

Transformation Guide

Informatica PowerCenter®
(Version 8.1.1)

Informatica PowerCenter Transformation Guide

Version 8.1.1

September 2006

Copyright (c) 1998–2006 Informatica Corporation.
All rights reserved. Printed in the USA.

This software and documentation contain proprietary information of Informatica Corporation and are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright law. Reverse engineering of the software is prohibited. No part of this document may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording or otherwise) without prior consent of Informatica Corporation.

Use, duplication, or disclosure of the Software by the U.S. Government is subject to the restrictions set forth in the applicable software license agreement and as provided in DFARS 227.7202-1(a) and 227.7702-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable.

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Informatica Corporation does not warrant that this documentation is error free.

Informatica, PowerCenter, PowerCenterRT, PowerCenter Connect, PowerCenter Data Analyzer, PowerMart, SuperGlue, Metadata Manager, Informatica Data Quality and Informatica Data Explorer are trademarks or registered trademarks of Informatica Corporation in the United States and in jurisdictions throughout the world. All other company and product names may be trade names or trademarks of their respective owners.

Portions of this software and/or documentation are subject to copyright held by third parties, including without limitation: Copyright DataDirect Technologies, 1999-2002. All rights reserved. Copyright © Sun Microsystems. All Rights Reserved. Copyright © RSA Security Inc. All Rights Reserved. Copyright © Ordinal Technology Corp. All Rights Reserved.

Informatica PowerCenter products contain ACE (TM) software copyrighted by Douglas C. Schmidt and his research group at Washington University and University of California, Irvine, Copyright (c) 1993-2002, all rights reserved.

Portions of this software contain copyrighted material from The JBoss Group, LLC. Your right to use such materials is set forth in the GNU Lesser General Public License Agreement, which may be found at <http://www.opensource.org/licenses/lgpl-license.php>. The JBoss materials are provided free of charge by Informatica, "as-is", without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Portions of this software contain copyrighted material from Meta Integration Technology, Inc. Meta Integration® is a registered trademark of Meta Integration Technology, Inc.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). The Apache Software is Copyright (c) 1999-2005 The Apache Software Foundation. All rights reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit and redistribution of this software is subject to terms available at <http://www.openssl.org>. Copyright 1998-2003 The OpenSSL Project. All Rights Reserved.

The zlib library included with this software is Copyright (c) 1995-2003 Jean-loup Gailly and Mark Adler.

The Curl license provided with this Software is Copyright 1996-2004, Daniel Stenberg, <Daniel@haxx.se>. All Rights Reserved.

The PCRE library included with this software is Copyright (c) 1997-2001 University of Cambridge Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel. The source for this library may be found at <ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre>.

InstallAnywhere is Copyright 2005 Zero G Software, Inc. All Rights Reserved.

Portions of the Software are Copyright (c) 1998-2005 The OpenLDAP Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted only as authorized by the OpenLDAP Public License, available at <http://www.openldap.org/software/release/license.html>.

This Software is protected by U.S. Patent Numbers 6,208,990; 6,044,374; 6,014,670; 6,032,158; 5,794,246; 6,339,775 and other U.S. Patents Pending.

DISCLAIMER: Informatica Corporation provides this documentation "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of non-infringement, merchantability, or use for a particular purpose. The information provided in this documentation may include technical inaccuracies or typographical errors. Informatica could make improvements and/or changes in the products described in this documentation at any time without notice.

Table of Contents

List of Figures	xxi
List of Tables	xxv
Preface	xxix
About This Book	xxx
Document Conventions	xxx
Other Informatica Resources	xxxi
Visiting Informatica Customer Portal	xxxi
Visiting the Informatica Web Site	xxxi
Visiting the Informatica Developer Network	xxxi
Visiting the Informatica Knowledge Base	xxxi
Obtaining Technical Support	xxxi
Chapter 1: Working with Transformations	1
Overview	2
Creating a Transformation	5
Configuring Transformations	6
Working with Ports	7
Creating Ports	7
Configuring Ports	7
Linking Ports	8
Multi-Group Transformations	9
Working with Expressions	10
Using the Expression Editor	12
Using Local Variables	14
Temporarily Store Data and Simplify Complex Expressions	14
Store Values Across Rows	15
Capture Values from Stored Procedures	15
Guidelines for Configuring Variable Ports	16
Using Default Values for Ports	18
Entering User-Defined Default Values	20
Entering User-Defined Default Input Values	22
Entering User-Defined Default Output Values	25

General Rules for Default Values	28
Entering and Validating Default Values	28
Configuring Tracing Level in Transformations	30
Reusable Transformations	31
Instances and Inherited Changes	31
Mapping Variables in Expressions	31
Creating Reusable Transformations	32
Promoting Non-Reusable Transformations	32
Creating Non-Reusable Instances of Reusable Transformations	33
Adding Reusable Transformations to Mappings	33
Modifying a Reusable Transformation	34
Chapter 2: Aggregator Transformation	37
Overview	38
Ports in the Aggregator Transformation	38
Components of the Aggregator Transformation	38
Aggregate Caches	39
Aggregate Expressions	40
Aggregate Functions	40
Nested Aggregate Functions	40
Conditional Clauses	41
Non-Aggregate Functions	41
Null Values in Aggregate Functions	41
Group By Ports	42
Non-Aggregate Expressions	43
Default Values	43
Using Sorted Input	45
Sorted Input Conditions	45
Pre-Sorting Data	45
Creating an Aggregator Transformation	47
Tips	50
Troubleshooting	51
Chapter 3: Custom Transformation	53
Overview	54
Working with Transformations Built On the Custom Transformation ..	54
Code Page Compatibility	55

Distributing Custom Transformation Procedures	56
Creating Custom Transformations	57
Rules and Guidelines	57
Custom Transformation Components	58
Working with Groups and Ports	59
Creating Groups and Ports	59
Editing Groups and Ports	60
Defining Port Relationships	60
Working with Port Attributes	62
Custom Transformation Properties	64
Setting the Update Strategy	66
Working with Thread-Specific Procedure Code	66
Working with Transaction Control	68
Transformation Scope	68
Generate Transaction	68
Working with Transaction Boundaries	69
Blocking Input Data	70
Writing the Procedure Code to Block Data	70
Configuring Custom Transformations as Blocking Transformations	70
Validating Mappings with Custom Transformations	71
Working with Procedure Properties	72
Creating Custom Transformation Procedures	73
Step 1. Create the Custom Transformation	73
Step 2. Generate the C Files	75
Step 3. Fill Out the Code with the Transformation Logic	76
Step 4. Build the Module	85
Step 5. Create a Mapping	87
Step 6. Run the Session in a Workflow	87
Chapter 4: Custom Transformation Functions	89
Overview	90
Working with Handles	90
Function Reference	92
Working with Rows	96
Rules and Guidelines	97
Generated Functions	98
Initialization Functions	98

Notification Functions	100
Deinitialization Functions	102
API Functions	104
Set Data Access Mode Function	104
Navigation Functions	105
Property Functions	108
Rebind Datatype Functions	115
Data Handling Functions (Row-Based Mode)	117
Set Pass-Through Port Function	120
Output Notification Function	121
Data Boundary Output Notification Function	121
Error Functions	122
Session Log Message Functions	123
Increment Error Count Function	124
Is Terminated Function	124
Blocking Functions	125
Pointer Functions	126
Change String Mode Function	126
Set Data Code Page Function	127
Row Strategy Functions (Row-Based Mode)	128
Change Default Row Strategy Function	129
Array-Based API Functions	130
Maximum Number of Rows Functions	130
Number of Rows Functions	131
Is Row Valid Function	132
Data Handling Functions (Array-Based Mode)	132
Row Strategy Functions (Array-Based Mode)	135
Set Input Error Row Functions	136
Java API Functions	138
C++ API Functions	139

Chapter 5: Expression Transformation **141**

Overview	142
Calculating Values	142
Adding Multiple Calculations	142
Creating an Expression Transformation	143

Chapter 6: External Procedure Transformation 145

Overview	146
Code Page Compatibility	146
External Procedures and External Procedure Transformations	147
External Procedure Transformation Properties	147
Pipeline Partitioning	147
COM Versus Informatica External Procedures	148
The BankSoft Example	148
Developing COM Procedures	149
Steps for Creating a COM Procedure	149
COM External Procedure Server Type	149
Using Visual C++ to Develop COM Procedures	149
Developing COM Procedures with Visual Basic	156
Developing Informatica External Procedures	159
Step 1. Create the External Procedure Transformation	159
Step 2. Generate the C++ Files	162
Step 3. Fill Out the Method Stub with Implementation	164
Step 4. Building the Module	165
Step 5. Create a Mapping	167
Step 6. Run the Session in a Workflow	167
Distributing External Procedures	169
Distributing COM Procedures	169
Distributing Informatica Modules	170
Development Notes	171
COM Datatypes	171
Row-Level Procedures	172
Return Values from Procedures	172
Exceptions in Procedure Calls	172
Memory Management for Procedures	173
Wrapper Classes for Pre-Existing C/C++ Libraries or VB Functions	173
Generating Error and Tracing Messages	173
Unconnected External Procedure Transformations	175
Initializing COM and Informatica Modules	175
Other Files Distributed and Used in TX	179
Service Process Variables in Initialization Properties	180
External Procedure Interfaces	181
Dispatch Function	181

External Procedure Function	181
Property Access Functions	182
Parameter Access Functions	183
Code Page Access Functions	185
Transformation Name Access Functions	185
Procedure Access Functions	186
Partition Related Functions	186
Tracing Level Function	187
Chapter 7: Filter Transformation	189
Overview	190
Filter Condition	192
Creating a Filter Transformation	193
Tips	195
Troubleshooting	196
Chapter 8: HTTP Transformation	197
Overview	198
Authentication	199
Connecting to the HTTP Server	199
Creating an HTTP Transformation	200
Configuring the Properties Tab	202
Configuring the HTTP Tab	204
Selecting a Method	204
Configuring Groups and Ports	205
Configuring a URL	207
Examples	209
GET Example	209
POST Example	210
SIMPLE POST Example	211
Chapter 9: Java Transformation	213
Overview	214
Steps to Define a Java Transformation	214
Active and Passive Java Transformations	215
Datatype Mapping	215
Using the Java Code Tab	217

Configuring Ports	219
Creating Groups and Ports	219
Setting Default Port Values	220
Configuring Java Transformation Properties	221
Working with Transaction Control	223
Setting the Update Strategy	224
Developing Java Code	225
Creating Java Code Snippets	225
Importing Java Packages	226
Defining Helper Code	226
On Input Row Tab	227
On End of Data Tab	228
On Receiving Transaction Tab	228
Configuring Java Transformation Settings	229
Configuring the Classpath	229
Enabling High Precision	230
Compiling a Java Transformation	231
Fixing Compilation Errors	232
Locating the Source of Compilation Errors	232
Identifying Compilation Errors	234
Chapter 10: Java Transformation API Reference	237
Java Transformation API Methods	238
commit	239
Syntax	239
Example	239
failSession	240
Syntax	240
Example	240
generateRow	241
Syntax	241
Example	241
getInRowType	242
Syntax	242
Example	242
incrementErrorCount	243
Syntax	243

Example	243
isNull	244
Syntax	244
Example	244
logInfo	245
Syntax	245
Example	245
logError	246
Syntax	246
Example	246
rollBack	247
Syntax	247
Example	247
setNull	248
Syntax	248
Example	248
setOutRowType	249
Syntax	249
Example	249
Chapter 11: Java Transformation Example	251
Overview	252
Step 1. Import the Mapping	253
Step 2. Create Transformation and Configure Ports	254
Step 3. Enter Java Code	256
Import Packages Tab	256
Helper Code Tab	257
On Input Row Tab	258
Step 4. Compile the Java Code	261
Step 5. Create a Session and Workflow	262
Sample Data	262
Chapter 12: Java Expressions	263
Overview	264
Expression Function Types	264
Using the Define Expression Dialog Box	266
Step 1. Configure the Function	266

Step 2. Create and Validate the Expression	267
Step 3. Generate Java Code for the Expression	267
Steps to Create an Expression and Generate Java Code	268
Java Expression Templates	269
Working with the Simple Interface	271
invokeJExpression	271
Simple Interface Example	272
Working with the Advanced Interface	273
Steps to Invoke an Expression with the Advanced Interface	273
Rules and Guidelines for Working with the Advanced Interface	273
EDataType Class	274
JExprParamMetadata Class	274
defineJExpression	275
JExpression Class	276
Advanced Interface Example	277
JExpression API Reference	279
invoke	279
getResultDataType	280
getResultMetadata	280
isResultNull	280
getInt	281
getDouble	281
getLong	281
getStringBuffer	282
getBytes	282
Chapter 13: Joiner Transformation	283
Overview	284
Working with the Joiner Transformation	284
Joiner Transformation Properties	286
Defining a Join Condition	288
Defining the Join Type	289
Normal Join	289
Master Outer Join	290
Detail Outer Join	290
Full Outer Join	291
Using Sorted Input	292

Configuring the Sort Order	292
Adding Transformations to the Mapping	293
Configuring the Joiner Transformation	293
Defining the Join Condition	294
Joining Data from a Single Source	296
Joining Two Branches of the Same Pipeline	296
Joining Two Instances of the Same Source	297
Guidelines	298
Blocking the Source Pipelines	299
Unsorted Joiner Transformation	299
Sorted Joiner Transformation	299
Working with Transactions	300
Preserving Transaction Boundaries for a Single Pipeline	301
Preserving Transaction Boundaries in the Detail Pipeline	301
Dropping Transaction Boundaries for Two Pipelines	302
Creating a Joiner Transformation	303
Tips	306
Chapter 14: Lookup Transformation	307
Overview	308
Connected and Unconnected Lookups	309
Connected Lookup Transformation	309
Unconnected Lookup Transformation	310
Relational and Flat File Lookups	311
Relational Lookups	311
Flat File Lookups	311
Lookup Components	313
Lookup Source	313
Lookup Ports	313
Lookup Properties	314
Lookup Condition	315
Metadata Extensions	315
Lookup Properties	316
Configuring Lookup Properties in a Session	320
Lookup Query	324
Default Lookup Query	324
Overriding the Lookup Query	324

Lookup Condition	328
Uncached or Static Cache	328
Dynamic Cache	329
Handling Multiple Matches	329
Lookup Caches	330
Configuring Unconnected Lookup Transformations	331
Step 1. Add Input Ports	331
Step 2. Add the Lookup Condition	332
Step 3. Designate a Return Value	332
Step 4. Call the Lookup Through an Expression	333
Creating a Lookup Transformation	335
Tips	336
Chapter 15: Lookup Caches	337
Overview	338
Cache Comparison	339
Building Connected Lookup Caches	340
Sequential Caches	340
Concurrent Caches	341
Using a Persistent Lookup Cache	342
Using a Non-Persistent Cache	342
Using a Persistent Cache	342
Rebuilding the Lookup Cache	342
Working with an Uncached Lookup or Static Cache	344
Working with a Dynamic Lookup Cache	345
Using the NewLookupRow Port	347
Using the Associated Input Port	348
Working with Lookup Transformation Values	349
Using the Ignore Null Property	353
Using the Ignore in Comparison Property	354
Using Update Strategy Transformations with a Dynamic Cache	354
Updating the Dynamic Lookup Cache	356
Using the WHERE Clause with a Dynamic Cache	358
Synchronizing the Dynamic Lookup Cache	359
Example Using a Dynamic Lookup Cache	360
Rules and Guidelines for Dynamic Caches	361
Sharing the Lookup Cache	363

Sharing an Unnamed Lookup Cache	363
Sharing a Named Lookup Cache	365
Lookup Cache Tips	369
Chapter 16: Normalizer Transformation	371
Overview	372
Normalizing Data in a Mapping	373
Normalizer Ports	373
Adding a COBOL Source to a Mapping	374
Differences Between Normalizer Transformations	378
Troubleshooting	379
Chapter 17: Rank Transformation	381
Overview	382
Ranking String Values	383
Rank Caches	383
Rank Transformation Properties	383
Ports in a Rank Transformation	384
Rank Index	384
Defining Groups	385
Creating a Rank Transformation	386
Chapter 18: Router Transformation	389
Overview	390
Working with Groups	392
Input Group	392
Output Groups	392
Using Group Filter Conditions	393
Adding Groups	394
Working with Ports	396
Connecting Router Transformations in a Mapping	398
Creating a Router Transformation	400
Chapter 19: Sequence Generator Transformation	401
Overview	402
Common Uses	403
Creating Keys	403

Replacing Missing Values	403
Sequence Generator Ports	404
NEXTVAL	404
CURRVAL	406
Transformation Properties	407
Start Value and Cycle	408
Increment By	408
End Value	409
Current Value	409
Number of Cached Values	410
Reset	411
Creating a Sequence Generator Transformation	412
Chapter 20: Sorter Transformation	415
Overview	416
Sorting Data	417
Sorter Transformation Properties	419
Sorter Cache Size	419
Case Sensitive	421
Work Directory	421
Distinct Output Rows	421
Tracing Level	421
Null Treated Low	422
Transformation Scope	422
Creating a Sorter Transformation	423
Chapter 21: Source Qualifier Transformation	425
Overview	426
Transformation Datatypes	426
Target Load Order	426
Parameters and Variables	427
Source Qualifier Transformation Properties	429
Default Query	431
Viewing the Default Query	432
Overriding the Default Query	432
Joining Source Data	434
Default Join	434

