. Finally, look through the PipelineInitialization log entries, which tell you about temporary BLOB and buffer storage locations for the pipeline execution environment.

Review

This exercise showed you that the pipeline engine divides the pipeline work in discrete execution paths on the basis of synchronicity in the package. Whenever it comes across a component with asynchronous outputs, the engine creates a new execution path by moving the data to the new buffer set. You've also seen that each execution path gets a WorkThread subject to the EngineThreads setting on the Data Flow task. Finally, the SourceThreads are directly linked to the data flow sources and asynchronous transformations present in the pipeline. The allocation of threads on the basis of the EngineThread property suggestion and the execution plans is done automatically by the pipeline engine; however, understanding the way these work will help you while designing or debugging your package for performance reasons.

Using Parallel Processing

You must have understood by now that Integration Services packages have increased parallelism compared to previous releases of this product. You've learned optimization techniques earlier in this chapter that talked about managing buffers so that you could

avoid swapping out buffers to disk and fit maximum rows in a buffer. Now you will learn how to make use of more processors available on your computer so that the packages run faster. Using both techniques—managing buffer utilization and using maximum CPUs on a multicore computer—you can develop highly efficient packages that work with peak performance.

Running Parallel Tasks in the Control Flow

Recall that the tasks in the control flow are connected by precedence constraints and a task in the control flow runs only when the previous task has finished based on the precedence condition specified. However, you can create unrelated parallel workflows—i.e., flows not connected to each other—in the control flow and run them simultaneously. The control flow of a package has a property called MaxConcurrentExecutables, shown in Figure 15-9, that specifies the number of executables—i.e., tasks that can run in parallel.

The value of *–1* shown in the figure is the default value of this property, indicating that the maximum executables that this package can run will be the number of processors on the computer plus 2. It counts the logical processors. On a basic four-CPU server, this value will allow maximum of six executables to run in parallel; however, if you have a server that has four dual-core CPUs with hyperthreading enabled, the default value will be interpreted as 18. Based on the number of CPUs on the server and whether other applications are also running on the server, you can configure this setting. If you configure this value too low for a package that can run more tasks in parallel than this setting indicates, some of the tasks will have to wait. However, if you configure this value too high and the server also hosts other applications, the performance and usability of those applications will suffer. Before deciding the number of CPUs for the

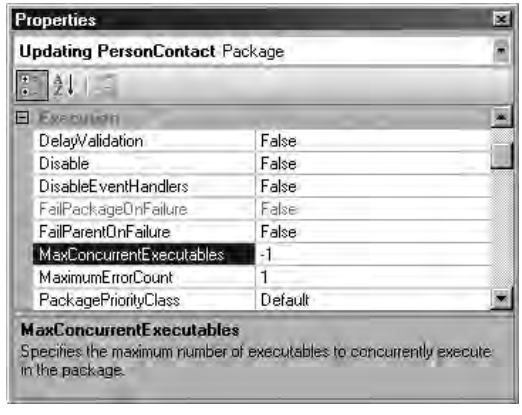


Figure 15-9 *Control flow properties of a package*

given tasks, consider the types of tasks that are running in parallel. Not all tasks require the same processing power—for example, an FTP task requires much less processing power than a Data Flow task.

Creating Multiple Data Flows

If you must deal with massive amounts of data or you've multiple data sources that require independent data conversions and transformations, you can create multiple Data Flow tasks in your package. As mentioned, these tasks can be executed in parallel, so if you've added more than one Data Flow task in your package, all will be executed in parallel and hence perform the work faster, depending upon the MaxConcurrentExecutables setting and availability of resources on the server. Remember that the execution paths are created for each Data Flow task, which means multiple Data Flow tasks will create more execution paths and will help the pipeline to execute faster, though they are subject to availability of sufficient number of CPUs on the computer to support multiple Data Flow tasks.

Enhancing EngineThreads

As discussed earlier and shown in Figure 15-3, the EngineThreads property specifies the maximum number of threads allocated to a data flow. The EngineThreads property hints to the pipeline scheduler about the maximum number of threads that can be allocated to this data flow. The SourceThreads and the WorkThreads are related to the EngineThreads property, though they are not hard-bound with it. The EngineThreads value applies to both source threads and the work threads equally. This means that the default setting of 10 can create up to ten source threads and up to ten worker threads. Bear in mind that this value is just a suggestion and not a hard-forced value on the data flow engine. For example, if the data flow engine needs only five threads and the default value is set to 10, the data flow engine will not use more than five threads. Also, if necessary, the data flow engine will use more threads to avoid unwanted conflicts if the value specified is lower than the requirement of threads. If you don't have enough execution paths on your package, increasing the EngineThreads value will not help much.

Summary

You've learned lots of techniques for debugging and optimizing your packages in this chapter. While learning these techniques, you learned the internal working of Integration Services and saw how it allocates resources at run time to different components on the basis of execution paths. The chapter discussed troubleshooting your packages using various features such as breakpoints, precedence constraints, and

data viewers. You also worked with breakpoints in a Hands-On exercise to see how variable values are changed during execution of a package.

In the performance section, you learned how to manage memory allocation to data buffers and the types of components that work on the data in the same set of buffers while others force the data to move to the new buffers. These components also exhibit blocking, nonblocking, and partially blocking nature for data. You worked through another Hands-On exercise to learn about execution paths created in a package.

Last, you learned that while SSIS is enhanced for parallelism, yet there are some configurations that you can use to put all the CPUs in your computer to work. Microsoft has published ETL Performance Guidelines documents that could be very helpful for optimizing your packages. Not only in this chapter, but throughout this book, you learned to use various components by working through Hands-On exercises. I hope this book becomes an oft-consulted source for your use of Integration Services, and I hope it sets you free to begin your exciting journey with Integration Services. Best of luck.

This page intentionally left blank

Appendix

How to Use the Provided Software

In This Chapter

- Downloaded Software
- Attaching Campaign Database

All the exercises you do in this book have been created and provided for your quick reference along with all the input files. The following steps explain how you can get and use the software provided for this book.

Downloaded Software

1. Go to www.mhprofessional.com/computingdownload. Under "Free Downloads," locate this book's title—*Hands-On Microsoft SQL Server 2008 Integration Services, Second Edition.* Click the link to download a Zip file.
2. Expand the Zip file to get all the files you need to practice, including input files, developed SSIS packages, and SQL Server database files. The Zip file contains the folder structure necessary for performing the exercises. All you need to do is copy the root folder, SSIS, to your C:\ folder. The following table explains the folder structure and their usage.

Folder	Description
SSIS	This is the root folder. Copy SSIS folder to C:\.
Database	This folder contains the SQL Server 2008 database files. The following section explains how you can attach these database files to your SQL Server 2008 server.
Deployment	Blank folder for exercises in this book.
Downloads	Blank folder for exercises in this book. Contains a subfolder, Archive, that is also blank.
Freeware	Holds the Freezip software used in Chapter 5.
Packages	This folder has a backup subfolder that contains the SSIS packages, which you develop in Chapter 2.
Projects	All the SSIS projects developed for this book are contained within a backup subfolder. You will be developing your own projects in this folder.
Rawfiles	Holds all the input files required for various exercises.
Code for individual chapters	This folder contains a folder structure similar to that contained in SSIS main folder. The code and the input files are relevant for the individual chapters and are provided for your review.

Attaching Campaign Database

Once you've copied SSIS folder from the Zip file to your C:\ folder, you are ready to attach the Campaign database to your copy of SQL Server 2008. As mentioned, the C:\SSIS\database folder contains the database files. To attach the Campaign database, follow these steps:

1. Run SQL Server Management Studio and connect the Database Engine to the localhost.
2. In the Object Explorer, right-click the Databases node and select Attach.
3. In the Attach Databases dialog box, click Add. Choose the Campaign_Data.mdf file from the C:\SSIS\Database folder and click OK.
4. Click OK in the Attach Databases dialog box to attach the selected file as the Campaign database.

This page intentionally left blank

Index

Numbers

32-bit vs. 64-bit systems
- compared with 64-bit, 17
- enhancing performance with, 641–643

A

access control
- exercise using roles for, 287
- fixed database-level roles and permissions, 282–283
- package security features, 7–8
- server storage and, 50
- user-defined roles for, 283

Access (Microsoft), as data source, 43
Active Directory, 172
ActiveX Script task
- backward compatibility and, 141
- extending Dts and, 484
- legacy scripting, 486–487
- mapping Dts services to SSIS, 615

administration
- adding root-level folder to stored packages, 232–233
- automating administrative tasks, 10
- connecting to SSIS service, 226
- connecting to SSIS service on remote servers, 228–229
- dtutil utility and. *See* dtutil utility
- importing and exporting packages, 233–235
- managing packages on instances of SQL Server, 228
- managing packages on remote servers, 227–228
- managing packages using default settings, 227
- overview of, 226
- running packages programmatically, 266–267
- running packages with BIDS, 244–245
- running packages with DTExec utility, 250–256
- running packages with DTExecUI, 245–250
- running packages with SQL Server Agent, 256–266
- running packages with SQL Server Import and Export Wizard, 244
- saving packages to SSIS package store, 230–232
- understanding storage areas, 230
- utilities for running packages, 244

ADO Connection Manager, 75
ADO.NET
- data sources, 343–345
- destination in data flow, 357
- improvements to ADO.NET components, 12

ADO.NET Connection Manager
- ADO.NET source and, 343
- checking database integrity and, 213–214
- overview of, 75–76

Advanced Editor, SQL Server Import and Export Wizard, 58
advanced features
- checkpoints. *See* checkpoints
- event handlers at runtime. *See* event handlers
- expressions. *See* expressions
- logging. *See* logging
- SSIS as data sources for Reporting Services. *See* data sources
- transactions. *See* transactions

Advanced tab, Fuzzy Grouping transformation, 462–463
Advanced tab, Fuzzy Lookup transformation, 466
Aggregate transformation
- exercise using, 431–436
- overview of, 5
- rowset transformations, 429–430

AllowConfigurationChanges property, deployment utility, 588
ALTER INDEX REBUILD, T-SQL statement, 220
ALTER INDEX REORGANIZE, T-SQL statement, 221
Analysis Services Connection Manager, 78–79
Analysis Services Execute DDL task, 139
Analysis Services objects, control flow tasks and, 139
Analysis Services Processing task, 139
Analysis Services task, mapping Dts services to SSIS, 615
analyzing logs, 299
APIs (application programming interfaces)
- programmability features and, 8
- in SSIS, 3

application logs, exercise reading, 197–200
architecture, data warehouse
- centralized EDW with dependent data marts, 539–540
- distributed independent data marts, 540–541
- overview of, 539
- Parallel Data Warehouse, 554–556

architecture, SSIS
data flow, 20, 644–645
engines for managing workflows in SSIS, 4
object model, 20
overview, 18–19
run time, 20
service component, 19
archives
Archiving Downloaded Files package, 175–179
exercise archiving downloaded files, 156–160
exercise deleting data month by month from archives. *See* data deletion exercise
historical data, 657
AssignExpression, For Loops and, 124
asynchronous transformations
blocking asynchronous full row-set-based transformations, 649–650
exercise writing asynchronous output to nonstandard text file, 528–529
partially blocking asynchronous row-set-based transformations, 648–649
performance enhancement with, 645–647
scripting, 520–525
attributes
of data in data warehouse, 441–443
in ER modeling, 541
setting on files and folders, 155
auditing transformations
Audit, 436–437
data flow and, 5, 355
overview of, 436
Row Count, 437–438
auto-generated scripts, 492–494
AVERAGE
as Aggregate transformation, 429–430
exercise aggregating sales orders, 433

B

Back Up Database task
control flow tasks, 212–213
as maintenance task, 140
backup compression, in data warehousing, 557–558
backward compatibility
Client Tools Backward Compatibility, 602
control flow tasks, 141
batch files
calling using Execute Process task, 152–154
creating, 149
BIDS (Business Intelligence Development Studio)
Adding Connection Managers to packages, 75
adding imported package to, 54–55
based on VSTA, 12
comparing with SQL Server Management Studio, 71
creating blank project for use with, 29–31
as development tool, 4–5
digitally signing packages, 271
exploring imported packages, 53
overview, 28–29
Package Configurations Organizer utility in, 569
running packages during development phase, 244–245
SSIS Designer, 111
starting SQL Server Import and Export Wizard, 43
validation alerts in, 632
windows and tab options in, 31–36
binary data option, Flat File source, 347
BLOB Bytes Read, performance counters, 659
BLOB Bytes Written, performance counters, 659
BLOB Files In Use, performance counters, 659
blocking asynchronous full row-set-based transformations, 649–650
bottom-up design, data warehouse, 539
breakpoints
debugging Script task and, 501–503
exercise setting, 635–638
overview of, 634
Buffer Memory, performance counters, 659
buffers, in memory management, 644
Buffers In Use, performance counters, 659
Buffers Spooled, performance counters, 659–660
BufferSizeTuning, logging event, 661
BufferWrapper class, in Script component, 510
Bulk Insert task
configuring to import text files, 183
as control flow tasks, 169
mapping Dts services to SSIS, 615
overview of, 5
as SQL Server task, 138
Business Intelligence Development Studio. *See* BIDS (Business Intelligence Development Studio)
business intelligence transformations
data flow and, 352–353
Data Mining Query, 455–456
exercise configuring SCD transformation, 445–452
exercise executing package for removing duplicates, 478–480
exercise filtering with SCD key, 444–445
exercise loading SCD transformation, 452–455
exercise removing duplicates by Fuzzy Grouping, 474–478
exercise removing exact duplicates, 470–472
exercise removing fuzzy duplicates, 472–474
exercise using SCD transformation, 444
Fuzzy Grouping, 460–463
Fuzzy Lookup, 463–467
overview of, 352–353, 439
SCD (Slowly Changing Dimension), 439–443
Term Extraction, 457–460
Term Lookup, 456–457

C

Cache Connection Manager, 76
Cache Mode section, of lookup transformation, 397–399
cache transformation, as split and join transformation, 394–395
Campaign database, attaching to SQL Server 2008, 674
candidate key profile, for multiple-column profiles, 65
case sensitivity, Term Lookup and, 457
CDC (Change Data Capture)
    data warehouse enhancements in SQL Server 2008 R2, 562–564
    MERGE statement used with, 561
    what's new in SQL Server 2008, 13
change types, available in SCD Wizard, 441–443
changing attribute
    attribute change types in SCD Wizard, 441–442, 546
    configuring SCD transformation, 450
Character Map transformation
    exercise using to convert pipeline data, 407–408
    as row transformation, 379–380
    standardization of data with, 6
Check Database Integrity task
    as control flow task, 213–214
    as maintenance task, 140
CheckFile option, DTExec utility, 253
CheckpointFileName property, 312
CheckPointing option, DTExec utility, 252
checkpoints
    exercise in *see*ing effect of transactions on checkpoints, 316–317
    restarting packages with, 311–313
CheckpointUsage property, 312
CIF (Corporate Information Factory), 538
CIM (Common Information Model), 196
Client Tools Backward Compatibility, 602
Code window, BIDS, 35
column length distribution profile, single-column profiles, 64, 68
column null ratio profile, single-column profiles, 64
column pattern profile, single-column profiles, 64
column statistics profile, single-column profiles, 64
column value distribution profile, single-column profiles, 64
columns
    ADO.NET source, 343
    copying input columns to output columns, 378–379
    creating new columns on row by row basis (row transformation), 353
    derivations on input column data, 382–385
    Excel source, 346
    Export Column transformation, 385–386
    Import Column transformation, 386
    mappings, 58–60
    raw file source, 350
    SQL Server Import and Export Wizard, 45–46
Columns tab, Fuzzy Grouping transformation, 462
Columns tab, Fuzzy Lookup transformation, 465
comma-separated value (CSV)
    exercise writing synchronous output to, 526–528
    text file log provider writing information into, 293
command prompt
    DTExec utility for running packages, 250–256
    installing SSIS from, 25–26
command sourcing phase, DTExec utility, 251
Common Information Model (CIM), 196
Component Wrapper class, in Script component, 510–511
compression, backup compression in SQL Server 2008 R2, 557–558
Conditional Split transformation, as row transformation, 389–390
ConfigFile option, DTExec utility, 253
configuration
    Archiving Downloaded Files for sending filenames, 175–179
    Bulk Insert task for importing text files, 183
    data flow path, 372–375
    Data Flow task, 421–424
    DTExec utility phase, 252–255
    Execute SQL task, 90–92, 115–118
    exercise applying checkpoint configurations to packages, 313–316
    FTP Connection Manager, 145
    FTP task, 143, 144–147
    logging, 296–299
    package. *See* package configurations
    SCD transformation, 445–452
    Transfer Database task, 202–203
    XML task areas, 162–165
configuring Script component
    as data source, 507–514
    as destination, 525–526
    as transformation, 515
conformed dimensions, in dimensional modeling, 544
Connection Manager tab, Fuzzy Grouping transformation, 461–462
Connection Managers
    adding for contacts email campaign, 114–115
    ADO Connection Manager, 75
    ADO.NET Connection Manager, 75–76, 213–214, 343
    Analysis Services Connection Manager, 78–79
    Cache Connection Manager, 76
    Connections Project Wizard and, 63
    displaying all Connection Managers defined in package, 58
    Excel Connection Manager, 76–77, 345
    exercise adding, 89–90, 431
    File Connection Manager, 77–78
    Flat File Connection Manager, 78
    FTP Connection Manager, 78, 143, 145
    HTTP Connection Manager, 78, 160
    Microsoft Connector 1.0 for SAP BI, 81–82
    Microsoft Connector for Oracle by Attunity, 82–83
    Microsoft Connector for Teradata by Attunity, 83–84
    MSMQ Connection Manager, 78

Connection Managers (*cont.*)
Multiple Files Connection Manager, 79
Multiple Flat Files Connection Manager, 79–80
ODBC Connection Manager, 80
OLE DB Connection Manager, 80, 114–115, 369, 461
overview of, 74–75
raw file source, 350
SMO Connection Manager, 80–81
SMTP Connection Manager, 81, 195
SQL Server Compact Edition Connection Manager, 81
types of objects in SSIS packages, 71
WMI Connection Manager, 81, 197–199
Connection option, DTExec utility, 253
connection strings, types of sensitive data, 271
Connections Managers tab, BIDS windows
displaying all Connection Managers defined in package, 58
overview of, 35
connections, mapping Dts services to SSIS, 616
Connections Project Wizard, 63
ConnectionString property, evaluating at runtime, 106
consistency of data. *See* data integrity
ConsoleLog option, DTExec utility, 253–254
constrained executable, 99–100
constraints. *See* precedence constraints
contacts email campaign
adding Connection Managers for, 114–115
adding Foreach Loop Container, 119–120
adding Send Mail Task and executing package, 119–122
configuring Execute SQL Task, 115–118
method, 113
review, 123
control flow
data flow and, 334–335
managing workflow with, 55–56
running parallel task in, 669–670
types of objects in SSIS packages, 70
control flow containers
Event Handler, 110
exercise avoiding inconsistency in single container, 306
exercise avoiding inconsistency over multiple container, 306–310
exercise creating e-mail campaign. *See* contacts email campaign
exercise deleting data month by month from archives. *See* data deletion exercise
Foreach Loop, 111–113, 119–120
For Loop, 123–126
packages, 110–111
Sequence, 131–132
summary, 133
Task Host, 132
types of, 110
control flow engine
comparing with data flow engine, 482–483
engines for managing workflows in SSIS, 3–4
separation from data flow engine, 334–335
in SSIS architecture, 20
working programmatically and, 483
Control Flow tab, BIDS SSIS Designer, 33, 111
control flow tasks
analysis services, 139
Back Up Database task, 212–213
backward compatibility and, 141
Bulk Insert task, 169
Check Database Integrity task, 213–214
custom, 141–142
data flow, 137
data preparation, 137
Execute Package task, 187–188
Execute Process task, 148
Execute SQL Server Agent Job task, 214–215
Execute SQL task, 165–168
Execute T-SQL Statement task, 215–216
exercise archiving downloaded files, 156–160
exercise consolidating workflow packages, 188–195
exercise downloading zipped files from FTP server, 144–148
exercise expanding zipped files, 149–154
exercise importing expanded files. *See* importing expanded zip files
exercise reading application log, 197–200
File System task, 155–156
FTP task, 142–143
History Cleanup task, 216–217
maintaining, 140–141
Maintenance Cleanup task, 217–218
managing from Control Flow pane, 482
Message Queue task, 169–173
Notify Operator task, 218–219
overview, 136
Rebuild Index task, 220–221
Reorganize Index task, 221–222
scripting, 139
Send Mail task, 195
Shrink Database task, 222–223
SQL Server, 138
summary, 223–224
tasks as smallest unit for restarting packages, 312
transfer, 139–140
Transfer Database task, 202–203
Transfer Error Messages task, 203–205
Transfer Jobs task, 205–206
Transfer Logins task, 206–208
Transfer Master Stored Procedures task, 208–209
Transfer SQL Server Objects task, 210–211
Update Statistics task, 223–224
Web Service task, 160–161
WMI Data Reader task, 196–197
WMI Event Watcher task, 200–202
workflow, 138
XML task, 161–165