Custom Joins	435
Heterogeneous Joins	436
Creating Key Relationships	436
Adding an SQL Query	438
Entering a User-Defined Join	440
Outer Join Support	442
Informatica Join Syntax	442
Creating an Outer Join	447
Common Database Syntax Restrictions	449
Entering a Source Filter	450
Using Sorted Ports	452
Select Distinct	454
Overriding Select Distinct in the Session	454
Adding Pre- and Post-Session SQL Commands	455
Creating a Source Qualifier Transformation	456
Creating a Source Qualifier Transformation By Default	456
Creating a Source Qualifier Transformation Manually	456
Configuring Source Qualifier Transformation Options	456
Troubleshooting	458
Chapter 22: SQL Transformation	459
Overview	460
Script Mode	461
Example	462
Script Mode Rules and Guidelines	462
Query Mode	464
Using Static SQL Queries	465
Using Dynamic SQL Queries	466
Query Mode Rules and Guidelines	469
Connecting to Databases	470
Using a Static Database Connection	470
Passing a Logical Database Connection	470
Passing Full Connection Information	470
Database Connections Rules and Guidelines	473
Session Processing	474
Input Row to Output Row Cardinality	474
Transaction Control	478

High Availability	478
Creating an SQL Transformation	480
SQL Transformation Properties	482
Properties Tab	482
SQL Settings Tab	485
SQL Ports Tab	486
SQL Statements	488
Chapter 23: Stored Procedure Transformation	489
Overview	490
Input and Output Data	490
Connected and Unconnected	491
Specifying when the Stored Procedure Runs	492
Using a Stored Procedure in a Mapping	494
Writing a Stored Procedure	495
Sample Stored Procedure	495
Creating a Stored Procedure Transformation	498
Importing Stored Procedures	498
Manually Creating Stored Procedure Transformations	500
Setting Options for the Stored Procedure	501
Using \$Source and \$Target Variables	502
Changing the Stored Procedure	503
Configuring a Connected Transformation	504
Configuring an Unconnected Transformation	506
Calling a Stored Procedure From an Expression	506
Calling a Pre- or Post-Session Stored Procedure	509
Error Handling	512
Pre-Session Errors	512
Post-Session Errors	513
Session Errors	513
Supported Databases	514
SQL Declaration	514
Parameter Types	514
Input/Output Port in Mapping	514
Type of Return Value Supported	515
Expression Rules	516
Tips	517

Troubleshooting	518
Chapter 24: Transaction Control Transformation	519
Overview	520
Transaction Control Transformation Properties	521
Properties Tab	521
Example	522
Using Transaction Control Transformations in Mappings	524
Sample Transaction Control Mappings with Multiple Targets	525
Mapping Guidelines and Validation	528
Creating a Transaction Control Transformation	529
Chapter 25: Union Transformation	531
Overview	532
Union Transformation Rules and Guidelines	532
Union Transformation Components	532
Working with Groups and Ports	534
Creating a Union Transformation	536
Using a Union Transformation in Mappings	538
Chapter 26: Update Strategy Transformation	539
Overview	540
Setting the Update Strategy	540
Flagging Rows Within a Mapping	541
Forwarding Rejected Rows	541
Update Strategy Expressions	541
Aggregator and Update Strategy Transformations	542
Lookup and Update Strategy Transformations	543
Setting the Update Strategy for a Session	544
Specifying an Operation for All Rows	544
Specifying Operations for Individual Target Tables	545
Update Strategy Checklist	547
Chapter 27: XML Transformations	549
XML Source Qualifier Transformation	550
XML Parser Transformation	551
XML Generator Transformation	552

Index	553
--------------	------------

List of Figures

Figure 1-1. Sample Ports Tab	7
Figure 1-2. Example of Input, Output, and Input/Output Ports	8
Figure 1-3. Sample Input and Output Ports	10
Figure 1-4. Expression Editor	12
Figure 1-5. Variable Ports Store Values Across Rows	15
Figure 1-6. Default Value for Input and Input/Output Ports	19
Figure 1-7. Default Value for Output Ports	19
Figure 1-8. Using a Constant as a Default Value	23
Figure 1-9. Using the ERROR Function to Skip Null Input Values	24
Figure 1-10. Entering and Validating Default Values	29
Figure 1-11. Reverting to Original Reusable Transformation Properties	35
Figure 2-1. Sample Mapping with Aggregator and Sorter Transformations	46
Figure 3-1. Custom Transformation Ports Tab	59
Figure 3-2. Editing Port Dependencies	61
Figure 3-3. Port Attribute Definitions Tab	62
Figure 3-4. Edit Port Attribute Values	63
Figure 3-5. Custom Transformation Properties	64
Figure 3-6. Custom Transformation Ports Tab - Union Example	74
Figure 3-7. Custom Transformation Properties Tab - Union Example	75
Figure 3-8. Mapping with a Custom Transformation - Union Example	87
Figure 4-1. Custom Transformation Handles	91
Figure 6-1. Process for Distributing External Procedures	169
Figure 6-2. External Procedure Transformation Initialization Properties	178
Figure 6-3. External Procedure Transformation Initialization Properties Tab	180
Figure 7-1. Sample Mapping with a Filter Transformation	190
Figure 7-2. Specifying a Filter Condition in a Filter Transformation	191
Figure 8-1. HTTP Transformation Processing	198
Figure 8-2. HTTP Transformation	201
Figure 8-3. HTTP Transformation Properties Tab	202
Figure 8-4. HTTP Transformation HTTP Tab	204
Figure 8-5. HTTP Tab for a GET Example	209
Figure 8-6. HTTP Tab for a POST Example	210
Figure 8-7. HTTP Tab for a SIMPLE POST Example	211
Figure 9-1. Java Code Tab Components	217
Figure 9-2. Java Transformation Ports Tab	219
Figure 9-3. Java Transformation Properties	221
Figure 9-4. Java Transformation Settings Dialog Box	229
Figure 9-5. Highlighted Error in Code Entry Tab	233
Figure 9-6. Highlighted Error in Full Code Window	234
Figure 11-1. Java Transformation Example - Sample Mapping	253

Figure 11-2. Java Transformation Example - Ports Tab	255
Figure 11-3. Java Transformation Example - Import Packages Tab	257
Figure 11-4. Java Transformation Example - Helper Code Tab	258
Figure 11-5. Java Transformation Example - On Input Row Tab	260
Figure 11-6. Java Transformation Example - Successful Compilation	261
Figure 12-1. Define Expression Dialog Box	267
Figure 12-2. Java Expressions Code Entry Tab	268
Figure 13-1. Mapping with Master and Detail Pipelines	284
Figure 13-2. Joiner Transformation Properties Tab	286
Figure 13-3. Mapping Configured to Join Data from Two Pipelines	295
Figure 13-4. Mapping that Joins Two Branches of a Pipeline	297
Figure 13-5. Mapping that Joins Two Instances of the Same Source	297
Figure 13-6. Preserving Transaction Boundaries when You Join Two Pipeline Branches	301
Figure 14-1. Session Properties for Flat File Lookups	321
Figure 14-2. Return Port in a Lookup Transformation	333
Figure 15-1. Building Lookup Caches Sequentially	340
Figure 15-2. Building Lookup Caches Concurrently	341
Figure 15-3. Mapping with a Dynamic Lookup Cache	346
Figure 15-4. Dynamic Lookup Transformation Ports Tab	347
Figure 15-5. Using Update Strategy Transformations with a Lookup Transformation	355
Figure 15-6. Slowly Changing Dimension Mapping with Dynamic Lookup Cache	360
Figure 16-1. COBOL Source Definition and a Normalizer Transformation	375
Figure 17-1. Sample Mapping with a Rank Transformation	382
Figure 18-1. Comparing Router and Filter Transformations	390
Figure 18-2. Sample Router Transformation	391
Figure 18-3. Using a Router Transformation in a Mapping	393
Figure 18-4. Specifying Group Filter Conditions	394
Figure 18-5. Router Transformation Ports Tab	396
Figure 18-6. Input Port Name and Corresponding Output Port Names	397
Figure 19-1. Connecting NEXTVAL to Two Target Tables in a Mapping	404
Figure 19-2. Mapping with a Sequence Generator and an Expression Transformation	405
Figure 19-3. Connecting CURRVAL and NEXTVAL Ports to a Target	406
Figure 20-1. Sample Mapping with a Sorter Transformation	416
Figure 20-2. Sample Sorter Transformation Ports Configuration	417
Figure 20-3. Sorter Transformation Properties	419
Figure 21-1. Source Definition Connected to a Source Qualifier Transformation	431
Figure 21-2. Joining Two Tables with One Source Qualifier Transformation	435
Figure 21-3. Creating a Relationship Between Two Tables	437
Figure 22-1. SQL Transformation Script Mode Ports	461
Figure 22-2. SQL Editor for an SQL Transformation Query	464
Figure 22-3. SQL Transformation Static Query Mode Ports	466
Figure 22-4. SQL Transformation Ports to Pass a Full Dynamic Query	467
Figure 22-5. SQL Transformation Properties Tab	483

Figure 22-6. SQL Settings Tab	485
Figure 22-7. SQL Transformation SQL Ports Tab	486
Figure 23-1. Sample Mapping with a Stored Procedure Transformation	504
Figure 23-2. Expression Transformation Referencing a Stored Procedure Transformation ..	506
Figure 23-3. Stored Procedure Error Handling	512
Figure 24-1. Transaction Control Transformation Properties	521
Figure 24-2. Sample Transaction Control Mapping	523
Figure 24-3. Effective and Ineffective Transaction Control Transformations	525
Figure 24-4. Transaction Control Transformation Effective for a Transformation	525
Figure 24-5. Valid Mapping with Transaction Control Transformations	526
Figure 24-6. Invalid Mapping with Transaction Control Transformations	527
Figure 25-1. Union Transformation Groups Tab	534
Figure 25-2. Union Transformation Group Ports Tab	535
Figure 25-3. Union Transformation Ports Tab	535
Figure 25-4. Mapping with a Union Transformation	538
Figure 26-1. Specifying Operations for Individual Target Tables	546

List of Tables

Table 1-1. Transformation Descriptions	2
Table 1-2. Multi-Group Transformations	9
Table 1-3. Transformations Containing Expressions.....	11
Table 1-4. Variable Usage.....	14
Table 1-5. System Default Values and Integration Service Behavior.....	18
Table 1-6. Transformations Supporting User-Defined Default Values	20
Table 1-7. Default Values for Input and Input/Output Ports	22
Table 1-8. Supported Default Values for Output Ports.....	26
Table 1-9. Session Log Tracing Levels	30
Table 3-1. Custom Transformation Properties	64
Table 3-2. Transaction Boundary Handling with Custom Transformations	69
Table 3-3. Module File Names	85
Table 3-4. UNIX Commands to Build the Shared Library.....	86
Table 4-1. Custom Transformation Handles	91
Table 4-2. Custom Transformation Generated Functions	92
Table 4-3. Custom Transformation API Functions.....	92
Table 4-4. Custom Transformation Array-Based API Functions	94
Table 4-5. INFA_CT_MODULE Property IDs	109
Table 4-6. INFA_CT_PROC_HANDLE Property IDs.....	110
Table 4-7. INFA_CT_TRANS_HANDLE Property IDs.....	111
Table 4-8. INFA_CT_INPUT_GROUP and INFA_CT_OUTPUT_GROUP Handle Property IDs	112
Table 4-9. INFA_CT_INPUTPORT and INFA_CT_OUTPUTPORT_HANDLE Handle Property IDs	113
Table 4-10. Property Functions (MBCS)	115
Table 4-11. Property Functions (Unicode).....	115
Table 4-12. Compatible Datatypes	116
Table 4-13. Get Data Functions	118
Table 4-14. Get Data Functions (Array-Based Mode).....	133
Table 6-1. Differences Between COM and Informatica External Procedures	148
Table 6-2. Visual C++ and Transformation Datatypes	171
Table 6-3. Visual Basic and Transformation Datatypes.....	171
Table 6-4. External Procedure Initialization Properties.....	180
Table 6-5. Descriptions of Parameter Access Functions.....	183
Table 6-6. Member Variable of the External Procedure Base Class.....	185
Table 8-1. HTTP Transformation Properties	202
Table 8-2. HTTP Transformation Methods	205
Table 8-3. GET Method Groups and Ports	206
Table 8-4. POST Method Groups and Ports	206
Table 8-5. SIMPLE POST Method Groups and Ports	207
Table 9-1. Mapping from PowerCenter Datatypes to Java Datatypes	215

Table 9-2. Java Transformation Properties	221
Table 11-1. Input and Output Ports	254
Table 12-1. Enumerated Java Datatypes	274
Table 12-2. JExpression API Methods	276
Table 13-1. Joiner Transformation Properties	286
Table 13-2. Integration Service Behavior with Transformation Scopes for the Joiner Transformation	300
Table 14-1. Differences Between Connected and Unconnected Lookups	309
Table 14-2. Lookup Transformation Port Types	314
Table 14-3. Lookup Transformation Properties	316
Table 14-4. Session Properties for Flat File Lookups	322
Table 15-1. Lookup Caching Comparison	339
Table 15-2. Integration Service Handling of Persistent Caches	343
Table 15-3. NewLookupRow Values	348
Table 15-4. Dynamic Lookup Cache Behavior for Insert Row Type	357
Table 15-5. Dynamic Lookup Cache Behavior for Update Row Type	358
Table 15-6. Location for Sharing Unnamed Cache	364
Table 15-7. Properties for Sharing Unnamed Cache	364
Table 15-8. Location for Sharing Named Cache	367
Table 15-9. Properties for Sharing Named Cache	367
Table 16-1. VSAM and Relational Normalizer Transformation Differences	378
Table 17-1. Rank Transformation Ports	384
Table 17-2. Rank Transformation Properties	387
Table 19-1. Sequence Generator Transformation Properties	407
Table 20-1. Column Sizes for Sorter Data Calculations	420
Table 21-1. Conversion for Datetime Mapping Parameters and Variables	427
Table 21-2. Source Qualifier Transformation Properties	429
Table 21-3. Locations for Entering Outer Join Syntax	443
Table 21-4. Syntax for Normal Joins in a Join Override	443
Table 21-5. Syntax for Left Outer Joins in a Join Override	445
Table 21-6. Syntax for Right Outer Joins in a Join Override	447
Table 22-1. Full Database Connection Information	471
Table 22-2. Native Connect String Syntax	471
Table 22-3. Output Rows By Query Statement - Query Mode	475
Table 22-4. NumRowsAffected Rows by Query Statement - Query Mode	475
Table 22-5. Output Rows by Query Statement - Query Mode	477
Table 22-6. SQL Transformation Connection Options	481
Table 22-7. SQL Transformation Properties	483
Table 22-8. SQL Settings Tab Attributes	485
Table 22-9. SQL Transformation Ports	486
Table 22-10. Standard SQL Statements	488
Table 23-1. Comparison of Connected and Unconnected Stored Procedure Transformations	492
Table 23-2. Setting Options for the Stored Procedure Transformation	501
Table 26-1. Constants for Each Database Operation	541

Table 26-2. Specifying an Operation for All Rows	544
Table 26-3. Update Strategy Settings.....	545

Preface

Welcome to PowerCenter, the Informatica software product that delivers an open, scalable data integration solution addressing the complete life cycle for all data integration projects including data warehouses, data migration, data synchronization, and information hubs. PowerCenter combines the latest technology enhancements for reliably managing data repositories and delivering information resources in a timely, usable, and efficient manner.

The PowerCenter repository coordinates and drives a variety of core functions, including extracting, transforming, loading, and managing data. The Integration Service can extract large volumes of data from multiple platforms, handle complex transformations on the data, and support high-speed loads. PowerCenter can simplify and accelerate the process of building a comprehensive data warehouse from disparate data sources.

About This Book

The *Transformation Guide* is written for the developers and software engineers responsible for implementing your data warehouse. The *Transformation Guide* assumes that you have a solid understanding of your operating systems, relational database concepts, and the database engines, flat files, or mainframe system in your environment. This guide also assumes that you are familiar with the interface requirements for your supporting applications.

The material in this book is also available online.

Document Conventions

This guide uses the following formatting conventions:

If you see...	It means...
<i>italicized text</i>	The word or set of words are especially emphasized.
boldfaced text	Emphasized subjects.
<i>italicized monospaced text</i>	This is the variable name for a value you enter as part of an operating system command. This is generic text that should be replaced with user-supplied values.
Note:	The following paragraph provides additional facts.
Tip:	The following paragraph provides suggested uses.
Warning:	The following paragraph notes situations where you can overwrite or corrupt data, unless you follow the specified procedure.
monospaced text	This is a code example.
bold monospaced text	This is an operating system command you enter from a prompt to run a task.