Copy Column, as row transformation, 378–379
COPY command, dtutil utility, 238–239
Copy SQL Server Objects task, mapping Dts services to SSIS, 615
Corporate Information Factory (CIF), 538
COUNT, as Aggregate transformation, 429–430
COUNT DISTINCT
    as Aggregate transformation, 429–430
    exercise aggregating sales orders, 433
CPUs
    32-bit vs. 64-bit systems, 17, 641–643
    parallel processing. *See* parallel processing
CreateDeploymentUtility property, deployment utility, 588
CRM (customer support systems)
    contact management and, 113
    data management and, 2
cryptography. *See* encryption
CSV (comma-separated value)
    exercise writing synchronous output to, 526–528
    text file log provider writing information into, 293
CUBE operators, GROUP BY supporting, 561
custom deployment, 592–594
customer support systems (CRM)
    contact management and, 113
    data management and, 2
customization. *See* programming SSIS

D

data
    analyzing data quality with Data Profiling Task, 63–64
    consolidation features, 7
    converting raw data into meaningful information, 7
    Data Flow pane managing movement of, 482
    preparing for control flow tasks, 137
    segmentation using Percentage Sampling transformation, 417–418
    standardization features in SSIS, 6
    timestamps, 436
data cleansing
    business intelligence operations, 352
    Fuzzy Grouping transformation and, 460
Data Conversion transformation
    converting data formats or locales, 7
    exercise converting Excel data and combining with flat files, 403–405
    as row transformation, 381–382
    standardization of data, 6
data deletion exercise
    adding Execute SQL task for deleting monthly records, 127–130
    adding For Loop Container for, 125–126
    method, 124–125
    review, 130
Data Driven Query task, mapping Dts services to SSIS, 615
data flow
    ADO.NET as destination, 357
    ADO.NET as source, 343–345
    architecture of, 644–645
    auditing transformations and, 355
    bringing data into, 337–341
    business intelligence transformations and, 352–353
    component interfaces, 335–336
    control flow and, 137, 334–335
    data mining model training destination, 357–358
    DataReader destination, 358
    destinations. *See* destinations, data flow
    dimension processing destination, 358–360
    error output interface, 337
    Excel as destination, 360
    Excel as source, 345–346
    exercise adding OLE DB source and Flat File destination, 370–372
    exercise configuring data flow path and executing package, 372–375
    exercise configuring OLE DB Connection Manager and adding data flow path, 369
    exercise executing package, 412–413
    external metadata interface, 336
    flat files as destination, 360–361
    flat files as source, 346–348
    input interface, 336–337
    multiple data flows with parallel processing, 670
    OLE DB as destination, 361–363
    OLE DB as source, 348–349
    output interface, 337
    overview of, 334
    partition processing destination, 363
    paths. *See* paths, data flow
    raw files as destination, 364–365
    raw files as source, 349–350
    recordset destination, 365
    row transformations, 353
    rowset transformations, 354
    Script components as destination, 365
    Script components as source, 350
    sources, 342–343
    split and join transformations, 354–355
    SQLServer Compact destination, 365–366
    SQLServer destination, 366–367
    summary, 375
    transformations. *See* transformations, data flow
    types of objects in SSIS packages, 70
    XML files as source, 351
data flow engine
    comparing with control flow engine, 482–483
    engines for managing workflows in SSIS, 3–4
    managing pipeline activities with, 57–58
    overview of, 334
    separation from control flow engine, 334–335
    in SSIS architecture, 20
    working programmatically and, 483
Data Flow tab, BIDS SSIS Designer, 33, 111

Data Flow task
checkpoints and, 312–313
creating multiple data flows with parallel processing, 670
exercise adding, 431, 444
exercise configuring, 421–424
exercise enabling logging on, 663–665
extracting data from disparate data sources, 5
overview of, 335
performance and, 644–645
source and destination adapters, 335
data integrity
exercise avoiding inconsistency in single container, 306
exercise avoiding inconsistency over multiple container, 306–310
exercise creating simulation package for data consistency issues, 302–306
maintaining with transactions, 301–302
data marts. *See also* data warehouses
bottom-up design, 539
centralized EDW with dependent data marts, 539–540
dimension types and, 544
distributed independent data marts, 540–541
hub-and-spoke architecture and, 556–557
integrating with parallel data warehouses, 552
star schema and, 547
top-down design, 538
data mining model training, 357–358
Data Mining Prediction task, mapping Dts services to SSIS, 615
Data Mining Query
as Analysis Services task, 139
as business intelligence transformations, 455–456
data model schemas
building star schema, 549–550
overview of, 547
snowflake model, 548–549
star schema, 547–548
data models
dimension types, 544–545
dimensional modeling, 542–544
ER modeling, 541–542
loading dimensions using SCD, 545–546
overview of, 541
data parsing, converting source data to SSIS data types, 338–339
Data Profile Viewer
overview of, 12
review statistics generated by Data Profiling Task, 66–67
reviewing data in XML format, 42
Data Profiling task
analyzing data quality with, 63–64
as data preparation task, 137
exercise profiling imported data, 66–68
function of, 42
multiple-column profiles, 65
overview of, 5
single-column profiles, 64
what's new, 12
Data Protection API (DPAPI), 272
data source views, 85–86
data sources
ADO.NET, 343–345
configuring Script component as, 507–514
as data flow component, 335–336
design-time connections to, 84–85
enabling SSIS as, 328–329
Excel, 345–346
exercise addings, 370–372, 401–403, 444
flat files, 346–348
Import and Export Wizard data source options, 43–44
OLE DB, 348–349
overview of, 342–343
raw files, 349–350
Script components, 350
SSIS as data source for Reporting Services, 328
SSIS packages as, 8, 330–331
XML files, 351
data storage systems, 2
data streams, exercise combining two, 403–405
data types
converting source data to SSIS data types, 338–339, 381–382
exercise converting Excel data and combining with flat files, 403–405
new date and time data types, 14
user-defined variables and, 95
data viewers
as debugging tool, 640
exercises adding, 412, 424–426
data warehouses
architecture, 539
backup compression in SQL Server 2008 R2, 557–558
bottom-up design, 539
building star schema, 549–550
centralized EDW with dependent data marts, 539–540
change data capture, 562–564
data model schemas, 547
data models for, 541
dimension types, 544–545
dimensional modeling, 542–544
distributed independent data marts, 540–541
ER modeling, 541–542
Fast Track Data Warehouse solution, 553–554
GROUP BY extensions, 561
loading data into, 5–6
loading dimension using SCD, 545–546
managing data attributes. *See* SCD (Slowly Changing Dimension) transformation
MERGE statement, 559–561
need for, 536–538
overview of, 536
Parallel Data Warehouse solution, 554–557
partitioned table parallelism, 564–565
snowflake model, 548–549

SQL Server 2008 R2 editions, 551–552
SQL Server 2008 R2 enhancements, 557
SQL Server 2008 R2 solutions, 552
star join query processing, 562
star schema, 547–548
summary, 566
top-down design, 538
database-level roles
applying access control with user-defined roles, 283
exercise using roles for granular access control, 287
exercise using to assign execute permissions to packages, 285–287
permissions and, 282–283
server storage and, 50
databases
attaching Campaign database (for exercises in this book) to SQL Server 2008, 674
backing up. *See* Back Up Database task
checking integrity of. *See* Check Database Integrity task
cleaning up backup or maintenance report files. *See* Maintenance Cleanup task
normalization of data, 419–420
Pivot transformation, 421–424
running T-SQL statements against. *See* Execute T-SQL Statement task
shrinking. *See* Shrink Database task
transferring from source SQL Server to destination SQL Server. *See* Transfer Database task
Unpivot transformation, 427–428
Datacenter Edition, SQL Server 2008 editions, 17
DataReader destination
data flow and, 358
using SSIS package as data source, 8
date data types, 14
date/time functions
in Expression Builder, 104
performing derivations on input column data, 383
DBCC CHECKDB, T-SQL statement, 213–214
DBCC SHRINKDATABASE, T-SQL statement, 222
db_ssisadmin role, 282–283
db_ssisltduser role, 282
db_ssisoperator role
exercise assigning execute permissions to packages, 285–287
overview of, 282
DCOM (Distributed COM)
Message Queuing DCOM Proxy, 172
permissions for accessing packages on remote servers, 229
debugging. *See also* troubleshooting packages
configurable debugging features, 634
default debugging features, 632–633
enhancements to, 14
package configurations, 575–576
Script component, 530–533
Script task, 501–503
decision support system (DSS), 537
default debugging features, 632–633
default settings, package management, 227
degenerate dimensions, in dimensional modeling, 545
DEL, deleting packages with dtutil utility, 240
delimited file formats. *See also* CSV (comma-separated value), 73
deploying packages
custom deployment, 592–594
direct and indirect configurations, 577–578
exercise assigning values to and debugging package configurations, 575–576
exercise building projects, 591
exercise configuring project properties, 590
exercise creating computer-specific package configuration, 579–582
exercise creating indirect package configuration, 583–585
exercise deploying and executing the package, 586
exercise enabling and adding package configurations, 572–575
exercise installing and running project package, 591–592
exercise setting up environment variable as pointer to configuration file, 582
exercise using property expressions to update exported properties, 582–583
overview of, 10, 568
package configurations and, 568–570
Parent Package Variable configuration type, 571
project deployment, 589
Registry Entry configuration type, 571
setting up deployment utility, 587–588
SQLServer configuration type, 571–572
steps in creating indirect configurations, 578–579
summary, 592–594
XML Configuration File, 570–571
deployment utility
overview of, 587–588
properties for controlling project deployment, 588
DeploymentOutputPath property, deployment utility, 588
Derived Column transformation
exercise using, 406–407
as row transformation, 382–385
for standardization of data, 6
design, data warehouses
bottom-up, 539
top-down, 538
designer, SSIS. *See* SSIS Designer
destinations, data flow
ADO.NET destination, 357
bringing data into data flows and, 340–341
configuring Script component as, 525–526
as data flow component, 335–336
data mining model training destination, 357–358
DataReader destination, 358
dimension processing destination, 358–360
Excel destination, 360
exercise adding flat files as, 370–372