Other Informatica Resources

In addition to the product manuals, Informatica provides these other resources:

- ◆ Informatica Customer Portal
- ◆ Informatica web site
- ◆ Informatica Developer Network
- ◆ Informatica Knowledge Base
- ◆ Informatica Technical Support

Visiting Informatica Customer Portal

As an Informatica customer, you can access the Informatica Customer Portal site at <http://my.informatica.com>. The site contains product information, user group information, newsletters, access to the Informatica customer support case management system (ATLAS), the Informatica Knowledge Base, Informatica Documentation Center, and access to the Informatica user community.

Visiting the Informatica Web Site

You can access the Informatica corporate web site at <http://www.informatica.com>. The site contains information about Informatica, its background, upcoming events, and sales offices. You will also find product and partner information. The services area of the site includes important information about technical support, training and education, and implementation services.

Visiting the Informatica Developer Network

You can access the Informatica Developer Network at <http://devnet.informatica.com>. The Informatica Developer Network is a web-based forum for third-party software developers. The site contains information about how to create, market, and support customer-oriented add-on solutions based on interoperability interfaces for Informatica products.

Visiting the Informatica Knowledge Base

As an Informatica customer, you can access the Informatica Knowledge Base at <http://my.informatica.com>. Use the Knowledge Base to search for documented solutions to known technical issues about Informatica products. You can also find answers to frequently asked questions, technical white papers, and technical tips.

Obtaining Technical Support

There are many ways to access Informatica Technical Support. You can contact a Technical Support Center by using the telephone numbers listed the following table, you can send email, or you can use the WebSupport Service.

Use the following email addresses to contact Informatica Technical Support:

- ◆ support@informatica.com for technical inquiries
- ◆ support_admin@informatica.com for general customer service requests

WebSupport requires a user name and password. You can request a user name and password at <http://my.informatica.com>.

North America / South America	Europe / Middle East / Africa	Asia / Australia
Informatica Corporation Headquarters 100 Cardinal Way Redwood City, California 94063 United States	Informatica Software Ltd. 6 Waltham Park Waltham Road, White Waltham Maidenhead, Berkshire SL6 3TN United Kingdom	Informatica Business Solutions Pvt. Ltd. Diamond District Tower B, 3rd Floor 150 Airport Road Bangalore 560 008 India
Toll Free 877 463 2435	Toll Free 00 800 4632 4357	Toll Free Australia: 00 11 800 4632 4357 Singapore: 001 800 4632 4357
Standard Rate United States: 650 385 5800	Standard Rate Belgium: +32 15 281 702 France: +33 1 41 38 92 26 Germany: +49 1805 702 702 Netherlands: +31 306 022 797 United Kingdom: +44 1628 511 445	Standard Rate India: +91 80 4112 5738

Chapter 1

Working with Transformations

This chapter includes the following topics:

- ◆ Overview, 2
- ◆ Creating a Transformation, 5
- ◆ Configuring Transformations, 6
- ◆ Working with Ports, 7
- ◆ Multi-Group Transformations, 9
- ◆ Working with Expressions, 10
- ◆ Using Local Variables, 14
- ◆ Using Default Values for Ports, 18
- ◆ Configuring Tracing Level in Transformations, 30
- ◆ Reusable Transformations, 31

Overview

A transformation is a repository object that generates, modifies, or passes data. The Designer provides a set of transformations that perform specific functions. For example, an Aggregator transformation performs calculations on groups of data.

Transformations in a mapping represent the operations the Integration Service performs on the data. Data passes through transformation ports that you link in a mapping or mapplet.

Transformations can be active or passive. An active transformation can change the number of rows that pass through it, such as a Filter transformation that removes rows that do not meet the filter condition. A passive transformation does not change the number of rows that pass through it, such as an Expression transformation that performs a calculation on data and passes all rows through the transformation.

Transformations can be connected to the data flow, or they can be unconnected. An unconnected transformation is not connected to other transformations in the mapping. An unconnected transformation is called within another transformation, and returns a value to that transformation.

Table 1-1 provides a brief description of each transformation:

Table 1-1. Transformation Descriptions

Transformation	Type	Description
Aggregator	Active/ Connected	Performs aggregate calculations.
Application Source Qualifier	Active/ Connected	Represents the rows that the Integration Service reads from an application, such as an ERP source, when it runs a session.
Custom	Active or Passive/ Connected	Calls a procedure in a shared library or DLL.
Expression	Passive/ Connected	Calculates a value.
External Procedure	Passive/ Connected or Unconnected	Calls a procedure in a shared library or in the COM layer of Windows.
Filter	Active/ Connected	Filters data.
HTTP Transformation	Passive/ Connected	Connects to an HTTP server to read or update data.
Input	Passive/ Connected	Defines mapplet input rows. Available in the Mapplet Designer.
Java	Active or Passive/ Connected	Executes user logic coded in Java. The byte code for the user logic is stored in the repository.

Table 1-1. Transformation Descriptions

Transformation	Type	Description
Joiner	Active/ Connected	Joins data from different databases or flat file systems.
Lookup	Passive/ Connected or Unconnected	Looks up values.
Normalizer	Active/ Connected	Source qualifier for COBOL sources. Can also use in the pipeline to normalize data from relational or flat file sources.
Output	Passive/ Connected	Defines mapplet output rows. Available in the Mapplet Designer.
Rank	Active/ Connected	Limits records to a top or bottom range.
Router	Active/ Connected	Routes data into multiple transformations based on group conditions.
Sequence Generator	Passive/ Connected	Generates primary keys.
Sorter	Active/ Connected	Sorts data based on a sort key.
Source Qualifier	Active/ Connected	Represents the rows that the Integration Service reads from a relational or flat file source when it runs a session.
SQL	Active or Passive/ Connected	Executes SQL queries against a database.
Stored Procedure	Passive/ Connected or Unconnected	Calls a stored procedure.
Transaction Control	Active/ Connected	Defines commit and rollback transactions.
Union	Active/ Connected	Merges data from different databases or flat file systems.
Update Strategy	Active/ Connected	Determines whether to insert, delete, update, or reject rows.
XML Generator	Active/ Connected	Reads data from one or more input ports and outputs XML through a single output port.
XML Parser	Active/ Connected	Reads XML from one input port and outputs data to one or more output ports.
XML Source Qualifier	Active/ Connected	Represents the rows that the Integration Service reads from an XML source when it runs a session.

When you build a mapping, you add transformations and configure them to handle data according to a business purpose. Complete the following tasks to incorporate a transformation into a mapping:

1. **Create the transformation.** Create it in the Mapping Designer as part of a mapping, in the Mapplet Designer as part of a mapplet, or in the Transformation Developer as a reusable transformation.
2. **Configure the transformation.** Each type of transformation has a unique set of options that you can configure.
3. **Link the transformation to other transformations and target definitions.** Drag one port to another to link them in the mapping or mapplet.

Creating a Transformation

You can create transformations using the following Designer tools:

- ◆ **Mapping Designer.** Create transformations that connect sources to targets. Transformations in a mapping cannot be used in other mappings unless you configure them to be reusable.
- ◆ **Transformation Developer.** Create individual transformations, called reusable transformations, that use in multiple mappings. For more information, see “Reusable Transformations” on page 31.
- ◆ **Mapplet Designer.** Create and configure a set of transformations, called mapplets, that you use in multiple mappings. For more information, see “Mapplets” in the *Designer Guide*.

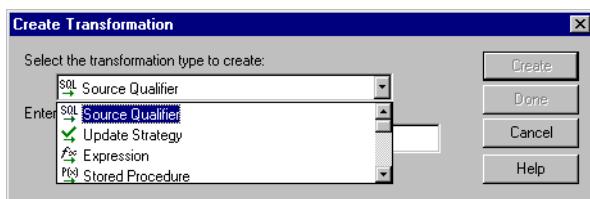
Use the same process to create a transformation in the Mapping Designer, Transformation Developer, and Mapplet Designer.

To create a transformation:

1. Open the appropriate Designer tool.
2. In the Mapping Designer, open or create a Mapping. In the Mapplet Designer, open or create a Mapplet.
3. On the Transformations toolbar, click the button corresponding to the transformation you want to create.

-or-

Click Transformation > Create and select the type of transformation you want to create.



4. Drag across the portion of the mapping where you want to place the transformation.

The new transformation appears in the workspace. Next, you need to configure the transformation by adding any new ports to it and setting other properties.

Configuring Transformations

After you create a transformation, you can configure it. Every transformation contains the following common tabs:

- ◆ **Transformation.** Name the transformation or add a description.
- ◆ **Port.** Add and configure ports.
- ◆ **Properties.** Configure properties that are unique to the transformation.
- ◆ **Metadata Extensions.** Extend the metadata in the repository by associating information with individual objects in the repository.

Some transformations might include other tabs, such as the Condition tab, where you enter conditions in a Joiner or Normalizer transformation.

When you configure transformations, you might complete the following tasks:

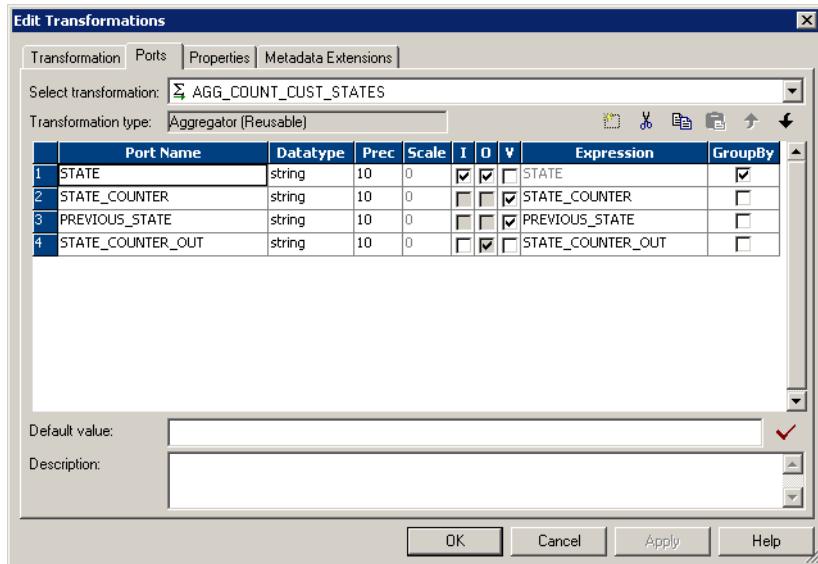
- ◆ **Add ports.** Define the columns of data that move into and out of the transformation.
- ◆ **Add groups.** In some transformations, define input or output groups that define a row of data entering or leaving the transformation.
- ◆ **Enter expressions.** Enter SQL-like expressions in some transformations that transform the data.
- ◆ **Define local variables.** Define local variables in some transformations that temporarily store data.
- ◆ **Override default values.** Configure default values for ports to handle input nulls and output transformation errors.
- ◆ **Enter tracing levels.** Choose the amount of detail the Integration Service writes in the session log about a transformation.

Working with Ports

After you create a transformation, you need to add and configure ports using the Ports tab.

Figure 1-1 shows a sample Ports tab:

Figure 1-1. Sample Ports Tab



Creating Ports

You can create a new port in the following ways:

- ♦ **Drag a port from another transformation.** When you drag a port from another transformation the Designer creates a port with the same properties, and it links the two ports. Click Layout > Copy Columns to enable copying ports.
- ♦ **Click the Add button on the Ports tab.** The Designer creates an empty port you can configure.

Configuring Ports

On the Ports tab, you can configure the following properties:

- ♦ **Port name.** The name of the port.
- ♦ **Datatype, precision, and scale.** If you plan to enter an expression or condition, make sure the datatype matches the return value of the expression.
- ♦ **Port type.** Transformations may contain a combination of input, output, input/output, and variable port types.

- ◆ **Default value.** The Designer assigns default values to handle null values and output transformation errors. You can override the default value in some ports.
- ◆ **Description.** A description of the port.
- ◆ **Other properties.** Some transformations have properties specific to that transformation, such as expressions or group by properties.

For more information about configuration options, see the appropriate sections in this chapter or in the specific transformation chapters.

Note: The Designer creates some transformations with configured ports. For example, the Designer creates a Lookup transformation with an output port for each column in the table or view used for the lookup. You need to create a port representing a value used to perform a lookup.

Linking Ports

Once you add and configure a transformation in a mapping, you link it to targets and other transformations. You link mapping objects through the ports. Data passes into and out of a mapping through the following ports:

- ◆ **Input ports.** Receive data.
- ◆ **Output ports.** Pass data.
- ◆ **Input/output ports.** Receive data and pass it unchanged.

Figure 1-2 shows an example of a transformation with input, output, and input/output ports:

Figure 1-2. Example of Input, Output, and Input/Output Ports

Name	Expression
PRICE	
MANUFACTURER	MANUFAC
OUT_MIN_PRICE	MIN(PRICE)
OUT_MAX_PRICE	MAX(PRICE)
OUT_AVG_PRICE	AVG(PRICE)

To link ports, drag between ports in different mapping objects. The Designer validates the link and creates the link only when the link meets validation requirements.

For more information about connecting mapping objects or about how to link ports, see “Mappings” in the *Designer Guide*.

Multi-Group Transformations

Transformations have input and output groups. A group is a set of ports that define a row of incoming or outgoing data. A group is analogous to a table in a relational source or target definition. Most transformations have one input and one output group. However, some have multiple input groups, multiple output groups, or both. A group is the representation of a row of data entering or leaving a transformation.

Table 1-2 lists the transformations with multiple groups:

Table 1-2. Multi-Group Transformations

Transformation	Description
Custom	Contains any number of input and output groups.
HTTP	Contains an input, output, and a header group.
Joiner	Contains two input groups, the master source and detail source, and one output group.
Router	Contains one input group and multiple output groups.
Union	Contains multiple input groups and one output group.
XML Source Qualifier	Contains multiple input and output groups.
XML Target Definition	Contains multiple input groups.
XML Parser	Contains one input group and multiple output groups.
XML Generator	Contains multiple input groups and one output group.

When you connect transformations in a mapping, you must consider input and output groups. For more information about connecting transformations in a mapping, see “Mappings” in the *Designer Guide*.

Some multiple input group transformations require the Integration Service to block data at an input group while the Integration Service waits for a row from a different input group. A blocking transformation is a multiple input group transformation that blocks incoming data. The following transformations are blocking transformations:

- ◆ Custom transformation with the Inputs May Block property enabled
- ◆ Joiner transformation configured for unsorted input

The Designer performs data flow validation when you save or validate a mapping. Some mappings that contain blocking transformations might not be valid. For more information about data flow validation, see “Mappings” in the *Designer Guide*.

For more information about blocking source data, see “Integration Service Architecture” in the *Administrator Guide*.

Working with Expressions

You can enter expressions using the Expression Editor in some transformations. Create expressions with the following functions:

- ◆ **Transformation language functions.** SQL-like functions designed to handle common expressions.
- ◆ **User-defined functions.** Functions you create in PowerCenter based on transformation language functions.
- ◆ **Custom functions.** Functions you create with the Custom Function API.

For more information about the transformation language and custom functions, see the *Transformation Language Reference*. For more information about user-defined functions, see “Working with User-Defined Functions” in the *Designer Guide*.

Enter an expression in an output port that uses the value of data from an input or input/output port. For example, you have a transformation with an input port IN_SALARY that contains the salaries of all the employees. You might want to use the individual values from the IN_SALARY column later in the mapping, and the total and average salaries you calculate through this transformation. For this reason, the Designer requires you to create a separate output port for each calculated value.

Figure 1-3 shows an Aggregator transformation that uses input ports to calculate sums and averages:

Figure 1-3. Sample Input and Output Ports

Name	Expression
PRICE	
MANUFACTURER_ID	MANUFACTURER_ID
OUT_MIN_PRICE	MIN(PRICE)
OUT_MAX_PRICE	MAX(PRICE)
OUT_AVG_PRICE	AVG(PRICE)

Table 1-3 lists the transformations in which you can enter expressions:

Table 1-3. Transformations Containing Expressions

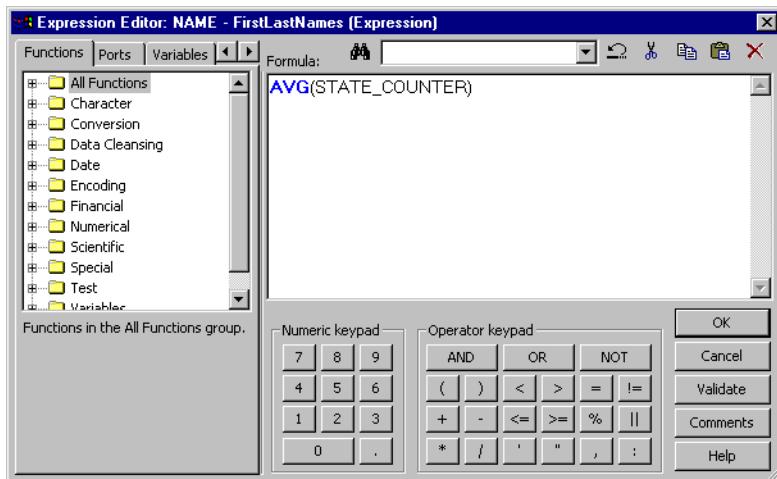
Transformation	Expression	Return Value
Aggregator	Performs an aggregate calculation based on all data passed through the transformation. Alternatively, you can specify a filter for records in the aggregate calculation to exclude certain kinds of records. For example, you can find the total number and average salary of all employees in a branch office using this transformation.	Result of an aggregate calculation for a port.
Expression	Performs a calculation based on values within a single row. For example, based on the price and quantity of a particular item, you can calculate the total purchase price for that line item in an order.	Result of a row-level calculation for a port.
Filter	Specifies a condition used to filter rows passed through this transformation. For example, if you want to write customer data to the BAD_DEBT table for customers with outstanding balances, you could use the Filter transformation to filter customer data.	TRUE or FALSE, depending on whether a row meets the specified condition. Only rows that return TRUE are passed through this transformation. The transformation applies this value to each row passed through it.
Rank	Sets the conditions for rows included in a rank. For example, you can rank the top 10 salespeople who are employed with the company.	Result of a condition or calculation for a port.
Router	Routes data into multiple transformations based on a group expression. For example, use this transformation to compare the salaries of employees at three different pay levels. You can do this by creating three groups in the Router transformation. For example, create one group expression for each salary range.	TRUE or FALSE, depending on whether a row meets the specified group expression. Only rows that return TRUE pass through each user-defined group in this transformation. Rows that return FALSE pass through the default group.
Update Strategy	Flags a row for update, insert, delete, or reject. You use this transformation when you want to control updates to a target, based on some condition you apply. For example, you might use the Update Strategy transformation to flag all customer rows for update when the mailing address has changed, or flag all employee rows for reject for people who no longer work for the company.	Numeric code for update, insert, delete, or reject. The transformation applies this value to each row passed through it.
Transaction Control	Specifies a condition used to determine the action the Integration Service performs, either commit, roll back, or no transaction change. You use this transformation when you want to control commit and rollback transactions based on a row or set of rows that pass through the transformation. For example, use this transformation to commit a set of rows based on an order entry date.	One of the following built-in variables, depending on whether or not a row meets the specified condition: - TC_CONTINUE_TRANSACTION - TC_COMMIT_BEFORE - TC_COMMIT_AFTER - TC_ROLLBACK_BEFORE - TC_ROLLBACK_AFTER The Integration Service performs actions based on the return value.

Using the Expression Editor

Use the Expression Editor to build SQL-like statements. Although you can enter an expression manually, you should use the point-and-click method. Select functions, ports, variables, and operators from the point-and-click interface to minimize errors when you build expressions.

Figure 1-4 shows an example of the Expression Editor:

Figure 1-4. Expression Editor



Entering Port Names into an Expression

For connected transformations, if you use port names in an expression, the Designer updates that expression when you change port names in the transformation. For example, you write a valid expression that determines the difference between two dates, Date_Promised and Date_Delivered. Later, if you change the Date_Promised port name to Due_Date, the Designer changes the Date_Promised port name to Due_Date in the expression.

Note: You can propagate the name Due_Date to other non-reusable transformations that depend on this port in the mapping. For more information, see “Mappings” in the *Designer Guide*.

Adding Comments

You can add comments to an expression to give descriptive information about the expression or to specify a valid URL to access business documentation about the expression.

You can add comments in one of the following ways:

- ◆ To add comments within the expression, use -- or // comment indicators.
- ◆ To add comments in the dialog box, click the Comments button.

For examples on adding comments to expressions, see “The Transformation Language” in the *Transformation Language Reference*.

For more information about linking to business documentation, see “Using the Designer” in the *Designer Guide*.

Validating Expressions

Use the Validate button to validate an expression. If you do not validate an expression, the Designer validates it when you close the Expression Editor. If the expression is invalid, the Designer displays a warning. You can save the invalid expression or modify it. You cannot run a session against a mapping with invalid expressions.

Expression Editor Display

The Expression Editor can display syntax expressions in different colors for better readability. If you have the latest Rich Edit control, riched20.dll, installed on the system, the Expression Editor displays expression functions in blue, comments in grey, and quoted strings in green.

You can resize the Expression Editor. Expand the dialog box by dragging from the borders. The Designer saves the new size for the dialog box as a client setting.

Adding Expressions to an Output Port

Complete the following steps to add an expression to an output port.

To add expressions:

1. In the transformation, select the port and open the Expression Editor.
2. Enter the expression.

Use the Functions and Ports tabs and the operator keys.

3. Add comments to the expression.

Use comment indicators -- or //.

4. Validate the expression.

Use the Validate button to validate the expression.

Using Local Variables

Use local variables in Aggregator, Expression, and Rank transformations. You can reference variables in an expression or use them to temporarily store data. Variables are an easy way to improve performance.

You might use variables to complete the following tasks:

- ◆ Temporarily store data.
- ◆ Simplify complex expressions.
- ◆ Store values from prior rows.
- ◆ Capture multiple return values from a stored procedure.
- ◆ Compare values.
- ◆ Store the results of an unconnected Lookup transformation.

Temporarily Store Data and Simplify Complex Expressions

Variables improve performance when you enter several related expressions in the same transformation. Rather than parsing and validating the same expression components each time, you can define these components as variables.

For example, if an Aggregator transformation uses the same filter condition before calculating sums and averages, you can define this condition once as a variable, and then reuse the condition in both aggregate calculations.

You can simplify complex expressions. If an Aggregator includes the same calculation in multiple expressions, you can improve session performance by creating a variable to store the results of the calculation.

For example, you might create the following expressions to find both the average salary and the total salary using the same data:

```
AVG( SALARY, ( ( JOB_STATUS = 'Full-time' ) AND (OFFICE_ID = 1000 ) ) )  
SUM( SALARY, ( ( JOB_STATUS = 'Full-time' ) AND (OFFICE_ID = 1000 ) ) )
```

Rather than entering the same arguments for both calculations, you might create a variable port for each condition in this calculation, then modify the expression to use the variables.

Table 1-4 shows how to use variables to simplify complex expressions and temporarily store data:

Table 1-4. Variable Usage

Port	Value
V_CONDITION1	JOB_STATUS = 'Full-time'
V_CONDITION2	OFFICE_ID = 1000

Table 1-4. Variable Usage

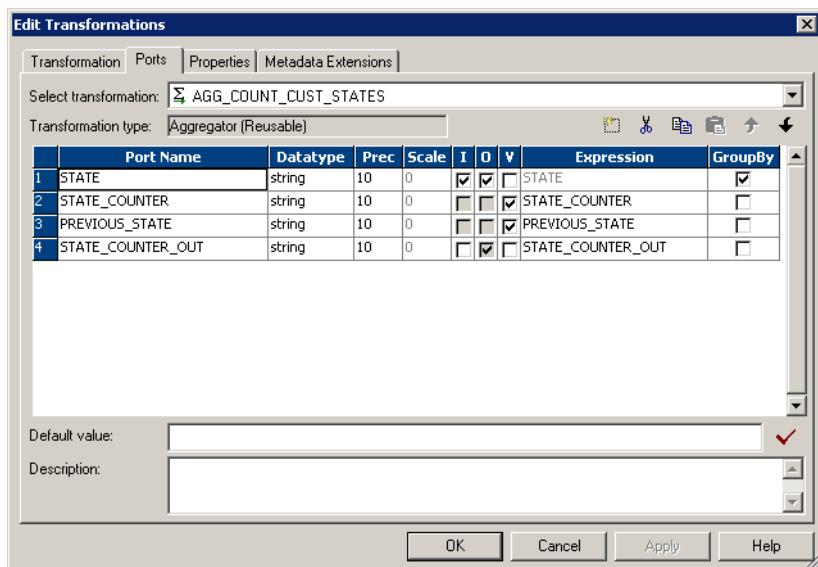
Port	Value
AVG_SALARY	AVG(SALARY, (V_CONDITION1 AND V_CONDITION2))
SUM_SALARY	SUM(SALARY, (V_CONDITION1 AND V_CONDITION2))

Store Values Across Rows

Use variables to store data from prior rows. This can help you perform procedural calculations.

Figure 1-5 shows how to use variables to find out how many customers are in each state:

Figure 1-5. Variable Ports Store Values Across Rows



Since the Integration Service groups the input data by state, the company uses variables to hold the value of the previous state read and a state counter. The following expression compares the previous state to the state just read:

```
IIF(PREVIOUS_STATE = STATE, STATE_COUNTER +1, 1)
```

The STATE_COUNTER is incremented if the row is a member of the previous state. For each new state, the Integration Service sets the counter back to 1. Then an output port passes the value of the state counter to the next transformation.

Capture Values from Stored Procedures

Variables also provide a way to capture multiple columns of return values from stored procedures. For more information, see “Stored Procedure Transformation” on page 489.

Guidelines for Configuring Variable Ports

Consider the following factors when you configure variable ports in a transformation:

- ◆ **Port order.** The Integration Service evaluates ports by dependency. The order of the ports in a transformation must match the order of evaluation: input ports, variable ports, output ports.
- ◆ **Datatype.** The datatype you choose reflects the return value of the expression you enter.
- ◆ **Variable initialization.** The Integration Service sets initial values in variable ports, where you can create counters.

Port Order

The Integration Service evaluates ports in the following order:

1. **Input ports.** The Integration Service evaluates all input ports first since they do not depend on any other ports. Therefore, you can create input ports in any order. Since they do not reference other ports, the Integration Service does not order input ports.
2. **Variable ports.** Variable ports can reference input ports and variable ports, but not output ports. Because variable ports can reference input ports, the Integration Service evaluates variable ports after input ports. Likewise, since variables can reference other variables, the display order for variable ports is the same as the order in which the Integration Service evaluates each variable.

For example, if you calculate the original value of a building and then adjust for depreciation, you might create the original value calculation as a variable port. This variable port needs to appear before the port that adjusts for depreciation.

3. **Output ports.** Because output ports can reference input ports and variable ports, the Integration Service evaluates output ports last. The display order for output ports does not matter since output ports cannot reference other output ports. Be sure output ports display at the bottom of the list of ports.

Datatype

When you configure a port as a variable, you can enter any expression or condition in it. The datatype you choose for this port reflects the return value of the expression you enter. If you specify a condition through the variable port, any numeric datatype returns the values for TRUE (non-zero) and FALSE (zero).

Variable Initialization

The Integration Service does not set the initial value for variables to NULL. Instead, the Integration Service uses the following guidelines to set initial values for variables:

- ◆ Zero for numeric ports
- ◆ Empty strings for string ports
- ◆ 01/01/1753 for Date/Time ports with PMServer 4.0 date handling compatibility disabled
- ◆ 01/01/0001 for Date/Time ports with PMServer 4.0 date handling compatibility enabled

Therefore, use variables as counters, which need an initial value. For example, you can create a numeric variable with the following expression:

```
VAR1 + 1
```

This expression counts the number of rows in the VAR1 port. If the initial value of the variable were set to NULL, the expression would always evaluate to NULL. This is why the initial value is set to zero.

Using Default Values for Ports

All transformations use default values that determine how the Integration Service handles input null values and output transformation errors. Input, output, and input/output ports are created with a system default value that you can sometimes override with a user-defined default value. Default values have different functions in different types of ports:

- ♦ **Input port.** The system default value for null input ports is NULL. It displays as a blank in the transformation. If an input value is NULL, the Integration Service leaves it as NULL.
- ♦ **Output port.** The system default value for output transformation errors is ERROR. The default value appears in the transformation as ERROR('transformation error'). If a transformation error occurs, the Integration Service skips the row. The Integration Service notes all input rows skipped by the ERROR function in the session log file.

The following errors are considered transformation errors:

- Data conversion errors, such as passing a number to a date function.
- Expression evaluation errors, such as dividing by zero.
- Calls to an ERROR function.
- ♦ **Input/output port.** The system default value for null input is the same as input ports, NULL. The system default value appears as a blank in the transformation. The default value for output transformation errors is the same as output ports. The default value for output transformation errors does not display in the transformation.

Table 1-5 shows the system default values for ports in connected transformations:

Table 1-5. System Default Values and Integration Service Behavior

Port Type	Default Value	Integration Service Behavior	User-Defined Default Value Supported
Input, Input/Output	NULL	Integration Service passes all input null values as NULL.	Input Input/Output
Output, Input/Output	ERROR	Integration Service calls the ERROR function for output port transformation errors. The Integration Service skips rows with errors and writes the input data and error message in the session log file.	Output

Note: Variable ports do not support default values. The Integration Service initializes variable ports according to the datatype. For more information, see “Using Local Variables” on page 14.

Figure 1-6 shows that the system default value for input and input/output ports appears as a blank in the transformation:

Figure 1-6. Default Value for Input and Input/Output Ports

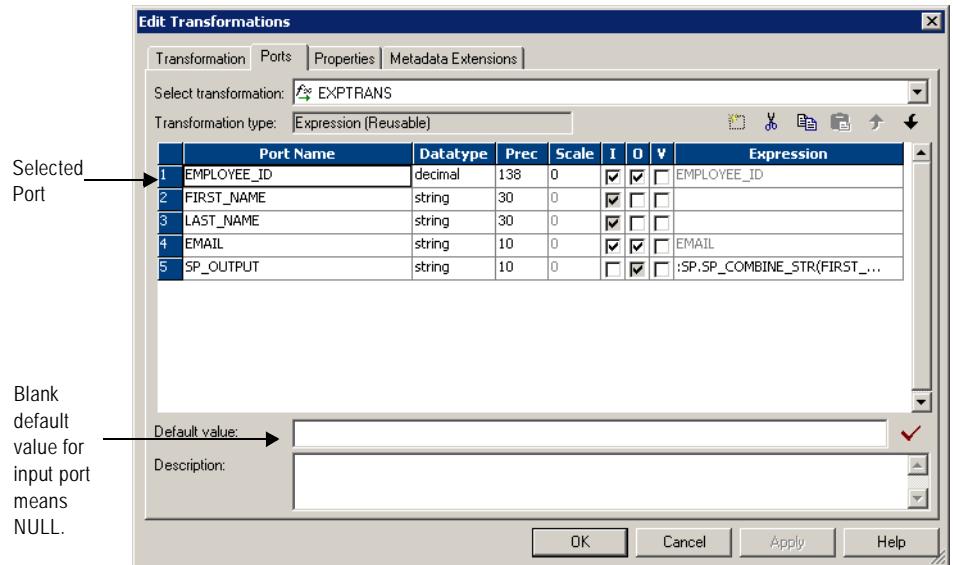
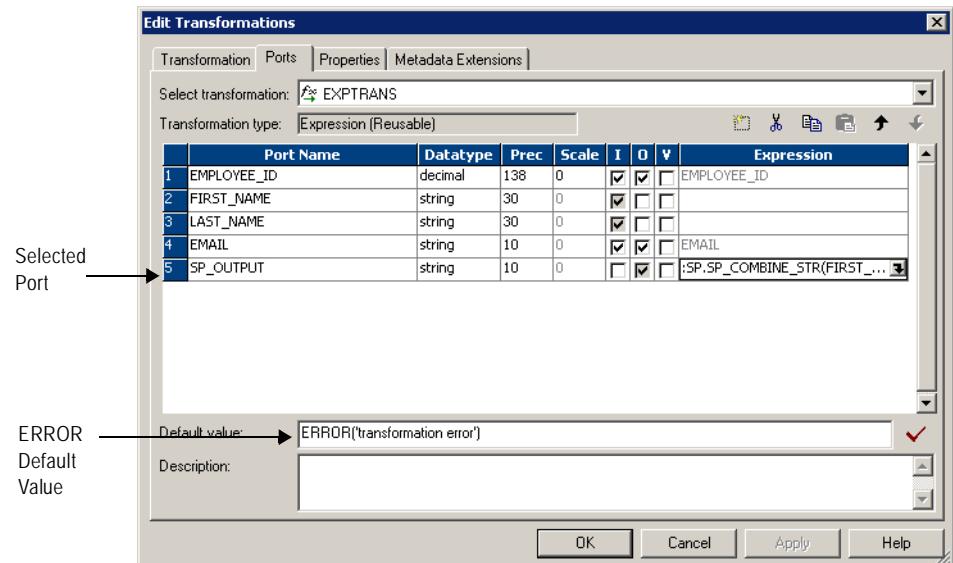


Figure 1-7 shows that the system default value for output ports appears ERROR('transformation error'):

Figure 1-7. Default Value for Output Ports



You can override some of the default values to change the Integration Service behavior when it encounters null input values and output transformation errors.

Entering User-Defined Default Values

You can override the system default values with user-defined default values for supported input, input/output, and output ports within a connected transformation:

- ◆ **Input ports.** You can enter user-defined default values for input ports if you do not want the Integration Service to treat null values as NULL.
- ◆ **Output ports.** You can enter user-defined default values for output ports if you do not want the Integration Service to skip the row or if you want the Integration Service to write a specific message with the skipped row to the session log.
- ◆ **Input/output ports.** You can enter user-defined default values to handle null input values for input/output ports in the same way you can enter user-defined default values for null input values for input ports. You cannot enter user-defined default values for output transformation errors in an input/output port.

Note: The Integration Service ignores user-defined default values for unconnected transformations. For example, if you call a Lookup or Stored Procedure transformation through an expression, the Integration Service ignores any user-defined default value and uses the system default value only.