destinations, data flow (*cont.*)
exercise writing asynchronous output to nonstandard text file, 528–529
exercise writing synchronous output to CSV file, 526–528
Flat File destination, 360–361
OLE DB destination, 361–363
overview of, 355–357
partition processing destination, 363
raw file destination, 364–365
recordset destination, 365
Script component destination, 365
SQL Server Import and Export Wizard, 46–49
SQLServer Compact destination, 365–366
SQLServer destination, 366–367
Developer Edition, SQL Server 2008 editions, 17
Diff operations, comparing XML documents, 164–165
digital certificates
digitally signing packages, 270–271
package security features, 7–8
dimension hierarchies, in dimensional modeling, 543–544
dimension members, in dimensional modeling, 543
Dimension Processing destination
data flow and, 358–360
data warehousing and, 5
dimension tables, in dimensional modeling, 439
dimensional modeling
dimension tables, 439
dimension types, 544–545
dimensions, 543–544
fact tables, 439, 542–543
loading dimensions using SCD, 545–546
measures, 543
overview of, 542
dimensions
in dimensional modeling, 543–544
loading using SCD, 545–546
types of, 544–545
direct package configurations, 577–578
directories
Active Directory, 172
exercise creating, 95–99
performing operations on using File System tasks, 155
Directory Service Integration, 172
Distributed COM (DCOM)
Message Queuing DCOM Proxy, 172
permissions for accessing packages on remote servers, 229
distributed independent data marts, 540–541
Distributed Management Task Force (DMTF), 196
DML operations, 559
DMTF (Distributed Management Task Force), 196
DMX prediction queries
business intelligence operations, 352
Data Mining Query transformation, 455–456
Document Type Definition (DTD), 163
DPAPI (Data Protection API), 272
DSS (decision support system), 537
DTD (Document Type Definition), 163
DTExec utility
command sourcing phase, 251
configuration phase, 252–255
overview of, 250–251
package load phase, 251–252
validation and execution phase, 256
DTExecUI (Execute Package Utility)
exercise running package with, 246–249
overview, 245
review, 249–250
DTS 2000 package migration
embedding DTS 2000 packages in SSIS packages, 609–610
Execute DTS 2000 Package task, 610
exercise configuring Execute DTS 2000 Package task, 610–614
exercise editing and executing DTS 2000 package, 607–609
exercise enumerating and importing DTS 2000 package, 605–607
exercise executing migrated package, 621–622
exercise using Package Migration Wizard, 618–621
exercise using Upgrade Advisor for analyzing, 597–599
installing Dts support components, 601–604
mapping DTS services to SSIS, 615–617
options for, 600–601
Package Migration Wizard and, 614, 617
running DTS 2000 packages as-is with run-time support, 604
Dts (Data Transformation Services)
ActiveX Script task for extending, 484, 486–487
comparing SSIS to, 270
connecting to SSIS and managing Dts packages, 37–39
legacy support, 10
limitations in package deployment, 568
run time, 602
SQL Server 2008 and, 14–15
upgrading from, 27
Dts Designer Components, 602, 607–608
Dts object, properties and methods of, 495–496
Dts Package Migration Wizard, 601
DtsDebugHost, 245
DTSInstall.exe, installing packages with, 591–592
DTSWizard.exe, starting SQL Server Import and Export Wizard, 43
dtutil utility
copying packages, 238–239
creating folders, 242
deleting folders, 243–244
deleting packages, 240
/Dump option, 14
encrypting packages, 241
importing and exporting packages, 233
listing folder contents, 242
moving packages, 239
overview, 235–236
renaming folders, 243

signing packages, 240–241
verifying existence of packages, 237–238
Dump errorcode option, DTExec utility, 256
Dump option, dtutil utility, 14
DumpOnError option, DTExec utility, 256
Dynamic Properties task, mapping Dts services to SSIS, 615

E

editions, SQL Server 2008, 15–17
EDW (Enterprise Data Warehouse), 539–540
embedding DTS 2000 packages, in SSIS packages, 609–610
encryption
with dtutil utility, 241
exercise encrypting all information with package password, 280–281
exercise encrypting sensitive information with package password, 277–279
exercises encrypting all information with user key, 279–280
exercises encrypting sensitive information with user key, 275–276
package security features, 7–8
ProtectionLevel property options, 272–273
of sensitive data, 49–50
Engine Threads property, parallel processing, 670
Enterprise Data Warehouse (EDW), 539–540
Enterprise Edition, SQL Server 2008 editions, 17
entities, in ER modeling, 541
enumerators, Foreach Loops, 112–113
environment variable
exercise setting up as pointer to package configuration file, 582
types of package configurations, 571
ER (entity-relationship) modeling, 541–542
error events, scripting, 503
Error List window
BIDS windows, 35
default debugging features, 633
error messages, transferring between SQL Servers. *See* Transfer Error Messages task
error output
ADO.NET source, 344
data flow paths and, 639
Excel source, 346
interface for data flow, 337
types of errors and, 339–340
ETL (extracting, transforming, loading) tools
data consolidation with, 42
Data Flow task and, 137
data management and, 2–3
EvalExpression, For Loops and, 124
EvalOp property, constraint options, 100
event handlers
exercise working with, 325–328
handling events at package run time, 323–325
scripting events and, 503
types of objects in SSIS packages, 71
Event Handlers tab, BIDS SSIS Designer, 34, 111, 324
events
handling at package run time, 323–325
logging, 298
raising with scripts, 503–504, 530–532
Excel
data source options for SQL Server Import and Export Wizard, 43
data sources, 345–346
destination in data flow, 360
exercise converting Excel data and combining with flat files, 403–405
exercise using Pivot transformation, 421–424
Excel Connection Manager, 76–77, 345
executables, precedence executable, 99–100
Execute DTS 2000 Package task
as backward compatibility task, 141
exercise configuring Execute DTS 2000 Package task, 610–614
overview of, 610
Execute Package task
control flow tasks, 187–188
embedding DTS 2000 packages in SSIS packages, 609–610
mapping Dts services to SSIS, 615
transactions and, 301
as workflow task, 138
as wrapper for using packages within packages, 308
Execute Package Utility. *See* DTExecUI (Execute Package Utility)
Execute Process task
calling batch files with, 152–154
control flow tasks, 148
mapping Dts services to SSIS, 615
as workflow task, 138
Execute SQL Server Agent Job task
as control flow task, 214–215
as maintenance task, 140
Execute SQL task
configuring for contacts email campaign, 115–118
for deleting monthly records, 127–130
exercise configuring, 90–92
expressions page, 168
general section, 166–168
mapping Dts services to SSIS, 615
overview of, 165–166
parameter mapping page, 168
result set page, 168
as SQL Server task, 138
SQL statements and, 387
Execute T-SQL Statement task
compared with Execute SQL task, 166
control flow tasks, 215–216
as maintenance task, 140
execution trees, as performance monitoring tool, 662–663
ExecutionLog file, 665–668
Export Column transformation, as row transformation, 385–386
Express Edition, SQL Server 2008 editions, 15

Expression Builder
building expressions with, 103
functions and operators, 104–105
expression field, constraint options, 101
expressions
building, 103–104
evaluating, 104, 106
exercise building property expressions for mailing opportunities packages, 318–323
exercise using to update exported properties, 582–583
exercise using to update properties at run time, 105–107
functions and operators, 104–105
uses of, 102
variables and, 317–318
external metadata, interface for data flow, 336
extracting, transforming, loading (ETL) tools
data consolidation with, 42
Data Flow task and, 137
data management and, 2–3

F

fact tables, in dimensional modeling, 439, 542–543
fast parsing
converting source data to SSIS data types, 338–339
Flat File source, 347
Fast Track Data Warehouse solution, in SQL Server 2008 R2, 553–554
FC, creating folders with dtutil utility, 242
FDe, deleting folders with dtutil utility, 243
FDi, showing directory contents in dtutil utility, 242
File Connection Manager, 77–78
File System folder, as root-level folders, 230
File System task
archiving downloaded files, 158–160
configuring with hard-coded values, 105
control flow tasks, 155–156
creating folder using, 96–97
as data preparation task, 137
user-defined variable for passing folder path to, 97–99
File Transfer Protocol task, mapping Dts services to SSIS, 615
files/file system
default settings, 227
delimited file formats, 73
exercise archiving downloaded files, 156–160
exercise downloading zipped files from FTP server, 144–148
exercise expanding zipped files, 149–154
exercise importing expanded files. *See* importing expanded zip files
fixed width file formats, 73
performing operations on using File System tasks, 155
ragged right file formats, 74
saving packages to, 290
fixed attribute, attribute change types in SCD Wizard, 441, 546
fixed database-level roles. *See* database-level roles
fixed width file formats, 73
Flat Buffer Memory, performance counters, 660
Flat Buffers In Use, performance counters, 660
Flat File Connection Manager, 78
Flat File source, 44
flat files
data sources, 346–348
destination in data flow, 360–361
exercise adding as data flow destination, 370–372
importing data from flat file into SQL Server, 169
flat files, importing into SQL Server 2008
column and row options, 45–46
data source selection, 43–44
destination options (new database), 46–49
mapping options, 49
report on steps in import process, 50–51
security options, 49–50
folders. *See also* files/file system
adding root-level folder to stored packages, 232–233
creating with dtutil, 242
deleting with dtutil, 243–244
exercise creating directory folder, 95–99
listing folder contents with dtutil, 242
renaming with dtutil, 243
SSIS service folder structure, 230
For Loops
applying repeating logic to units of work, 7
checkpoints and, 313
container, 123–126
Foreach ADO enumerator, 119–120
Foreach Loops
adding for contacts email campaign, 119–120
applying repeating logic to units of work, 7
archiving downloaded files, 156–158
checkpoints and, 313
enumerating multiple zipped files, 150–152
using Execute Process task inside, 152–154
FR, renaming folders with dtutil utility, 243
FTP Connection Manager
configuring, 145
connecting FTP Task Editor to FTP server, 143
overview of, 78
FTP task
configuring, 143
configuring for downloading zipped files, 144–147
as data preparation task, 137
exercise downloading zipped files from FTP server, 144–148
file transfer operations, 142
full outer joins, T-SQL, 392
functional dependency profile, multiple-column profiles, 65
functions
in Expression Builder, 104–105
performing derivations on input column data, 384
Fuzzy Grouping transformations
Advanced tab, 462–463
Columns tab, 462