Table 1-6 shows the ports for each transformation that support user-defined default values:

Table 1-6. Transformations Supporting User-Defined Default Values

Transformation	Input Values for Input Port Input/Output Port	Output Values for Output Port	Output Values for Input/Output Port
Aggregator	Supported	Not Supported	Not Supported
Custom	Supported	Supported	Not Supported
Expression	Supported	Supported	Not Supported
External Procedure	Supported	Supported	Not Supported
Filter	Supported	Not Supported	Not Supported
HTTP	Supported	Not Supported	Not Supported
Java	Supported	Supported	Supported
Lookup	Supported	Supported	Not Supported
Normalizer	Supported	Supported	Not Supported
Rank	Not Supported	Supported	Not Supported
Router	Supported	Not Supported	Not Supported
SQL	Supported	Not Supported	Supported
Stored Procedure	Supported	Supported	Not Supported

Table 1-6. Transformations Supporting User-Defined Default Values

Transformation	Input Values for Input Port Input/Output Port	Output Values for Output Port	Output Values for Input/Output Port
Sequence Generator	n/a	Not Supported	Not Supported
Sorter	Supported	Not Supported	Not Supported
Source Qualifier	Not Supported	n/a	Not Supported
Transaction Control	Not Supported	n/a	Not Supported
Union	Supported	Supported	n/a
Update Strategy	Supported	n/a	Not Supported
XML Generator	n/a	Supported	Not Supported
XML Parser	Supported	n/a	Not Supported
XML Source Qualifier	Not Supported	n/a	Not Supported

Use the following options to enter user-defined default values:

- ◆ **Constant value.** Use any constant (numeric or text), including NULL.
- ◆ **Constant expression.** You can include a transformation function with constant parameters.
- ◆ **ERROR.** Generate a transformation error. Write the row and a message in the session log or row error log. The Integration Service writes the row to session log or row error log based on session configuration.
- ◆ **ABORT.** Abort the session.

Entering Constant Values

You can enter any constant value as a default value. The constant value must match the port datatype. For example, a default value for a numeric port must be a numeric constant. Some constant values include:

```
0
9999
NULL
'Unknown Value'
'Null input data'
```

Entering Constant Expressions

A constant expression is any expression that uses transformation functions (except aggregate functions) to write constant expressions. You cannot use values from input, input/output, or variable ports in a constant expression.

Some valid constant expressions include:

```
500 * 1.75  
TO_DATE('January 1, 1998, 12:05 AM')  
ERROR ('Null not allowed')  
ABORT('Null not allowed')  
SYSDATE
```

You cannot use values from ports within the expression because the Integration Service assigns default values for the entire mapping when it initializes the session. Some invalid default values include the following examples, which incorporate values read from ports:

```
AVG(IN_SALARY)  
IN_PRICE * IN_QUANTITY  
:LKP(LKP_DATES, DATE_SHIPPED)
```

Note: You cannot call a stored procedure or lookup table from a default value expression.

Entering ERROR and ABORT Functions

Use the ERROR and ABORT functions for input and output port default values, and input values for input/output ports. The Integration Service skips the row when it encounters the ERROR function. It aborts the session when it encounters the ABORT function.

Entering User-Defined Default Input Values

You can enter a user-defined default input value if you do not want the Integration Service to treat null values as NULL. You can complete the following functions to override null values:

- ◆ Replace the null value with a constant value or constant expression.
- ◆ Skip the null value with an ERROR function.
- ◆ Abort the session with the ABORT function.

Table 1-7 summarizes how the Integration Service handles null input for input and input/output ports:

Table 1-7. Default Values for Input and Input/Output Ports

Default Value	Default Value Type	Description
NULL (displays blank)	System	Integration Service passes NULL.
Constant or Constant expression	User-Defined	Integration Service replaces the null value with the value of the constant or constant expression.

Table 1-7. Default Values for Input and Input/Output Ports

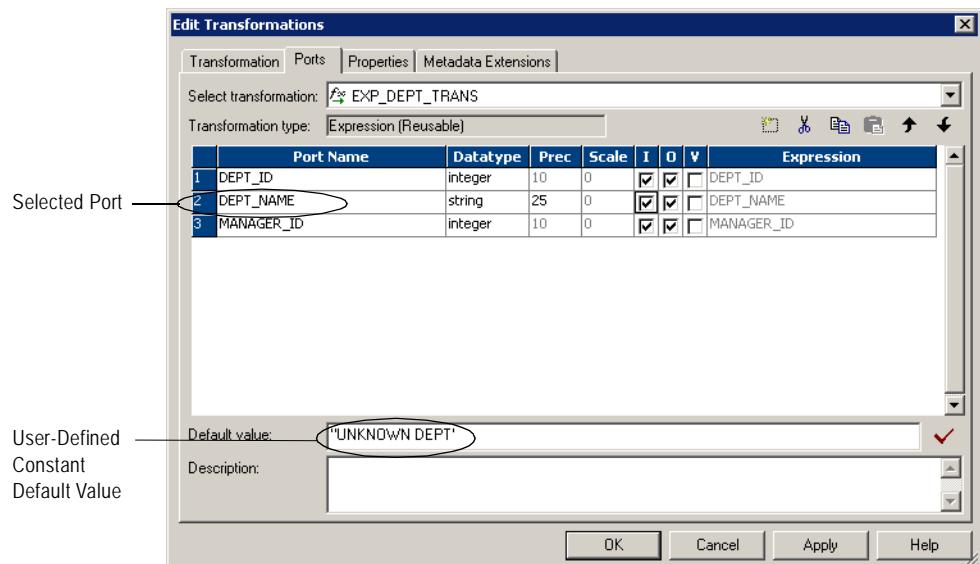
Default Value	Default Value Type	Description
ERROR	User-Defined	Integration Service treats this as a transformation error: - Increases the transformation error count by 1. - Skips the row, and writes the error message to the session log file or row error log. The Integration Service does not write rows to the reject file.
ABORT	User-Defined	Session aborts when the Integration Service encounters a null input value. The Integration Service does not increase the error count or write rows to the reject file.

Replacing Null Values

Use a constant value or expression to substitute a specified value for a NULL. For example, if an input string port is called DEPT_NAME and you want to replace null values with the string ‘UNKNOWN DEPT’, you could set the default value to ‘UNKNOWN DEPT’. Depending on the transformation, the Integration Service passes ‘UNKNOWN DEPT’ to an expression or variable within the transformation or to the next transformation in the data flow.

Figure 1-8 shows a string constant as a user-defined default value for input or input/output ports:

Figure 1-8. Using a Constant as a Default Value



The Integration Service replaces all null values in the EMAIL port with the string 'UNKNOWN DEPT.'

DEPT_NAME	REPLACED VALUE
Housewares	Housewares
NULL	UNKNOWN DEPT
Produce	Produce

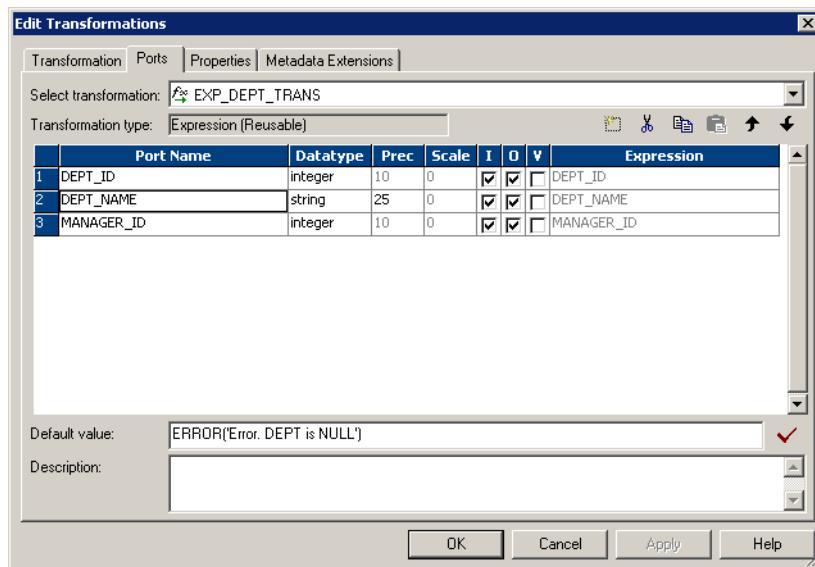
Skipping Null Records

Use the ERROR function as the default value when you do not want null values to pass into a transformation. For example, you might want to skip a row when the input value of DEPT_NAME is NULL. You could use the following expression as the default value:

```
ERROR('Error. DEPT is NULL')
```

Figure 1-9 shows a default value that instructs the Integration Service to skip null values:

Figure 1-9. Using the ERROR Function to Skip Null Input Values



When you use the ERROR function as a default value, the Integration Service skips the row with the null value. The Integration Service writes all rows skipped by the ERROR function into the session log file. It does not write these rows to the session reject file.

DEPT_NAME	RETURN VALUE
Housewares	Housewares
NULL	'Error. DEPT is NULL' (Row is skipped)
Produce	Produce

The following session log shows where the Integration Service skips the row with the null value:

```
TE_11019 Port [DEPT_NAME]: Default value is: ERROR(<<Transformation  
Error>> [error]: Error. DEPT is NULL  
... error('Error. DEPT is NULL')  
).  
CMN_1053 EXPTTRANS: : ERROR: NULL input column DEPT_NAME: Current Input  
data:  
CMN_1053 Input row from SRCTRANS: Rowdata: ( RowType=4 Src Rowid=2 Targ  
Rowid=2  
DEPT_ID (DEPT_ID:Int): "2"  
DEPT_NAME (DEPT_NAME:Char.25): "NULL"  
MANAGER_ID (MANAGER_ID:Int): "1"  
)
```

For more information about the ERROR function, see “Functions” in the *Transformation Language Reference*.

Aborting the Session

Use the ABORT function to abort a session when the Integration Service encounters any null input values. For more information about the ABORT function, see “Functions” in the *Transformation Language Reference*.

Entering User-Defined Default Output Values

You can enter user-defined default values for output ports if you do not want the Integration Service to skip rows with errors or if you want the Integration Service to write a specific message with the skipped row to the session log. You can enter default values to complete the following functions when the Integration Service encounters output transformation errors:

- ◆ Replace the error with a constant value or constant expression. The Integration Service does not skip the row.
- ◆ Abort the session with the ABORT function.
- ◆ Write specific messages in the session log for transformation errors.

You cannot enter user-defined default output values for input/output ports.

Table 1-8 summarizes how the Integration Service handles output port transformation errors and default values in transformations:

Table 1-8. Supported Default Values for Output Ports

Default Value	Default Value Type	Description
Transformation Error	System	When a transformation error occurs and you did not override the default value, the Integration Service performs the following tasks: - Increases the transformation error count by 1. - Skips the row, and writes the error and input row to the session log file or row error log, depending on session configuration. The Integration Service does not write the row to the reject file.
Constant or Constant Expression	User-Defined	Integration Service replaces the error with the default value. The Integration Service does not increase the error count or write a message to the session log.
ABORT	User-Defined	Session aborts and the Integration Service writes a message to the session log. The Integration Service does not increase the error count or write rows to the reject file.

Replacing Errors

If you do not want the Integration Service to skip a row when a transformation error occurs, use a constant or constant expression as the default value for an output port. For example, if you have a numeric output port called NET_SALARY and you want to use the constant value '9999' when a transformation error occurs, assign the default value 9999 to the NET_SALARY port. If there is any transformation error (such as dividing by zero) while computing the value of NET_SALARY, the Integration Service uses the default value 9999.

Aborting the Session

Use the ABORT function as the default value in an output port if you do not want to allow any transformation errors.

Writing Messages in the Session Log or Row Error Logs

You can enter a user-defined default value in the output port if you want the Integration Service to write a specific message in the session log with the skipped row. The system default is ERROR ('transformation error'), and the Integration Service writes the message 'transformation error' in the session log along with the skipped row. You can replace 'transformation error' if you want to write a different message.

When you enable row error logging, the Integration Service writes error messages to the error log instead of the session log and the Integration Service does not log Transaction Control transformation rollback or commit errors. If you want to write rows to the session log in addition to the row error log, you can enable verbose data tracing.

Working with ERROR Functions in Output Port Expressions

If you enter an expression that uses the ERROR function, the user-defined default value for the output port might override the ERROR function in the expression.

For example, you enter the following expression that instructs the Integration Service to use the value 'Negative Sale' when it encounters an error:

```
IIF( TOTAL_SALES>0, TOTAL_SALES, ERROR ('Negative Sale'))
```

The following examples show how user-defined default values may override the ERROR function in the expression:

- ◆ **Constant value or expression.** The constant value or expression overrides the ERROR function in the output port expression.

For example, if you enter '0' as the default value, the Integration Service overrides the ERROR function in the output port expression. It passes the value 0 when it encounters an error. It does not skip the row or write 'Negative Sale' in the session log.

- ◆ **ABORT.** The ABORT function overrides the ERROR function in the output port expression.

If you use the ABORT function as the default value, the Integration Service aborts the session when a transformation error occurs. The ABORT function overrides the ERROR function in the output port expression.

- ◆ **ERROR.** If you use the ERROR function as the default value, the Integration Service includes the following information in the session log:

- Error message from the default value
- Error message indicated in the ERROR function in the output port expression
- Skipped row

For example, you can override the default value with the following ERROR function:

```
ERROR('No default value')
```

The Integration Service skips the row, and includes both error messages in the log.

```
TE_7007 Transformation Evaluation Error; current row skipped...
TE_7007 [<<Transformation Error>> [error]: Negative Sale
... error('Negative Sale')
]
Sun Sep 20 13:57:28 1998
TE_11019 Port [OUT_SALES]: Default value is: ERROR(<<Transformation
Error>> [error]: No default value
... error('No default value')
```

General Rules for Default Values

Use the following rules and guidelines when you create default values:

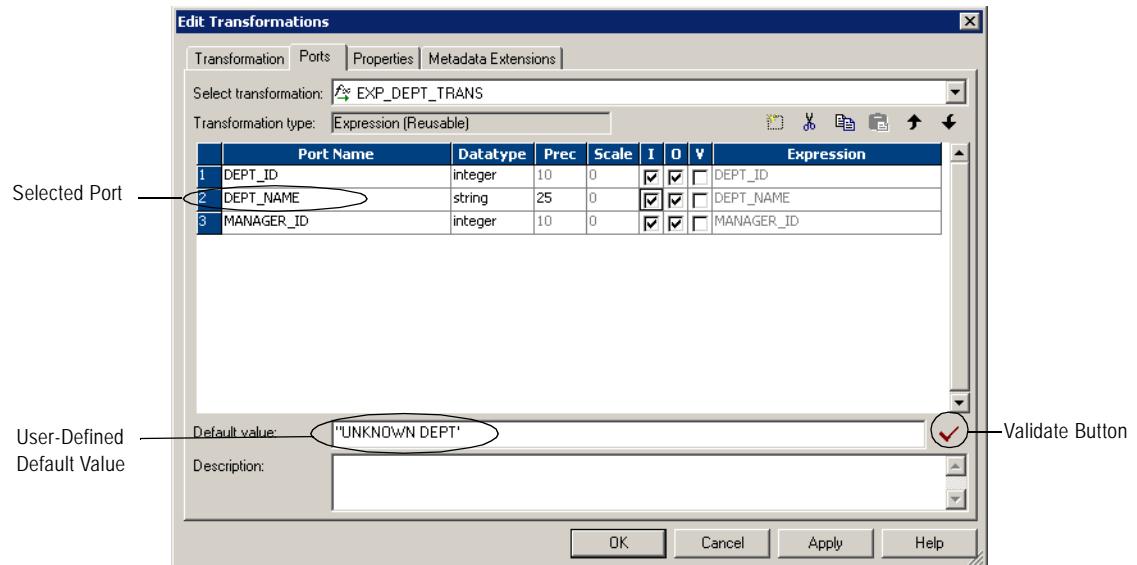
- ◆ The default value must be either a NULL, a constant value, a constant expression, an ERROR function, or an ABORT function.
- ◆ For input/output ports, the Integration Service uses default values to handle null input values. The output default value of input/output ports is always ERROR('Transformation Error').
- ◆ Variable ports do not use default values.
- ◆ You can assign default values to group by ports in the Aggregator and Rank transformations.
- ◆ Not all port types in all transformations allow user-defined default values. If a port does not allow user-defined default values, the default value field is disabled.
- ◆ Not all transformations allow user-defined default value. For more information, see Table 1-6 on page 20.
- ◆ If a transformation is not connected to the mapping data flow (an unconnected transformation), the Integration Service ignores user-defined default values.
- ◆ If any input port is unconnected, its value is assumed to be NULL and the Integration Service uses the default value for that input port.
- ◆ If an input port default value contains the ABORT function and the input value is NULL, the Integration Service immediately stops the session. Use the ABORT function as a default value to restrict null input values. The first null value in an input port stops the session.
- ◆ If an output port default value contains the ABORT function and any transformation error occurs for that port, the session immediately stops. Use the ABORT function as a default value to enforce strict rules for transformation errors. The first transformation error for this port stops the session.
- ◆ The ABORT function, constant values, and constant expressions override ERROR functions configured in output port expressions.

Entering and Validating Default Values

You can validate default values as you enter them. The Designer includes a Validate button so you can ensure valid default values. A message appears indicating if the default is valid.

Figure 1-10 shows the user-defined value for a port and the Validate button:

Figure 1-10. Entering and Validating Default Values



The Designer also validates default values when you save a mapping. If you enter an invalid default value, the Designer marks the mapping invalid.

Configuring Tracing Level in Transformations

When you configure a transformation, you can set the amount of detail the Integration Service writes in the session log.

Table 1-9 describes the session log tracing levels:

Table 1-9. Session Log Tracing Levels

Tracing Level	Description
Normal	Integration Service logs initialization and status information, errors encountered, and skipped rows due to transformation row errors. Summarizes session results, but not at the level of individual rows.
Terse	Integration Service logs initialization information and error messages and notification of rejected data.
Verbose Initialization	In addition to normal tracing, Integration Service logs additional initialization details, names of index and data files used, and detailed transformation statistics.
Verbose Data	In addition to verbose initialization tracing, Integration Service logs each row that passes into the mapping. Also notes where the Integration Service truncates string data to fit the precision of a column and provides detailed transformation statistics. Allows the Integration Service to write errors to both the session log and error log when you enable row error logging. When you configure the tracing level to verbose data, the Integration Service writes row data for all rows in a block when it processes a transformation.

By default, the tracing level for every transformation is Normal. Change the tracing level to a Verbose setting only when you need to debug a transformation that is not behaving as expected. To add a slight performance boost, you can also set the tracing level to Terse, writing the minimum of detail to the session log when running a workflow containing the transformation.

When you configure a session, you can override the tracing levels for individual transformations with a single tracing level for all transformations in the session.

Reusable Transformations

Mappings can contain reusable and non-reusable transformations. Non-reusable transformations exist within a single mapping. Reusable transformations can be used in multiple mappings.

For example, you might create an Expression transformation that calculates value-added tax for sales in Canada, which is useful when you analyze the cost of doing business in that country. Rather than perform the same work every time, you can create a reusable transformation. When you need to incorporate this transformation into a mapping, you add an instance of it to the mapping. Later, if you change the definition of the transformation, all instances of it inherit the changes.

The Designer stores each reusable transformation as metadata separate from any mapping that uses the transformation. If you review the contents of a folder in the Navigator, you see the list of all reusable transformations in that folder.

Each reusable transformation falls within a category of transformations available in the Designer. For example, you can create a reusable Aggregator transformation to perform the same aggregate calculations in multiple mappings, or a reusable Stored Procedure transformation to call the same stored procedure in multiple mappings.

You can create most transformations as a non-reusable or reusable. However, you can only create the External Procedure transformation as a reusable transformation.

When you add instances of a reusable transformation to mappings, you must be careful that changes you make to the transformation do not invalidate the mapping or generate unexpected data.

Instances and Inherited Changes

When you add a reusable transformation to a mapping, you add an instance of the transformation. The definition of the transformation still exists outside the mapping, while a copy (or instance) appears within the mapping.

Since the instance of a reusable transformation is a pointer to that transformation, when you change the transformation in the Transformation Developer, its instances reflect these changes. Instead of updating the same transformation in every mapping that uses it, you can update the reusable transformation once, and all instances of the transformation inherit the change. Note that instances do not inherit changes to property settings, only modifications to ports, expressions, and the name of the transformation.

Mapping Variables in Expressions

Use mapping parameters and variables in reusable transformation expressions. When the Designer validates the parameter or variable, it treats it as an Integer datatype. When you use the transformation in a mapplet or mapping, the Designer validates the expression again. If the mapping parameter or variable does not exist in the mapplet or mapping, the Designer

logs an error. For more information, see “Mapping Parameters and Variables” in the *Designer Guide*.

Creating Reusable Transformations

You can create a reusable transformation using the following methods:

- ◆ **Design it in the Transformation Developer.** In the Transformation Developer, you can build new reusable transformations.
- ◆ **Promote a non-reusable transformation from the Mapping Designer.** After you add a transformation to a mapping, you can promote it to the status of reusable transformation. The transformation designed in the mapping then becomes an instance of a reusable transformation maintained elsewhere in the repository.

If you promote a transformation to reusable status, you cannot demote it. However, you can create a non-reusable instance of it.

Note: Sequence Generator transformations must be reusable in mapplets. You cannot demote reusable Sequence Generator transformations to non-reusable in a mapplet.

To create a reusable transformation:

1. In the Designer, switch to the Transformation Developer.
2. Click the button on the Transformation toolbar corresponding to the type of transformation you want to create.
3. Drag within the workbook to create the transformation.
4. Double-click the transformation title bar to open the dialog displaying its properties.
5. Click the Rename button and enter a descriptive name for the transformation, and click OK.
6. Click the Ports tab, then add any input and output ports you need for this transformation.
7. Set the other properties of the transformation, and click OK.

These properties vary according to the transformation you create. For example, if you create an Expression transformation, you need to enter an expression for one or more of the transformation output ports. If you create a Stored Procedure transformation, you need to identify the stored procedure to call.

8. Click Repository > Save.

Promoting Non-Reusable Transformations

The other technique for creating a reusable transformation is to promote an existing transformation within a mapping. By checking the Make Reusable option in the Edit Transformations dialog box, you instruct the Designer to promote the transformation and create an instance of it in the mapping.

To promote a non-reusable transformation:

1. In the Designer, open a mapping and double-click the title bar of the transformation you want to promote.
2. Select the Make Reusable option.
3. When prompted whether you are sure you want to promote the transformation, click Yes.
4. Click OK to return to the mapping.
5. Click Repository > Save.

Now, when you look at the list of reusable transformations in the folder you are working in, the newly promoted transformation appears in this list.

Creating Non-Reusable Instances of Reusable Transformations

You can create a non-reusable instance of a reusable transformation within a mapping. Reusable transformations must be made non-reusable within the same folder. If you want to have a non-reusable instance of a reusable transformation in a different folder, you need to first make a non-reusable instance of the transformation in the source folder, and then copy it into the target folder.

To create a non-reusable instance of a reusable transformation:

1. In the Designer, open a mapping.
2. In the Navigator, select an existing transformation and drag the transformation into the mapping workspace. Hold down the Ctrl key before you release the transformation.

The status bar displays the following message:

Make a non-reusable copy of this transformation and add it to this mapping.

3. Release the transformation.

The Designer creates a non-reusable instance of the existing reusable transformation.

4. Click Repository > Save.

Adding Reusable Transformations to Mappings

After you create a reusable transformation, you can add it to mappings.

To add a reusable transformation:

1. In the Designer, switch to the Mapping Designer.
2. Open or create a mapping.
3. In the list of repository objects, drill down until you find the reusable transformation you want in the Transformations section of a folder.
4. Drag the transformation from the Navigator into the mapping.

A copy (or instance) of the reusable transformation appears.

5. Link the new transformation to other transformations or target definitions.
6. Click Repository > Save.

Modifying a Reusable Transformation

Changes to a reusable transformation that you enter through the Transformation Developer are immediately reflected in all instances of that transformation. While this feature is a powerful way to save work and enforce standards (for example, by publishing the official version of a depreciation calculation through a reusable transformation), you risk invalidating mappings when you modify a reusable transformation.

To see what mappings, mapplets, or shortcuts may be affected by changes you make to a transformation, select the transformation in the workspace or Navigator, right-click, and select View Dependencies.

If you make any of the following changes to the reusable transformation, mappings that use instances of it may be invalidated:

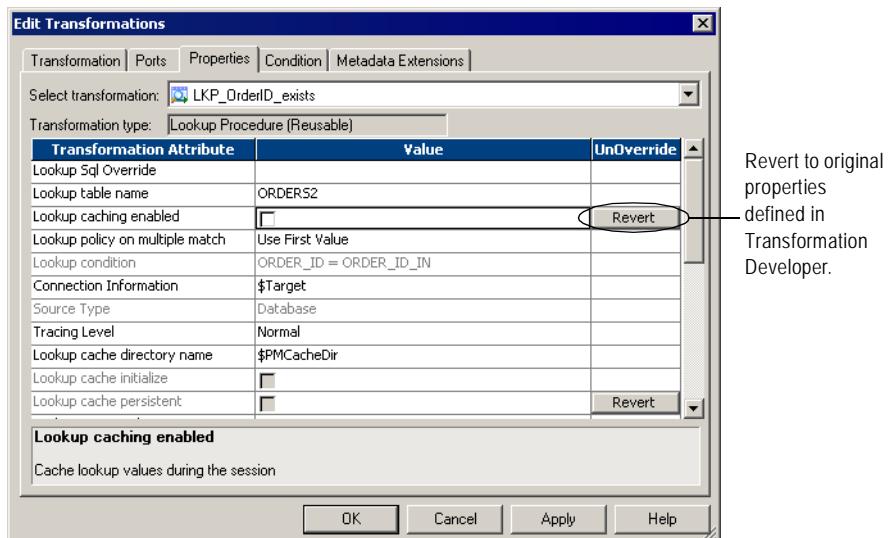
- ◆ When you delete a port or multiple ports in a transformation, you disconnect the instance from part or all of the data flow through the mapping.
- ◆ When you change a port datatype, you make it impossible to map data from that port to another port using an incompatible datatype.
- ◆ When you change a port name, expressions that refer to the port are no longer valid.
- ◆ When you enter an invalid expression in the reusable transformation, mappings that use the transformation are no longer valid. The Integration Service cannot run sessions based on invalid mappings.

Reverting to Original Reusable Transformation

If you change the properties of a reusable transformation in a mapping, you can revert to the original reusable transformation properties by clicking the Revert button.

Figure 1-11 shows how you can revert to the original properties of the reusable transformation:

Figure 1-11. Reverting to Original Reusable Transformation Properties



Chapter 2

Aggregator Transformation

This chapter includes the following topics:

- ◆ Overview, 38
- ◆ Aggregate Expressions, 40
- ◆ Group By Ports, 42
- ◆ Using Sorted Input, 45
- ◆ Creating an Aggregator Transformation, 47
- ◆ Tips, 50
- ◆ Troubleshooting, 51

Overview

Transformation type:

Active
Connected

The Aggregator transformation lets you perform aggregate calculations, such as averages and sums. The Aggregator transformation is unlike the Expression transformation, in that you use the Aggregator transformation to perform calculations on groups. The Expression transformation permits you to perform calculations on a row-by-row basis only.

When using the transformation language to create aggregate expressions, use conditional clauses to filter rows, providing more flexibility than SQL language.

The Integration Service performs aggregate calculations as it reads, and stores necessary data group and row data in an aggregate cache.

After you create a session that includes an Aggregator transformation, you can enable the session option, Incremental Aggregation. When the Integration Service performs incremental aggregation, it passes new source data through the mapping and uses historical cache data to perform new aggregation calculations incrementally. For information about incremental aggregation, see “Using Incremental Aggregation” in the *Workflow Administration Guide*.

Ports in the Aggregator Transformation

To configure ports in the Aggregator transformation, complete the following tasks:

- ♦ Enter an expression in any output port, using conditional clauses or non-aggregate functions in the port.
- ♦ Create multiple aggregate output ports.
- ♦ Configure any input, input/output, output, or variable port as a group by port.
- ♦ Improve performance by connecting only the necessary input/output ports to subsequent transformations, reducing the size of the data cache.
- ♦ Use variable ports for local variables.
- ♦ Create connections to other transformations as you enter an expression.

Components of the Aggregator Transformation

The Aggregator is an active transformation, changing the number of rows in the pipeline. The Aggregator transformation has the following components and options:

- ♦ **Aggregate expression.** Entered in an output port. Can include non-aggregate expressions and conditional clauses.
- ♦ **Group by port.** Indicates how to create groups. The port can be any input, input/output, output, or variable port. When grouping data, the Aggregator transformation outputs the last row of each group unless otherwise specified.

- ◆ **Sorted input.** Use to improve session performance. To use sorted input, you must pass data to the Aggregator transformation sorted by group by port, in ascending or descending order.
- ◆ **Aggregate cache.** The Integration Service stores data in the aggregate cache until it completes aggregate calculations. It stores group values in an index cache and row data in the data cache.

Aggregate Caches

When you run a session that uses an Aggregator transformation, the Integration Service creates index and data caches in memory to process the transformation. If the Integration Service requires more space, it stores overflow values in cache files.

You can configure the index and data caches in the Aggregator transformation or in the session properties. Or, you can configure the Integration Service to determine the cache size at runtime.

For more information about configuring index and data caches, see “Creating an Aggregator Transformation” on page 47.

For information about configuring the Integration Service to determine the cache size at runtime, see “Working with Sessions” in the *Workflow Administration Guide*.

Note: The Integration Service uses memory to process an Aggregator transformation with sorted ports. It does not use cache memory. You do not need to configure cache memory for Aggregator transformations that use sorted ports.

Aggregate Expressions

The Designer allows aggregate expressions only in the Aggregator transformation. An aggregate expression can include conditional clauses and non-aggregate functions. It can also include one aggregate function nested within another aggregate function, such as:

```
MAX( COUNT( ITEM ) )
```

The result of an aggregate expression varies depending on the group by ports used in the transformation. For example, when the Integration Service calculates the following aggregate expression with no group by ports defined, it finds the total quantity of items sold:

```
SUM( QUANTITY )
```

However, if you use the same expression, and you group by the ITEM port, the Integration Service returns the total quantity of items sold, by item.

You can create an aggregate expression in any output port and use multiple aggregate ports in a transformation.

Aggregate Functions

Use the following aggregate functions within an Aggregator transformation. You can nest one aggregate function within another aggregate function.

The transformation language includes the following aggregate functions:

- ◆ AVG
- ◆ COUNT
- ◆ FIRST
- ◆ LAST
- ◆ MAX
- ◆ MEDIAN
- ◆ MIN
- ◆ PERCENTILE
- ◆ STDDEV
- ◆ SUM
- ◆ VARIANCE

When you use any of these functions, you must use them in an expression within an Aggregator transformation. For a description of these functions, see “Functions” in the *Transformation Language Reference*.

Nested Aggregate Functions

You can include multiple single-level or multiple nested functions in different output ports in an Aggregator transformation. However, you cannot include both single-level and nested

functions in an Aggregator transformation. Therefore, if an Aggregator transformation contains a single-level function in any output port, you cannot use a nested function in any other port in that transformation. When you include single-level and nested functions in the same Aggregator transformation, the Designer marks the mapping or mapplet invalid. If you need to create both single-level and nested functions, create separate Aggregator transformations.

Conditional Clauses

Use conditional clauses in the aggregate expression to reduce the number of rows used in the aggregation. The conditional clause can be any clause that evaluates to TRUE or FALSE.

For example, use the following expression to calculate the total commissions of employees who exceeded their quarterly quota:

```
SUM( COMMISSION, COMMISSION > QUOTA )
```

Non-Aggregate Functions

You can also use non-aggregate functions in the aggregate expression.

The following expression returns the highest number of items sold for each item (grouped by item). If no items were sold, the expression returns 0.

```
IIF( MAX( QUANTITY ) > 0, MAX( QUANTITY ), 0 )
```

Null Values in Aggregate Functions

When you configure the Integration Service, you can choose how you want the Integration Service to handle null values in aggregate functions. You can choose to treat null values in aggregate functions as NULL or zero. By default, the Integration Service treats null values as NULL in aggregate functions.

For information about changing this default behavior, see “Configuring Services” in the *Installation and Configuration Guide*.

Group By Ports

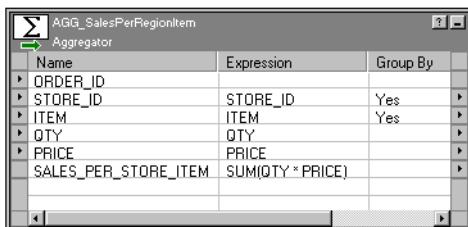
The Aggregator transformation lets you define groups for aggregations, rather than performing the aggregation across all input data. For example, rather than finding the total company sales, you can find the total sales grouped by region.

To define a group for the aggregate expression, select the appropriate input, input/output, and variable ports in the Aggregator transformation. You can select multiple group by ports, creating a new group for each unique combination of groups. The Integration Service then performs the defined aggregation for each group.

When you group values, the Integration Service produces one row for each group. If you do not group values, the Integration Service returns one row for all input rows. The Integration Service typically returns the last row of each group (or the last row received) with the result of the aggregation. However, if you specify a particular row to be returned (for example, by using the FIRST function), the Integration Service then returns the specified row.

When selecting multiple group by ports in the Aggregator transformation, the Integration Service uses port order to determine the order by which it groups. Since group order can affect the results, order group by ports to ensure the appropriate grouping. For example, the results of grouping by ITEM_ID then QUANTITY can vary from grouping by QUANTITY then ITEM_ID, because the numeric values for quantity are not necessarily unique.