Connection Manager tab, 461–462
exercise removing duplicates by Fuzzy Grouping, 474–478
overview of, 460–461
Fuzzy Lookup transformations
Advanced tab, 466
Columns tab, 465
considerations before running, 467
exercise removing fuzzy duplicates, 472–474
operations for standardization of data, 6
overview of, 463–464
Reference Table tab, 464–465

G

GAC (global assembly cache), 485
GROUP BY
as Aggregate transformation, 429–430
data warehouse enhancements in SQL Server 2008 R2, 561
exercise aggregating sales orders, 431–433
GROUPING SETS, 561
GUI (graphical user interface)
for package management, 226
for Script task, 489–492
in SQL Server Management Studio, 229

H

handshake, scripting functionality for in data warehousing scenario, 488–492
hardware, Parallel Data Warehouse, 554–556
historical attribute
attribute change types in SCD Wizard, 442, 546
configuring SCD transformation, 451–452
historical data, archiving, 657
History Cleanup task
as control flow task, 216–217
as maintenance task, 140
HTTP Connection Manager, 78, 160
HTTP, Message Queuing service supporting, 172
hub-and-spoke architecture, Parallel Data Warehouse, 556–557

I

Import Column transformation, as row transformation, 386
importing and exporting packages, 233–235
importing expanded zip files
building expanded files package, 179–186
configuring Archiving Downloaded Files to send filenames, 175–179
installing Message Queuing service, 174–175
method, 173–174
overview of, 173
review, 186–187
indexes
partition-aligned indexed views, 564–565
rebuilding. *See* Rebuild Index task
reorganizing. *See* Reorganize Index task
indirect package configurations
exercise creating, 583–585
overview of, 577–578
steps in creating, 578–579
inferred member updates output
attribute change types in SCD Wizard, 442–443
configuring SCD transformation, 450
informational messages, raising with scripts, 530
InitExpression, For Loops and, 123
Inmon, Bill, 538
inner joins, T-SQL, 392
input
interface for data flow, 336–337
SQL Server Import and Export Wizard, 60–62
Input section, XML task, 162
installing project package, 591–592
installing SSIS
from command prompt, 25–26
overview, 21
side by side install with earlier versions of SSIS or Dts, 26–27
upgrade installation, 27–28
using installation wizard on clean system, 22–25
interfaces, data flow
error output interface, 337
external metadata interface, 336
input interface, 336–337
output interface, 337
overview of, 335–336

J

jobs, SQL Server Agent
cleaning up job history. *See* History Cleanup task
copying jobs. *See* Transfer Job task
creating and adding package to, 257–258
executing. *See* Execute SQL Server Agent Job task
review of job automation process, 265–266
scheduling, 258–261
using proxy account to run jobs, 261–264
viewing job history, 264–265
join transformations. *See* split and join transformations
junk dimensions, in dimensional modeling, 544

K

Kimball, Ralph, 539

L

left outer joins, T-SQL, 392
legacy support, 10, 605–607
LOB (line of business) applications, 536
Locals window
BIDS windows, 36
default debugging features, 633

log providers
overview of, 293–295
types of objects in SSIS packages, 71
log schema, 294
Logger option, DTExec utility, 254
logging
debugging task failures and performance monitoring with, 639
exercise analyzing logs, 299
exercise enabling and configuring, 296–299
exercise enabling logging on Data Flow task, 663–665
exercise reading application log, 197–200
exercise reviewing ExecutionLog file, 665–668
exercise using Script task for, 503–504
exercise using system variables to create custom logs, 87–93
log providers, 293–295
managing SSIS packages and, 9
overview of, 292–293
as performance monitoring tool, 661–662
scripting log information, 532–533
Logical AND/OR constraint
multiple constraints and, 102
setting Logical OR constraint, 498–499
logins, transferring between SQL Servers. *See* Transfer Logins task
lookup features
split and join transformations, 354
what's new, 11
lookup transformation
exercise using, 407–410
Fuzzy Lookup transformation and, 463–464
improved in SSIS 2008, 11
operations for standardization of data, 6
as split and join transformation, 395–400

M

Main class, in Script component, 511
Maintenance Cleanup task
as control flow task, 217–218
as maintenance task, 140
maintenance, in control flow tasks, 140–141
managed code
building custom objects from scratch, 485–486
vs. native code, 483
management tools, SSIS, 4–5
mappings
columns, 58–60
SQL Server Import and Export Wizard and, 49
mathematical functions
in Expression Builder, 104
performing derivations on input column data, 383
MaxConcurrent option, DTExec utility, 255
MAXIMUM, as Aggregate transformation, 429–430
measures, in dimensional modeling, 543
memory management
64-bit systems and, 641–643
buffers, 644
performance enhancement and, 640–641
Merge Join transformation
data consolidation transformations, 7
inner joins, left outer joins, full outer joins, 392–393
split and join transformations, 392–394
Merge operations, joining two XML documents, 164
MERGE statement, T-SQL, 13–14, 559–561
Merge transformation
data consolidation transformations, 7
split and join transformations, 391–392
Message Queue task
control flow tasks, 169–173
installing Message Queuing service, 174–175
mapping Dts services to SSIS, 616
as workflow task, 138
Message Queuing DCOM Proxy, 172
Message Queuing Server, 172
Message Queuing service
components of, 172
installing, 174–175
Message Queuing Triggers, 172
metadata
interface for data flow, 336
Script task vs. Script component and, 506
Microsoft Access, as data source, 43
Microsoft Connector 1.0 for SAP BI, 81–82
Microsoft Connector for Oracle by Attunity, 82–83
Microsoft Connector for Teradata by Attunity, 83–84
Microsoft Distributed Transaction Coordinator (MSDTC), 301
Microsoft Excel. *See* Excel
Microsoft Message Queuing (MSMQ), 169
Microsoft Visual Basic 2008, scripting with, 488, 505
Microsoft Visual C#, scripting with, 488, 505
migration to SSIS 2008
different server options for upgrading SSIS 2005, 625–626
embedding DTS 2000 packages in SSIS packages, 609–610
Execute DTS 2000 Package task, 610
exercise configuring Execute DTS 2000 Package task, 610–614
exercise editing and executing DTS 2000 package, 607–609
exercise enumerating and importing DTS 2000 package, 605–607
exercise executing migrated package, 621–622
exercise installing Upgrade Advisor, 597
exercise using Package Migration Wizard, 618–621
exercise using Upgrade Advisor to analyze DTS 2000 packages, 597–599
installing DTS 2000 support components, 601–604
mapping Dts services to SSIS, 615–617
options for migrating DTS 2000 packages, 600–601
overview of, 596
Package Migration Wizard and, 614, 617
running DTS 2000 packages as-is with run-time support, 604
same server options for upgrading SSIS 2005, 623–625
summary, 630
Upgrade Advisor and, 596

upgrading SSIS 2005, 622–623
upgrading SSIS 2005 packages following migration, 626–630
MINIMUM, as Aggregate transformation, 429–430
MOVE, dtutil utility, 239
MPP (massively parallel processing). *See also* parallel processing
hub-and-spoke architecture and, 556–557
SQL Server 2008 R2 Parallel Data Warehouse, 552, 554–555
MSDB database, roles available in, 282–283
MSDB folder
importing packages to MSDB folder, 233–235
as root-level folders, 230
MSDTC (Microsoft Distributed Transaction Coordinator), 301
MSMQ Connection Manager, 78
MSMQ (Microsoft Message Queuing), 169
Multicast transformation, 390
multicasting, Message Queuing service supporting, 172
multiple-column profiles, Data Profiling Task, 65
Multiple Files Connection Manager, 79
Multiple Flat Files Connection Manager, 79–80

N

namespaces, variable, 93–94
native code, vs. managed code, 483
nested transactions, 301
.NET Framework
list of common classes, 494
Script task calling .NET libraries, 488
.NET Framework Data Provider for Oracle, as data source, 43
nonblocking synchronous row-based transformations, 647–648
normalization of data
CIF (Corporate Information Factory) and, 538
in ER modeling, 542
exercise using Pivot transformation, 421–424
Pivot transformation, 419–420
in Term Extraction, 460
Unpivot transformation, 427–428
Notify Operator task
as control flow task, 218–219
as maintenance task, 140
NotSupported, TransactionOption property, 301
Null functions
in Expression Builder, 104
performing derivations on input column data, 383