The following Aggregator transformation groups first by STORE_ID and then by ITEM:



If you send the following data through this Aggregator transformation:

STORE_ID	ITEM	QTY	PRICE
101	'battery'	3	2.99
101	'battery'	1	3.19
101	'battery'	2	2.59
101	'AAA'	2	2.45
201	'battery'	1	1.99
201	'battery'	4	1.59
301	'battery'	1	2.45

The Integration Service performs the aggregate calculation on the following unique groups:

STORE_ID	ITEM
101	'battery'
101	'AAA'
201	'battery'
301	'battery'

The Integration Service then passes the last row received, along with the results of the aggregation, as follows:

STORE_ID	ITEM	QTY	PRICE	SALES_PER_STORE
101	'battery'	2	2.59	17.34
101	'AAA'	2	2.45	4.90
201	'battery'	4	1.59	8.35
301	'battery'	1	2.45	2.45

Non-Aggregate Expressions

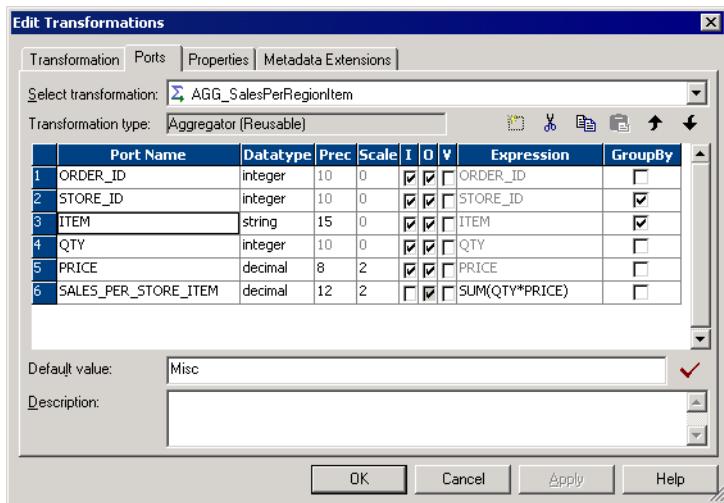
Use non-aggregate expressions in group by ports to modify or replace groups. For example, if you want to replace 'AAA battery' before grouping, you can create a new group by output port, named CORRECTED_ITEM, using the following expression:

```
IIF( ITEM = 'AAA battery', battery, ITEM )
```

Default Values

Use default values in the group by port to replace null input values. This allows the Integration Service to include null item groups in the aggregation. For more information about default values, see “Using Default Values for Ports” on page 18.

For example, if you define a default value of ‘Misc’ in the ITEM column as shown below, the Integration Service replaces null groups with ‘Misc’:



Using Sorted Input

You can improve Aggregator transformation performance by using the sorted input option. When you use sorted input, the Integration Service assumes all data is sorted by group. As the Integration Service reads rows for a group, it performs aggregate calculations. When necessary, it stores group information in memory. To use the Sorted Input option, you must pass sorted data to the Aggregator transformation. You can gain performance with sorted ports when you configure the session with multiple partitions.

When you do not use sorted input, the Integration Service performs aggregate calculations as it reads. However, since data is not sorted, the Integration Service stores data for each group until it reads the entire source to ensure all aggregate calculations are accurate.

For example, one Aggregator transformation has the STORE_ID and ITEM group by ports, with the sorted input option selected. When you pass the following data through the Aggregator, the Integration Service performs an aggregation for the three rows in the 101/battery group as soon as it finds the new group, 201/battery:

STORE_ID	ITEM	QTY	PRICE
101	'battery'	3	2.99
101	'battery'	1	3.19
101	'battery'	2	2.59
201	'battery'	4	1.59
201	'battery'	1	1.99

If you use sorted input and do not presort data correctly, you receive unexpected results.

Sorted Input Conditions

Do not use sorted input if either of the following conditions are true:

- ◆ The aggregate expression uses nested aggregate functions.
- ◆ The session uses incremental aggregation.

If you use sorted input and do not sort data correctly, the session fails.

Pre-Sorting Data

To use sorted input, you pass sorted data through the Aggregator.

Data must be sorted as follows:

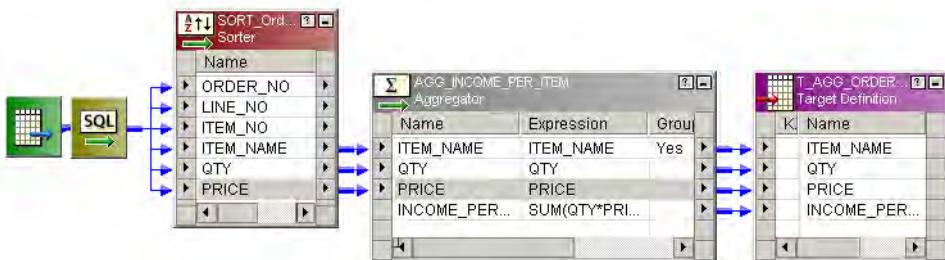
- ◆ By the Aggregator group by ports, in the order they appear in the Aggregator transformation.
- ◆ Using the same sort order configured for the session. If data is not in strict ascending or descending order based on the session sort order, the Integration Service fails the session. For example, if you configure a session to use a French sort order, data passing into the Aggregator transformation must be sorted using the French sort order.

For relational and file sources, use the Sorter transformation to sort data in the mapping before passing it to the Aggregator transformation. You can place the Sorter transformation anywhere in the mapping prior to the Aggregator if no transformation changes the order of the sorted data. Group by columns in the Aggregator transformation must be in the same order as they appear in the Sorter transformation. For information about sorting data using the Sorter transformation, see “Sorter Transformation” on page 415.

If the session uses relational sources, you can also use the Number of Sorted Ports option in the Source Qualifier transformation to sort group by columns in the source database. Group by columns must be in the same order in both the Aggregator and Source Qualifier transformations. For information about sorting data in the Source Qualifier, see “Using Sorted Ports” on page 452.

Figure 2-1 shows the mapping with a Sorter transformation configured to sort the source data in descending order by ITEM_NAME:

Figure 2-1. Sample Mapping with Aggregator and Sorter Transformations



The Sorter transformation sorts the data as follows:

ITEM_NAME	QTY	PRICE
Soup	4	2.95
Soup	1	2.95
Soup	2	3.25
Cereal	1	4.49
Cereal	2	5.25

With sorted input, the Aggregator transformation returns the following results:

ITEM_NAME	QTY	PRICE	INCOME_PER_ITEM
Cereal	2	5.25	14.99
Soup	2	3.25	21.25

Creating an Aggregator Transformation

To use an Aggregator transformation in a mapping, add the Aggregator transformation to the mapping. Then configure the transformation with an aggregate expression and group by ports.

To create an Aggregator transformation:

1. In the Mapping Designer, click Transformation > Create. Select the Aggregator transformation.
2. Enter a name for the Aggregator, click Create. Then click Done.

The Designer creates the Aggregator transformation.
3. Drag the ports to the Aggregator transformation.

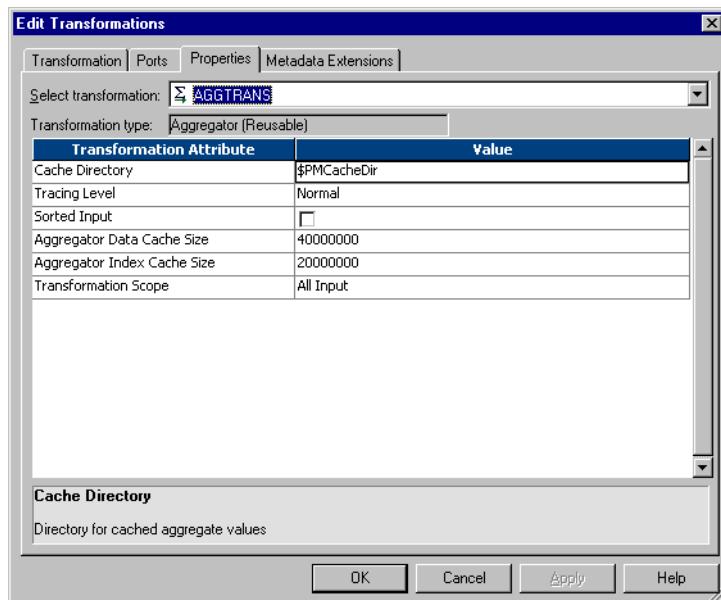
The Designer creates input/output ports for each port you include.
4. Double-click the title bar of the transformation to open the Edit Transformations dialog box.
5. Select the Ports tab.
6. Click the group by option for each column you want the Aggregator to use in creating groups.

Optionally, enter a default value to replace null groups.
If you want to use a non-aggregate expression to modify groups, click the Add button and enter a name and data type for the port. Make the port an output port by clearing Input (I). Click in the right corner of the Expression field, enter the non-aggregate expression using one of the input ports, and click OK. Select Group By.
7. Click Add and enter a name and data type for the aggregate expression port. Make the port an output port by clearing Input (I). Click in the right corner of the Expression field to open the Expression Editor. Enter the aggregate expression, click Validate, and click OK.

Make sure the expression validates before closing the Expression Editor.
8. Add default values for specific ports.

If certain ports are likely to contain null values, you might specify a default value if the target database does not handle null values.

9. Select the Properties tab.



Select and modify these options:

Aggregator Setting	Description
Cache Directory	Local directory where the Integration Service creates the index and data cache files. By default, the Integration Service uses the directory entered in the Workflow Manager for the process variable \$PMCacheDir. If you enter a new directory, make sure the directory exists and contains enough disk space for the aggregate caches. If you have enabled incremental aggregation, the Integration Service creates a backup of the files each time you run the session. The cache directory must contain enough disk space for two sets of the files. For information about incremental aggregation, see "Using Incremental Aggregation" in the <i>Workflow Administration Guide</i> .
Tracing Level	Amount of detail displayed in the session log for this transformation.
Sorted Input	Indicates input data is presorted by groups. Select this option only if the mapping passes sorted data to the Aggregator transformation.
Aggregator Data Cache Size	Data cache size for the transformation. Default cache size is 2,000,000 bytes. If the total configured session cache size is 2 GB (2,147,483,648 bytes) or greater, you must run the session on a 64-bit Integration Service. You can configure the Integration Service to determine the cache size at runtime, or you can configure a numeric value. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.

Aggregator Setting	Description
Aggregator Index Cache Size	Index cache size for the transformation. Default cache size is 1,000,000 bytes. If the total configured session cache size is 2 GB (2,147,483,648 bytes) or greater, you must run the session on a 64-bit Integration Service. You can configure the Integration Service to determine the cache size at runtime, or you can configure a numeric value. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.
Transformation Scope	<p>Specifies how the Integration Service applies the transformation logic to incoming data:</p> <ul style="list-style-type: none"> - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions. - All Input. Applies the transformation logic on all incoming data. When you choose All Input, the PowerCenter drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source. <p>For more information about transformation scope, see "Understanding Commit Points" in the <i>Workflow Administration Guide</i>.</p>

10. Click OK.

11. Click Repository > Save to save changes to the mapping.

Tips

Use the following guidelines to optimize the performance of an Aggregator transformation.

Use sorted input to decrease the use of aggregate caches.

Sorted input reduces the amount of data cached during the session and improves session performance. Use this option with the Sorter transformation to pass sorted data to the Aggregator transformation.

Limit connected input/output or output ports.

Limit the number of connected input/output or output ports to reduce the amount of data the Aggregator transformation stores in the data cache.

Filter before aggregating.

If you use a Filter transformation in the mapping, place the transformation before the Aggregator transformation to reduce unnecessary aggregation.

Troubleshooting

I selected sorted input but the workflow takes the same amount of time as before.

You cannot use sorted input if any of the following conditions are true:

- ◆ The aggregate expression contains nested aggregate functions.
- ◆ The session uses incremental aggregation.
- ◆ Source data is data driven.

When any of these conditions are true, the Integration Service processes the transformation as if you do not use sorted input.

A session using an Aggregator transformation causes slow performance.

The Integration Service may be paging to disk during the workflow. You can increase session performance by increasing the index and data cache sizes in the transformation properties. For more information about caching, see “Session Caches” in the *Workflow Administration Guide*.

I entered an override cache directory in the Aggregator transformation, but the Integration Service saves the session incremental aggregation files somewhere else.

You can override the transformation cache directory on a session level. The Integration Service notes the cache directory in the session log. You can also check the session properties for an override cache directory.

Chapter 3

Custom Transformation

This chapter includes the following topics:

- ◆ Overview, 54
- ◆ Creating Custom Transformations, 57
- ◆ Working with Groups and Ports, 59
- ◆ Working with Port Attributes, 62
- ◆ Custom Transformation Properties, 64
- ◆ Working with Transaction Control, 68
- ◆ Blocking Input Data, 70
- ◆ Working with Procedure Properties, 72
- ◆ Creating Custom Transformation Procedures, 73

Overview

Transformation type:

Active/Passive
Connected

Custom transformations operate in conjunction with procedures you create outside of the Designer interface to extend PowerCenter functionality. You can create a Custom transformation and bind it to a procedure that you develop using the functions described in “Custom Transformation Functions” on page 89.

Use the Custom transformation to create transformation applications, such as sorting and aggregation, which require all input rows to be processed before outputting any output rows. To support this process, the input and output functions occur separately in Custom transformations compared to External Procedure transformations.

The Integration Service passes the input data to the procedure using an input function. The output function is a separate function that you must enter in the procedure code to pass output data to the Integration Service. In contrast, in the External Procedure transformation, an external procedure function does both input and output, and its parameters consist of all the ports of the transformation.

You can also use the Custom transformation to create a transformation that requires multiple input groups, multiple output groups, or both. A group is the representation of a row of data entering or leaving a transformation. For example, you might create a Custom transformation with one input group and multiple output groups that parses XML data. Or, you can create a Custom transformation with two input groups and one output group that merges two streams of input data into one stream of output data.

Working with Transformations Built On the Custom Transformation

You can build transformations using the Custom transformation. Some of the PowerCenter transformations are built using the Custom transformation. Rules that apply to Custom transformations, such as blocking rules, also apply to transformations built using Custom transformations. For example, when you connect a Custom transformation in a mapping, you must verify that the data can flow from all sources in a target load order group to the targets without the Integration Service blocking all sources. Similarly, you must also verify this for transformations built using a Custom transformation. For more information about data flow validation, see “Mappings” in the *Designer Guide*.

The following transformations that ship with Informatica products are built using the Custom transformation:

- ◆ HTTP transformation with PowerCenter
- ◆ Java transformation with PowerCenter
- ◆ SQL transformation with PowerCenter
- ◆ Union transformation with PowerCenter

- ◆ XML Parser transformation with PowerCenter
- ◆ XML Generator transformation with PowerCenter
- ◆ SAP/ALE_IDoc_Interpreter transformation with PowerCenter Connect for SAP NetWeaver mySAP Option
- ◆ SAP/ALE_IDoc_Prepare transformation with PowerCenter Connect for SAP NetWeaver mySAP Option
- ◆ Web Service Consumer transformation with PowerCenter Connect for Web Services
- ◆ Address transformation with Data Cleansing Option
- ◆ Parse transformation with Data Cleansing Option

Code Page Compatibility

The Custom transformation procedure code page is the code page of the data the Custom transformation procedure processes. The following factors determine the Custom transformation procedure code page:

- ◆ Integration Service data movement mode
- ◆ The INFA_CTChangeStringMode() function
- ◆ The INFA_CTSGetDataCodePageID() function

The Custom transformation procedure code page must be two-way compatible with the Integration Service code page. The Integration Service passes data to the procedure in the Custom transformation procedure code page. Also, the data the procedure passes to the Integration Service must be valid characters in the Custom transformation procedure code page.

By default, when the Integration Service runs in ASCII mode, the Custom transformation procedure code page is ASCII. Also, when the Integration Service runs in Unicode mode, the Custom transformation procedure code page is UCS-2, but the Integration Service only passes characters that are valid in the Integration Service code page.

However, use the INFA_CTChangeStringMode() functions in the procedure code to request the data in a different format. In addition, when the Integration Service runs in Unicode mode, you can request the data in a different code page using the INFA_CTSGetDataCodePageID() function.

Changing the format or requesting the data in a different code page changes the Custom transformation procedure code page to the code page the procedure requests:

- ◆ **ASCII mode.** You can write the external procedure code to request the data in UCS-2 format using the INFA_CTChangeStringMode() function. When you use this function, the procedure must pass only ASCII characters in UCS-2 format to the Integration Service. Do not use the INFA_CTSGetDataCodePageID() function when the Integration Service runs in ASCII mode.
- ◆ **Unicode mode.** You can write the external procedure code to request the data in MBCS using the INFA_CTChangeStringMode() function. When the external procedure requests the data in MBCS, the Integration Service passes the data in the Integration Service code

page. When you use the INFA_CTChangeStringMode() function, you can write the external procedure code to request the data in a different code page from the Integration Service code page using the INFA_CTSGetDataCodePageID() function. The code page you specify in the INFA_CTSGetDataCodePageID() function must be two-way compatible with the Integration Service code page.

Note: You can also use the INFA_CTRereadInputDataType() function to change the format for a specific port in the Custom transformation.

Distributing Custom Transformation Procedures

You can copy a Custom transformation from one repository to another. When you copy a Custom transformation between repositories, you must verify that the Integration Service machine the target repository uses contains the Custom transformation procedure.

Creating Custom Transformations

You can create reusable Custom transformations in the Transformation Developer, and add instances of the transformation to mappings. You can create non-reusable Custom transformations in the Mapping Designer or Mapplet Designer.