O

object model
building packages programmatically, 486
developing custom objects from scratch, 485–486
in SSIS architecture, 20
types of objects in SSIS packages, 70–71
ODBC Connection Manager, 80
ODS (operational data stores), 537–538
OLAP (online analytical processing)
data warehousing and, 439
SSAS tool for snowflaked queries, 548
OLE DB
data sources, 348–349
as destination for imported data, 60–61
destination in data flow, 361–363
exercise adding as data flow source, 370–372, 444
OLE DB command transformation
exercise using to delete duplicates, 410–411
row transformations, 387–388
OLE DB Connection Manager
adding for contacts email campaign, 114–115
exercise configuring and adding data flow path, 369
Fuzzy Grouping transformation and, 461
overview of, 80
OLE DB Provider for SQL Server, as data source, 44
OLTP (online transaction processing)
data warehousing and, 439
support for LOB applications, 536–538
online analytical processing (OLAP)
data warehousing and, 439
SSAS tool for snowflaked queries, 548
online transaction processing (OLTP)
data warehousing and, 439
support for LOB applications, 536–538
operational data stores (ODS), 537–538
Operations Options section, XML task, 162–163
operators
in Expression Builder, 105
performing derivations on input column data, 383–384
optimization techniques. *See also* performance enhancements
archiving historical data, 657
avoiding resource contentions, 657
choosing right operation, 651–654
discussing and adopting best practices, 657–658
limiting work only to what is needed, 654–656
overview of, 650
running packages at most suitable location, 656
running packages at most suitable machine, 656–657
testing, measuring, and recording, 658
Oracle, Microsoft Connector for, 82–83
ORDER BY clause, T-SQL, 415
output
interface for data flow, 337
SQL Server Import and Export Wizard, 60–62
Output section, XML task, 162
Output window
BIDS windows, 35
default debugging features, 633

P

Package Configuration Wizard
building package configurations with, 569–570
Environment Variable configuration type, 571
Parent Package Variable configuration type, 571
Registry Entry configuration type, 571
SQLServer configuration type, 571–572

Package Configuration Wizard (*cont.*)
updating package components, 10
XML Configuration File configuration type, 570–571
package configurations
direct and indirect, 577–578
exercise assigning values to and debugging, 575–576
exercise creating computer-specific, 579–582
exercise creating indirect package configuration, 583–585
exercise deploying and executing the package, 586
exercise enabling and adding, 572–575
exercise setting up environment variable as pointer to configuration file, 582
exercise using property expressions to update exported properties, 582–583
overview of, 568–570
Parent Package Variable configuration type, 571
Registry Entry configuration type, 571
SQLServer configuration type, 571–572
steps in creating indirect configurations, 578–579
XML Configuration File, 570–571
Package Configurations Organizer utility, in BIDS, 569
Package Explorer tab, BIDS SSIS Designer, 34, 111
Package Installation Wizard, 591–592
package load phase, DTExec utility, 251–252
Package Migration Wizard
exercise using Package Migration Wizard, 618–621
migration to SSIS 2008, 614
options for starting, 617
Package Protection levels, SQL Server Import and Export Wizard, 49
package store, saving packages to, 230–232
Package Upgrade Wizard, 13
packages
building consolidated, 189–192
building programmatically, 486
control flow and, 110–111
copying with dtutil, 238–239
as data source, 8
deleting with dtutil, 240
deploying. *See* deploying packages
encrypting, 241
enhancing performance of. *See* performance enhancements
Execute Package task, 187–188
importing and exporting, 233–235
logging. *See* logging
managing, 9
managing on instances of SQL Server, 228
managing packages saved on remote servers, 227–228
managing using default settings, 227
migrating DTS 2000 packages. *See* DTS 2000 package migration
migrating SSIS 2005 packages. *See* SSIS 2005 package migration
moving with dtutil, 239
running. *See* running packages
saving to SSIS package store, 230–232
securing. *See* securing packages
signing with dtutil, 240–241
top-level object in SSIS component hierarchy, 92–99
transactions. *See* transactions
troubleshooting. *See* troubleshooting packages
verifying existence of with dtutil utility, 237–238
Parallel Data Warehouse
architecture and hardware, 554–556
hub-and-spoke architecture and, 556–557
overview of, 552, 554
in SQL Server 2008 R2, 554–557
Parallel Data Warehouse Edition, SQL Server 2008 editions, 17
parallel processing. *See also* performance enhancements
creating multiple data flows, 670
enhancing engine threads, 670
MPP (massively parallel processing), 552, 554–557
overview of, 668–669
running parallel task in control flow, 669–670
Parent Package Variable configuration type, types of package configurations, 571
parsing, converting source data to SSIS data types, 338–339
partially blocking asynchronous row-set-based transformations, 648–649
partition-aligned indexed views, 564–565
Partition Processing Destination
data flow and, 363
data warehousing and, 5
partitioned table parallelism, data warehouse enhancements in SQL Server 2008 R2, 564–565
passwords
encrypting packages with, 241
encrypting sensitive data with, 49–50
exercise encrypting all information with package password, 280–281
exercise encrypting sensitive information with package password, 277–279
mapping Dts services to SSIS, 616
ProtectionLevel property encryption options, 272–273
types of sensitive data, 271
Patch operations, XML documents and, 165
paths, data flow
adding data viewer to, 412
bringing data into data flows and, 341
error output and, 639
exercise adding OLE DB source and Flat File destination, 370–372
exercise configuring data flow path and executing package, 372–375
exercise configuring OLE DB Connection Manager and adding data flow path, 369
overview of, 367–368
Percentage Sampling transformation, as rowset transformation, 417–418
performance counters
managing SSIS packages and, 9
monitoring performance with, 659–660
performance enhancements. *See also* parallel processing
64-bit systems and, 641–643
architecture of data flow, 644–645

archiving historical data, 657
avoiding resource contentions, 657
blocking asynchronous full row-set-based transformations, 649–650
choosing right operation, 651–654
classifying transformations, 647
discussing and adopting best practices, 657–658
limiting work only to what is needed, 654–656
memory management and, 640–641
nonblocking synchronous row-based transformations, 647–648
optimization techniques, 650
overview of, 640
partially blocking asynchronous row-set-based transformations, 648–649
running packages at most suitable location, 656
running packages at most suitable machine, 656–657
synchronous and synchronous transformations, 645–647
testing, measuring, and recording, 658
performance monitoring tools
execution trees, 662–663
exercise enabling logging on Data Flow task, 663–665
exercise reviewing ExecutionLog file, 665–668
logging, 661–662
overview of, 658
performance counters, 9, 659–660
SQL Server Profiler, 660
permissions. *See also* access control
for accessing packages on remote servers, 229
exercise assigning execute permissions to packages, 285–287
exercise using roles for granular access control, 287
fixed database-level roles and, 282–283
pipeline. *See* data flow
PipelineBufferLeak, logging event, 661
PipelineComponentTime, logging event, 661
PipelineExecutionPlan, logging event, 661–662
PipelineExecutionTrees, logging event, 662
PipelineInitialization, logging event, 662
pivot key values, 420
Pivot transformation
Data Flow Transformations, 5
exercise using, 421–424
rowset transformations, 419–420
pivoting, 419
precedence constraints
Constraint Options, 100–101
controlling workflow with, 639
data flow paths and, 367
exercise using, 192–194
multiple constraints, 102
overview of, 99–100
setting Logical OR constraint, 498–499
precedence executable, 99–100
Private Buffer Memory, performance counters, 660
Private Buffers In Use, performance counters, 660
Process tab or Execution Result tab, BIDS windows, 34
programming SSIS
ActiveX task (legacy scripting), 486–487
comparing Script task and Script component, 506–507
configuring Script component as data source, 507–514
configuring Script component as destination, 525–526
configuring Script component as transformation, 515
control flow tasks, 141–142
data flow control flow engines and, 482–483
executing packages programmatically, 266–267
exercise adding code in script projects, 494–501
exercise debugging Script component, 530–533
exercise debugging Script task, 501–503
exercise in understanding auto-generated script, 492–494
exercise raising events and logging in Script task, 503–504
exercise using system variables to create custom logs, 87–93
exercise working with Script task, 489–492
exercise writing asynchronous output to nonstandard text file, 528–529
exercise writing synchronous output to CSV file, 526–528
options for, 483–486
overview of, 8–9, 482
review of Script task, 504–505
Script component, 505
Script task, 488
scripting a synchronous transformation, 516–520
scripting an asynchronous transformation, 520–525
Progress tab, default debugging features, 633
project deployment. *See also* deploying packages
exercise building projects, 591
exercise configuring project properties, 590
exercise installing and running project package, 591–592
overview of, 589
projects
building with Solution Explorer, 591
exercise creating new, 295
solutions and, 71–73
properties, configuring project properties before deploying, 590
Properties window, BIDS windows, 32
property expressions. *See* expressions
protection levels, packages. *See* securing packages
ProtectionLevel property
DontSaveSensitive option, 274–275
EncryptAllWithPassword option, 280–281
EncryptAllWithUserKey option, 279–280
encrypting all information, 273
encrypting sensitive information, 241, 272–273
EncryptSensitiveWithPassword option, 277–279
EncryptSensitiveWithUserKey option, 275–276
excluding sensitive information, 271–272
exercise applying ServerStorage ProtectionLevel to package, 284–285
ServerStorage option, 282
proxy account, using to run SQL Server Agent jobs, 261–264

R

R2 Premium Editions, SQL Server 2008 editions, 17
ragged right file formats, 74
raw files
    data sources, 349–350
    destination in data flow, 364–365
Reader Role, 287
Rebuild Index task
    as control flow task, 220–221
    as maintenance task, 141
recordsets
    destination in data flow, 365
    rowset transformations. *See* rowset transformations
Reference Table tab, Fuzzy Lookup transformation, 464–465
Registry Entry configuration type, types of package configurations, 571
relationships, in ER modeling, 541
remote servers
    connecting to SSIS service, 228–229
    managing packages saved on, 227–228
Reorganize Index task
    as control flow task, 221–222
    as maintenance task, 141
Reporting Services. *See* SSRS (SQL Server Reporting Services)
reports
    DTExec utility, 254
    scripting report events, 503
    on steps in import process, 50–51
Required, TransactionOption property, 300, 316–317
Restart option, DTExec utility, 254–255
restarting packages
    with checkpoints, 311–313
    exercise applying checkpoint configurations to packages, 313–316
    exercise in *see*ing effect of transactions on checkpoints, 316–317
    managing SSIS packages and, 9
role-playing dimensions, in dimensional modeling, 545
roles
    applying access control with user-defined roles, 283
    exercise using roles for granular access control, 287
    fixed database-level roles and permissions, 282–283
    server storage and, 50
ROLLUP operators, GROUP BY supporting, 561
root-level folders
    adding to stored packages, 232–233
    SSIS service folder structure, 230
routing, Message Queuing service supporting, 172
Row Count transformation, as auditing transformation, 437–438
Row Sampling transformation, as rowset transformation, 418–419
row transformations
    Character Map, 379–380
    Copy Column, 378–379
    Data Conversion, 381–382
    Derived Column, 382–385
    exercise using Character Map transformation, 407–408
    exercise using Derived Column transformation, 406–407
    exercise using OLE DB command, 411
    Export Column, 385–386
    Import Column, 386
    nonblocking synchronous row-based transformations, 649–650
    OLE DB command, 387–388
    overview of, 353, 378
    Script component, 386–387, 505
rows
    counting, 355
    sorting, 415–417
    SQL Server Import and Export Wizard, 45–46
Rows Read, performance counters, 660
Rows Written, performance counters, 660
rowset transformations
    Aggregate, 429–430
    blocking asynchronous full row-set-based transformations, 649–650
    exercise adding data viewers and executing package, 424–426
    exercise using Aggregate transformation, 431–436
    exercise using Pivot transformation, 421–424
    overview of, 354, 415
    partially blocking asynchronous row-set-based transformations, 648–649
    Percentage Sampling, 417–418
    Pivot, 419–420
    Row Sampling, 418–419
    Sort, 415–417
    Unpivot, 427–428
RSReportDesigner.Config, 328–329
RSReportServer.Config, 328
run time. *See also* control flow
    exercise working with event handlers at, 325–328
    handling events at package run time, 323–325
run-time engine. *See* control flow engine
running packages
    BIDS for, 244–245
    DTExec utility for, 250–256
    DTExecUI for, 245–250
    programmatically, 266–267
    SQL Server Agent for, 256–266
    SQL Server Import and Export Wizard for, 244
    utilities for, 244
Running Packages folder, 230

S

SAP BI, Microsoft Connector 1.0 for, 81–82
SaveCheckpoints property, 312
saving packages
    to file systems, 290
    to SQL Server, 289

SCD (Slowly Changing Dimension) transformation
as business intelligence transformation, 439–443
data warehousing and, 6
exercise configuring SCD transformation, 445–452
exercise filtering with SCD key, 444–445
exercise loading SCD transformation, 452–455
exercise using SCD transformation, 444
loading dimensions using, 545–546
SCD Wizard
attribute change types in, 441–443
configuring SCD transformation, 445–452
schedules, creating for package execution, 258–261
scope, of user-defined variables, 94
Script component
comparing Script task with, 506–507
configuring as data source, 507–514
configuring as destination, 525–526
configuring as transformation, 515
in Control Flow and Data Flow environments, 9
data sources, 350
debugging Script component, 530–533
destination in data flow, 365
exercise writing asynchronous output to nonstandard text file, 528–529
exercise writing synchronous output to CSV file, 526–528
improvements to, 12
overview of, 505
as row transformations, 386–387
scripting a synchronous transformation, 516–520
scripting an asynchronous transformation, 520–525
Script task
exercise adding code in script projects, 494–501
exercise debugging, 501–503, 506–507
exercise in understanding auto-generated script, 492–494
exercise raising events and logging in, 503–504
exercise working with GUI for, 489–492
overview of, 488
review, 504–505
scripting
control flow tasks and, 139
extending packages with, 486
options for, 484–485
Script component. *See* Script component
Script task. *See* Script task
SecondOperand section, XML task, 162
securing packages
applying access control with user-defined roles, 283
digitally signing, 270–271
encrypting all information, 273
encrypting sensitive information, 272–273
excluding sensitive information, 271–272
exercise applying ServerStorage ProtectionLevel to package, 284–285
exercise encrypting all information with package password, 280–281
exercise encrypting sensitive information with package password, 277–279
exercise excluding sensitive information, 274–275
exercise using assigning execute permissions to packages, 285–287
exercise using roles for granular access control, 287
exercises encrypting all information with user key, 279–280
exercises encrypting sensitive information with user key, 275–276
features for, 7–8
fixed database-level roles and permissions, 282–283
overview of, 270
saving to file systems and, 290
saving to SQL Server and, 289
storage areas and, 288–289
summary, 290
working with package protection levels, 273–274
security options, SQL Server Import and Export Wizard, 49–50
Send Mail task
adding for contacts email campaign, 119–122
control flow tasks, 195
mapping Dts services to SSIS, 616
as workflow task, 138
sensitive information
excluding, 271–272
exercise encrypting sensitive information with package password, 277–279
exercise excluding, 274–275
exercises encrypting with user key, 275–276
Package Protection levels, 49
in SSIS, 270
Sequence Container, 131–132
ServerStorage ProtectionLevel
overview of, 282
securing packages, 284–285
Service-Oriented Architecture (SOA), SSIS support for, 8
service, SSIS
connecting to, 19, 226
connecting to on remote servers, 228–229
folder structure of, 230
mapping Dts services to SSIS, 615–617
set keys, for pivoting, 420
Set option, DTExec utility, 255
Shrink Database task
as control flow task, 222–223
as maintenance task, 141
side-by-side upgrades
with earlier versions of SSIS or Dts, 26–27
same server option for upgrading SSIS 2005, 623–624
SIGN, signing packages with dtutil utility, 240–241
simulation package, exercise creating for data consistency issues, 302–306
single-column profiles, Data Profiling task, 64
Slowly Changing Dimension (SCD) transformation. *See* SCD (Slowly Changing Dimension) transformation

SMO Connection Manager, 80–81
SMP (symmetric multiprocessor)
Fast Track Data Warehouse solution, 553
hub-and-spoke architecture and, 556
SMTP Connection Manager, 81, 195
snowflake model, data models, 548–549
SOA (Service-Oriented Architecture), SSIS support for, 8
software, downloading software accompanying this book, 674
Solution Explorer window
BIDS windows, 32
building projects, 591
solutions, as project container, 71–73
Sort transformation
Data Flow transformations, 5
rowset transformations, 415–417
split and join transformations
cache, 394–395
Conditional Split, 389–390
exercise combining two data streams, 403–405
exercise using lookup, 407–410
lookup, 395–400
Merge, 391–392
Merge Join, 392–394
Multicast, 390
overview of, 354–355, 389
Union All, 391
SQL commands
Excel destination, 360
loading result sets of, 62
OLE DB command transformation and, 387–388
OLE DB destination, 363
OLE DB source and, 349
running. *See* Execute SQL task
SQL Server
32-bit vs. 64-bit versions of SQL Server 2008, 17
control flow tasks, 138
destination in data flow, 366–367
editions, 15–17
importing data from flat files into SQL Server, 169
log provider for, 293
managing packages on instances of, 228
saving packages to, 289
transferring objects between SQL Servers. *See* Transfer SQL Server Objects task
upgrading SQL Server 2005 simultaneously with SSIS 2005, 624–625
upgrading to SQL Server 2008, 596
SQL Server 2008 R2
backup compression, 557–558
change data capture, 562–564
data warehouse enhancements, 557
data warehouse solutions, 552
Datacenter, 551–552
editions, 551–552
Fast Track Data Warehouse solution, 553–554
GROUP BY extensions, 561
MERGE statement, 559–561
Parallel Data Warehouse, 552
Parallel Data Warehouse solution, 554–557
partitioned table parallelism, 564–565
star join query processing, 562
SQL Server Agent
automating running of packages, 256–257
cleaning up job history. *See* History Cleanup task
copying jobs. *See* Transfer Job task
creating new job and adding package to, 257–258
creating schedule for package execution, 258–261
executing jobs. *See* Execute SQL Server Agent Job task
notification messages to. *See* Notify Operator task
review of job automation process, 265–266
using proxy account to run jobs, 261–264
viewing job history, 264–265
SQL Server Compact Edition Connection Manager, 81
SQL Server Import and Export Wizard
adding imported package to BIDS, 54–55
Advanced Editor, 58
column and row options, 45–46
column mappings, 58–60
control flow and, 55–56
data flow and, 57–58
data source selection, 43–44
destination options (new database), 46–49
executing program following import using SQL Server Management Studio, 51–53
exploring imported package using BIDS, 53
extending, 12
Flat File source, 44
input and output options, 60–62
managing SSIS packages and, 9
mapping options, 49
report on steps in import process, 50–51
for running packages, 244
security options, 49–50
starting, 43
variables and, 317
SQL Server Installation Wizard, 22–25
SQL Server Management Studio
analyzing logs, 299
comparing with BIDS, 71
connecting to SSIS and managing Dts packages, 37–39
connecting to SSIS service, 19, 226
connecting to SSIS service and viewing folder structure, 230
executing program following import using, 51–53
exercise enumerating and importing DTS 2000 package, 605–607
GUI interface for package management, 229
as management tool, 4–5
overview, 37
starting Import and Export wizard, 43
SQL Server Native Client 10.0, 44

SQL Server Profiler
log provider for, 293
as performance monitoring tool, 660
SQL Server Reporting Services (SSRS)
enabling SSIS as data source, 328–329
using SSIS as data source, 330–331
SQLServer Compact, destination in data flow, 365–366
SQLServer configuration type, types of package configurations, 571–572
SSAS tool, for snowflaked queries, 548
SSIS 2005 package migration
different server options, 625–626
overview of, 622–623
same server options, 623–625
upgrading from, 27
upgrading packages following migration, 626–630
SSIS Designer, in BIDS, 4–5, 31–32, 111
SSIS Object Model, 266–267
SSIS Package Upgrade Wizard, 627–629
SSIS (SQL Server Integration Services), features and uses
administrative tasks, automating, 10
architecture, 4
converting raw data into meaningful information, 7
data consolidation features, 7
data standardization features, 6
deployment features, 10
designer and management tools, 4–5
legacy support, 10
loading data into data warehouse, 5–6
overview, 2–3
package management features, 9
package security features, 7–8
programmability features, 8–9
scripting components, 9
SOA (Service-Oriented Architecture) support, 8
SSIS packages as data source, 8
SSIS (SQL Server Integration Services), what's new
ADO NET components improved, 12
Change Data Capture, 13
data profiles, 12
date and time data types, new, 14
debugging enhancements, 14
lookup features, 11
overview, 10–11
Package Upgrade Wizard, 13
scripting components, 12
SQL Server Import and Export Wizard extended, 12
T-SQL merge statement, 13–14
thread allocation, 13
SSISDeploymentManifest file, installing packages with, 591–592
SSRS (SQL Server Reporting Services)
enabling SSIS as data source, 328–329
using SSIS as data source, 330–331
Standard Edition, SQL Server 2008 editions, 16–17
standard parsing, converting source data to SSIS data types, 338–339
star join query processing, 562
star schema
building, 549–550
data models, 547–548
stemming, in Term Extraction, 460
storage
adding root-level folder to stored packages, 232–233
saving packages to SSIS package store, 230–232
saving to file systems, 290
saving to SQL Server, 289
securing packages and, 288–289
understanding storage areas, 230
Stored Packages folder, 230
stored procedures
running. *See* Execute SQL task
transferring between SQL Servers. *See* Transfer Master Stored Procedures task
string functions
Character Map transformation and, 379
in Expression Builder, 104
performing derivations on input column data, 383
SUM
as Aggregate transformation, 429–430
exercise aggregating sales orders, 433
Sum option, DTExec utility, 255
Supported, TransactionOption property, 301
symmetric multiprocessor (SMP)
Fast Track Data Warehouse solution, 553
hub-and-spoke architecture and, 556
synchronous transformations
exercise writing synchronous output to CSV file, 526–528
nonblocking synchronous row-based transformations, 647–648
performance enhancement with, 645–647
scripting, 516–520
system variables
exercise using to create custom logs, 87–93
list of, 88
namespaces for, 93–94
overview of, 87

T

T-SQL
ALTER INDEX REBUILD, 220
ALTER INDEX REORGANIZE, 221
DBCC SHRINKDATABASE statement, 222
Merge Join transformations compared with joins, 392
MERGE statement, 13–14, 559–561
ORDER BY clause, 415
running T-SQL statements against database. *See* Execute T-SQL Statement task
UPDATE STATISTICS, 223

tables
Excel destination, 360
loading imported data into, 61–62
OLE DB destination, 361–363
OLE DB source and, 349
partitioned table parallelism, 564–565
tagging, in Term Extraction, 460
Task Editor, File System task, 155
Task Host Container, 132
Task List window, BIDS windows, 35
tasks, control flow. *See* control flow tasks
Teradata, Microsoft Connector for, 83–84
Term Extraction transformation, as business intelligence transformation, 457–460
Term Lookup transformation, as business intelligence transformation, 456–457
text
Term Extraction, 457–460
Term Lookup, 456–457
text files
exercise writing asynchronous output to nonstandard text file, 528–529
exercise writing synchronous output to CSV file, 526–528
log provider for, 293
text mining, business intelligence operations, 352
thread allocation
enhancing engine threads, 670
what's new, 13
time data types, 14
timestamps, Audit transformation and, 436
tokenizing, in Term Extraction, 459
Toolbox, BIDS windows, 32–33
top-down design, of data warehouses, 538
trace listeners, using as alternative to breakpoints, 507
TransactionOption property, 300–301, 316–317
transactions
exercise avoiding inconsistency in single container, 306
exercise avoiding inconsistency over multiple container, 306–310
exercise creating simulation package for data consistency issues, 302–306
exercise in seeing effect of transactions on checkpoints, 316–317
maintaining data integrity with, 301–302
overview of, 300–301
review, 310
Transfer Database task
configuring, 202–203
as transfer task, 139
Transfer Error Messages task
as control flow task, 203–205
as transfer task, 139
Transfer Job task
as control flow task, 205–206
as transfer task, 140
Transfer Logins task
as control flow task, 206–208
as transfer task, 140
Transfer Master Stored Procedures task
as control flow task, 208–209
as transfer task, 140
Transfer SQL Server Objects task
as control flow task, 210–211
as transfer task, 140
transfer tasks
control flow tasks, 139–140
mapping Dts services to SSIS, 616
Transform Data task, mapping Dts services to SSIS, 616
transformations, data flow
asynchronous. *See* asynchronous transformations
auditing. *See* auditing transformations
blocking asynchronous full row-set-based transformations, 649–650
bringing data into data flows and, 339–340
business intelligence. *See* business intelligence transformations
classifying, 647
configuring Script component as, 515
as data flow component, 335–336
Data Flow pane managing, 482
exercise adding new package and Excel Connection Manager, 421
exercise deleting duplicates, 407–410
nonblocking synchronous row-based transformations, 647–648
overview of, 351–352
partially blocking asynchronous row-set-based transformations, 648–649
row. *See* row transformations
rowset. *See* rowset transformations
scripting a synchronous transformation, 516–520
scripting an asynchronous transformation, 520–525
split and join. *See* split and join transformations
synchronous. *See* synchronous transformations
types of, 5
Triple DES encryption algorithm, 272
troubleshooting packages
configurable debugging features, 634
data viewers, 640
default debugging features, 632–633
exercise setting breakpoints, 635–638
logging features, 639
overview of, 632
precedence constraints and data flow paths and, 639
TRY/CATCH blocks, 504
type casts
in Expression Builder, 104
performing derivations on input column data, 383

U

Union All transformation, as split and join transformation, 391
Union, as data consolidation transformation, 7
Unpivot transformation, as rowset transformation, 427–428
UPDATE STATISTICS, T-SQL statement, 223
Update Statistics task
    as control flow task, 223–224
    as maintenance task, 141
Upgrade Advisor
    exercise installing, 597
    exercise using to analyze DTS 2000 packages, 597–599
    upgrading to SQL Server 2008, 596
upgrades
    DTS 2000 packages. *See* DTS 2000 package migration
    exercise installing Upgrade Advisor, 597
    exercise using Upgrade Advisor to analyze DTS 2000 packages, 597–599
    overview of, 27–28
    Package Upgrade Wizard, 13
    to SQL Server 2008, 596
    SSIS 2005 packages. *See* SSIS 2005 package migration
    Upgrade Advisor, 596
user-defined variables
    data types and values and, 95
    exercise creating directory with, 95–99
    namespaces for, 93–94
    overview, 92
    scope of, 94
user keys
    encrypting sensitive data with, 50
    exercises encrypting all information with user key, 279–280
    exercises encrypting sensitive information with user key, 275–276
    ProtectionLevel property encryption options, 272–273
utilities, for managing SSIS packages, 9

V

Validate option, DTExec utility, 256
Validate section, XML task, 163
validation and execution phase, DTExec utility, 256
value inclusion profile, multiple-column profiles, 65
value property, constraint options, 100–101
values, user-defined variables and, 95
variables
    creating user variable, 118
    expressions and, 317–318
    mapping Dts services to SSIS, 616
    overview of, 86–87
    system variables, 87–92
    types of objects in SSIS packages, 71
    user-defined variables, 92–99
VerifyBuild option, DTExec utility, 255
VerifyPackageID option, DTExec utility, 255
VerifySigned option, DTExec utility, 255
VerifyVersionID option, DTExec utility, 255
views, loading imported data into, 61–62
Visual Basic 2008, scripting with, 488, 505
Visual C#, scripting with, 488, 505
Visual Studio 2008, BIDS built on, 28
Visual Studio Conversion Wizard, 626–627
VSA (Visual Studio for Applications), 488
VSTA (Visual Studio Tools for Applications)
    BIDS based on, 12
    Script task and Script component using, 488
    as scripting environment, 484

W

WarnAsError option, DTExec utility, 256
warnings, scripting events, 503
Watch window
    BIDS windows, 36
    default debugging features, 633
WBEM (Web-Based Enterprise Management), 196
Web Edition, SQL Server 2008 editions, 16
Web Service Description Language (WSDL), 161
Web Service task
    as control flow task, 160–161
    as data preparation task, 137
Windows Application Event Log, 294
Windows Management Instrumentation (WMI), 196
wizards
    Connections Project Wizard, 63
    Dts Package Migration Wizard, 601
    for managing SSIS packages, 9
    overview, 42
    Package Configuration Wizard. *See* Package Configuration Wizard
    Package Installation Wizard, 591–592
    Package Migration Wizard, 614, 617
    Package Upgrade Wizard, 13
    SCD Wizard, 441–443, 445–452
    SQL Server Import and Export Wizard. *See* SQL Server Import and Export Wizard
    SQL Server Installation Wizard, 22–25
    SSIS Package Upgrade Wizard, 627–629
    Visual Studio Conversion Wizard, 626–627
WMI Connection Manager, 81, 197–199
WMI Data Reader task
    as control flow task, 196–197
    as workflow task, 138
WMI Event Watcher task
    as control flow task, 200–202
    as workflow task, 138
WMI Query Language (WQL), 196

WMI (Windows Management Instrumentation), 196
workflows
    Connection Managers. *See* Connection Managers
    control flow tasks, 138, 482
    data source view, 85–86
    data sources, 84–85
    exercise consolidating workflow packages, 188–195
    expressions, 102–107
    file formats, 73–74
    overview of, 69
    precedence constraints, 99–102
    solutions and projects, 71–73
    SSIS variables, 86–87
    system variables, 87–92
    types of objects in SSIS packages, 70–71
    user-defined variables, 92–99
Workgroup Edition, SQL Server 2008 editions, 16
WQL (WMI Query Language), 196
Writer Role, exercise using roles for granular access control, 287
WSDL (Web Service Description Language), 161

X

XML Configuration File
    indirect configurations, 578
    types of package configurations, 570–571
XML data, SOA for accessing, 8
XML documents, operations on, 161
XML files
    data sources, 351
    log provider for, 294
XML task
    configuration areas of, 162–165
    as data preparation task, 137
    overview of, 161–162
XPATH, 164
XSD (XML Schema Definition), 163
XSLT, 164

Z

zipped files
    exercise downloading from FTP server, 144–148
    exercise expanding, 149–154