Each Custom transformation specifies a module and a procedure name. You can create a Custom transformation based on an existing shared library or DLL containing the procedure, or you can create a Custom transformation as the basis for creating the procedure. When you create a Custom transformation to use with an existing shared library or DLL, make sure you define the correct module and procedure name.

When you create a Custom transformation as the basis for creating the procedure, select the transformation and generate the code. The Designer uses the transformation properties when it generates the procedure code. It generates code in a single directory for all transformations sharing a common module name.

The Designer generates the following files:

- ◆ **m_<module_name>.c.** Defines the module. This file includes an initialization function, `m_<module_name>_moduleInit()` that lets you write code you want the Integration Service to run when it loads the module. Similarly, this file includes a deinitialization function, `m_<module_name>_moduleDeinit()`, that lets you write code you want the Integration Service to run before it unloads the module.
- ◆ **p_<procedure_name>.c.** Defines the procedure in the module. This file contains the code that implements the procedure logic, such as data cleansing or merging data.
- ◆ **makefile.aix, makefile.aix64, makefile.hp, makefile.hp64, makefile.hpparisc64, makefile.linux, makefile.sol, and makefile.sol64.** Make files for the UNIX platforms. Use `makefile.aix64` for 64-bit AIX platforms, `makefile.sol64` for 64-bit Solaris platforms, and `makefile.hp64` for 64-bit HP-UX (Itanium) platforms.

Rules and Guidelines

Use the following rules and guidelines when you create a Custom transformation:

- ◆ Custom transformations are connected transformations. You cannot reference a Custom transformation in an expression.
- ◆ You can include multiple procedures in one module. For example, you can include an XML writer procedure and an XML parser procedure in the same module.
- ◆ You can bind one shared library or DLL to multiple Custom transformation instances if you write the procedure code to handle multiple Custom transformation instances.
- ◆ When you write the procedure code, you must make sure it does not violate basic mapping rules. For more information about mappings and mapping validation, see “Mappings” in the *Designer Guide*.
- ◆ The Custom transformation sends and receives high precision decimals as high precision decimals.

- ◆ Use multi-threaded code in Custom transformation procedures.

Custom Transformation Components

When you configure a Custom transformation, you define the following components:

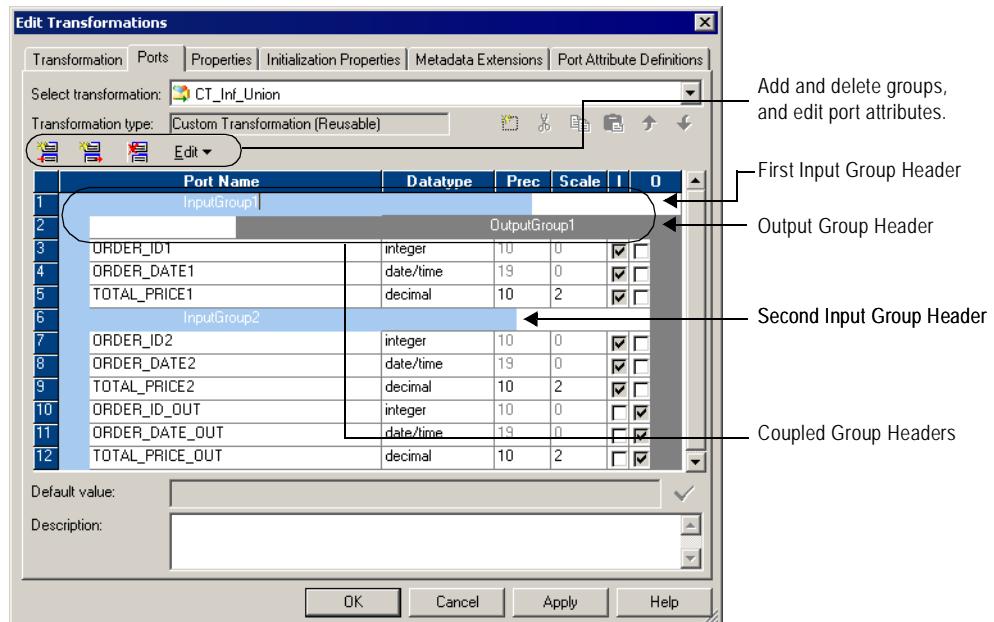
- ◆ **Transformation tab.** You can rename the transformation and add a description on the Transformation tab.
- ◆ **Ports tab.** You can add and edit ports and groups to a Custom transformation. For more information about creating ports and groups, see “Working with Groups and Ports” on page 59. You can also define the input ports an output port depends on. For more information about defining port dependencies, see “Defining Port Relationships” on page 60.
- ◆ **Port Attribute Definitions tab.** You can create user-defined port attributes for Custom transformation ports. For more information about creating and editing port attributes, see “Working with Port Attributes” on page 62.
- ◆ **Properties tab.** You can define transformation properties such as module and function identifiers, transaction properties, and the runtime location. For more information about defining transformation properties, see “Custom Transformation Properties” on page 64.
- ◆ **Initialization Properties tab.** You can define properties that the external procedure uses at runtime, such as during initialization. For more information about creating initialization properties, see “Working with Procedure Properties” on page 72.
- ◆ **Metadata Extensions tab.** You can create metadata extensions to define properties that the procedure uses at runtime, such as during initialization. For more information about using metadata extensions for procedure properties, see “Working with Procedure Properties” on page 72.

Working with Groups and Ports

A Custom transformation has both input and output groups. It also can have input ports, output ports, and input/output ports. You create and edit groups and ports on the Ports tab of the Custom transformation. You can also define the relationship between input and output ports on the Ports tab.

Figure 3-1 shows the Custom transformation Ports tab:

Figure 3-1. Custom Transformation Ports Tab



Creating Groups and Ports

You can create multiple input groups and multiple output groups in a Custom transformation. You must create at least one input group and one output group. To create an input group, click the Create Input Group icon. To create an output group, click the Create Output Group icon. When you create a group, the Designer adds it as the last group. When you create a passive Custom transformation, you can only create one input group and one output group.

To create a port, click the Add button. When you create a port, the Designer adds it below the currently selected row or group. Each port contains attributes defined on the Port Attribute Definitions tab. You can edit the attributes for each port. For more information about creating and editing user-defined port attributes, see “Working with Port Attributes” on page 62.

Editing Groups and Ports

Use the following rules and guidelines when you edit ports and groups in a Custom transformation:

- ◆ You can change group names by typing in the group header.
- ◆ You can only enter ASCII characters for port and group names.
- ◆ Once you create a group, you cannot change the group type. If you need to change the group type, delete the group and add a new group.
- ◆ When you delete a group, the Designer deletes all ports of the same type in that group. However, all input/output ports remain in the transformation, belong to the group above them, and change to input ports or output ports, depending on the type of group you delete. For example, an output group contains output ports and input/output ports. You delete the output group. The Designer deletes the output ports. It changes the input/output ports to input ports. Those input ports belong to the input group with the header directly above them.
- ◆ To move a group up or down, select the group header and click the Move Port Up or Move Port Down button. The ports above and below the group header remain the same, but the groups to which they belong might change.

Defining Port Relationships

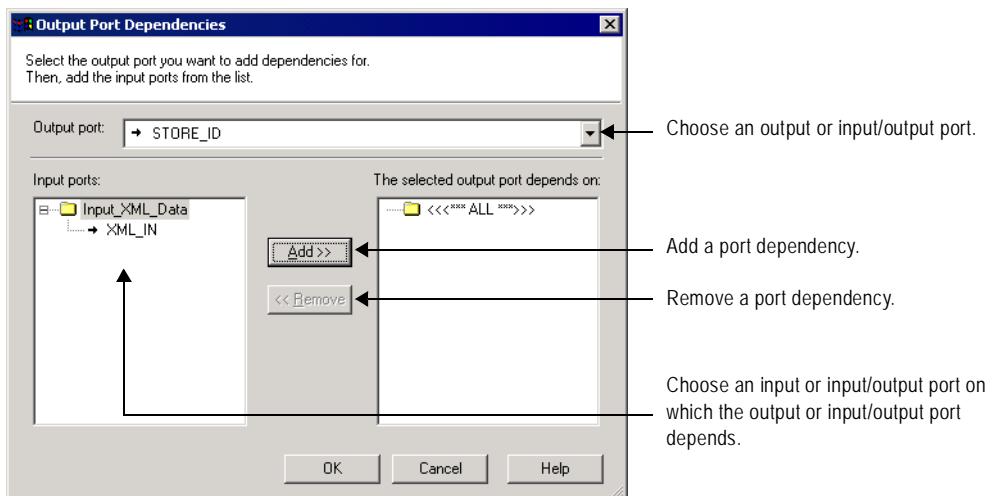
By default, an output port in a Custom transformation depends on all input ports. However, you can define the relationship between input and output ports in a Custom transformation. When you do this, you can view link paths in a mapping containing a Custom transformation and you can see which input ports an output port depends on. You can also view source column dependencies for target ports in a mapping containing a Custom transformation.

To define the relationship between ports in a Custom transformation, create a port dependency. A port dependency is the relationship between an output or input/output port and one or more input or input/output ports. When you create a port dependency, base it on the procedure logic in the code.

To create a port dependency, click Custom Transformation on the Ports tab and choose Port Dependencies.

Figure 3-2 shows where you create and edit port dependencies:

Figure 3-2. Editing Port Dependencies



For example, create a external procedure that parses XML data. You create a Custom transformation with one input group containing one input port and multiple output groups containing multiple output ports. According to the external procedure logic, all output ports depend on the input port. You can define this relationship in the Custom transformation by creating a port dependency for each output port. Define each port dependency so that the output port depends on the one input port.

To create a port dependency:

1. On the Ports tab, click Custom Transformation and choose Port Dependencies.
2. In the Output Port Dependencies dialog box, select an output or input/output port in the Output Port field.
3. In the Input Ports pane, select an input or input/output port on which the output port or input/output port depends.
4. Click Add.
5. Repeat steps 3 to 4 to include more input or input/output ports in the port dependency.
6. To create another port dependency, repeat steps 2 to 5.
7. Click OK.

Working with Port Attributes

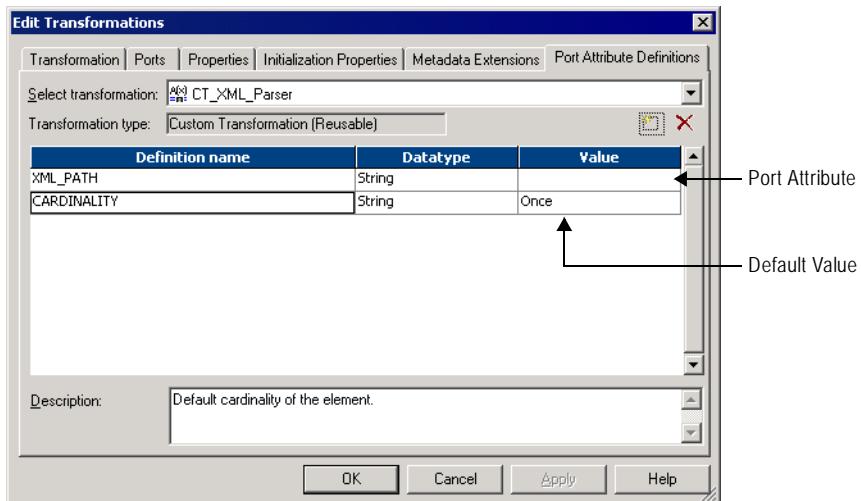
Ports have certain attributes, such as datatype and precision. When you create a Custom transformation, you can create user-defined port attributes. User-defined port attributes apply to all ports in a Custom transformation.

For example, you create an external procedure to parse XML data. You can create a port attribute called “XML path” where you can define the position of an element in the XML hierarchy.

Create port attributes and assign default values on the Port Attribute Definitions tab of the Custom transformation. You can define a specific port attribute value for each port on the Ports tab.

Figure 3-3 shows the Port Attribute Definitions tab where you create port attributes:

Figure 3-3. Port Attribute Definitions Tab



When you create a port attribute, define the following properties:

- ◆ **Name.** The name of the port attribute.
- ◆ **Datatype.** The datatype of the port attribute value. You can choose Boolean, Numeric, or String.
- ◆ **Value.** The default value of the port attribute. This property is optional. When you enter a value here, the value applies to all ports in the Custom transformation. You can override the port attribute value for each port on the Ports tab.

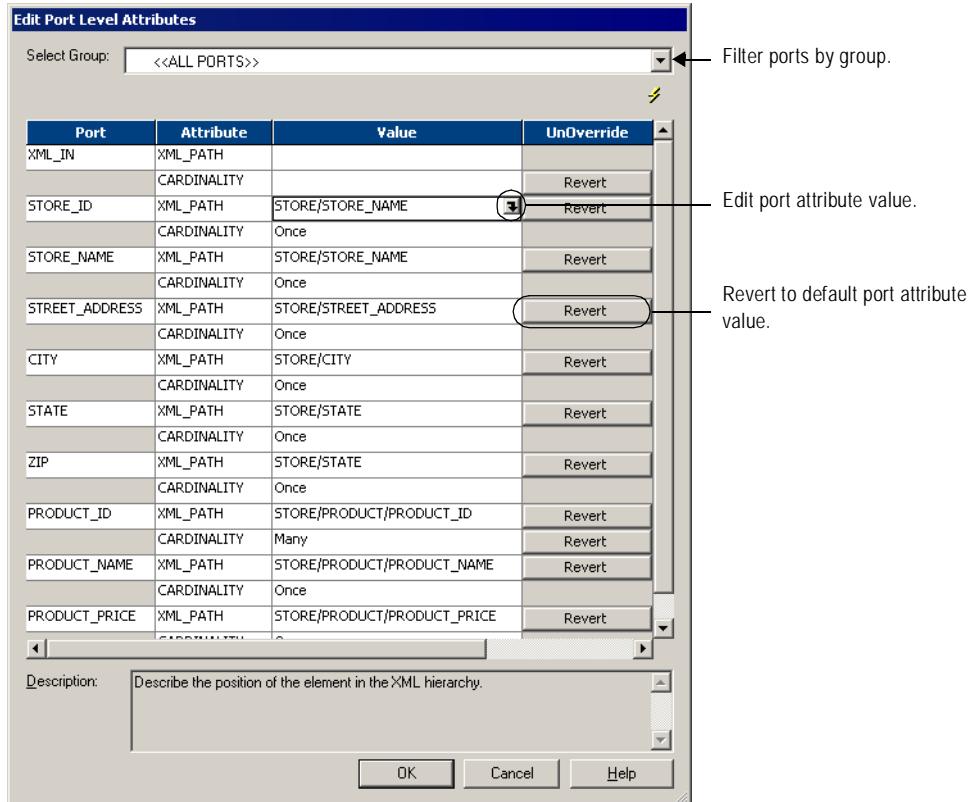
You define port attributes for each Custom transformation. You cannot copy a port attribute from one Custom transformation to another.

Editing Port Attribute Values

After you create port attributes, you can edit the port attribute values for each port in the transformation. To edit the port attribute values, click Custom Transformation on the Ports tab and choose Edit Port Attribute.

Figure 3-4 shows where you edit port attribute values:

Figure 3-4. Edit Port Attribute Values



You can change the port attribute value for a particular port by clicking the Open button. This opens the Edit Port Attribute Default Value dialog box. Or, you can enter a new value by typing directly in the Value column.

You can filter the ports listed in the Edit Port Level Attributes dialog box by choosing a group from the Select Group field.

Custom Transformation Properties

Properties for the Custom transformation apply to both the procedure and the transformation. Configure the Custom transformation properties on the Properties tab of the Custom transformation.

Figure 3-5 shows the Custom transformation Properties tab:

Figure 3-5. Custom Transformation Properties

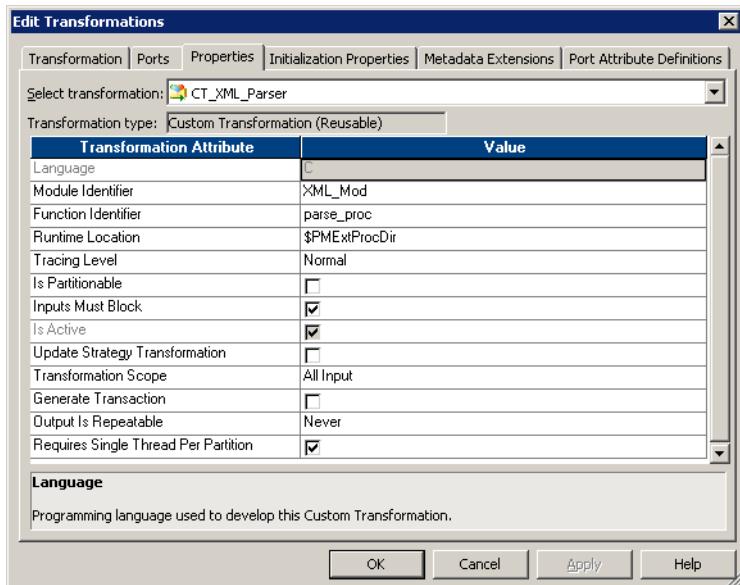


Table 3-1 describes the Custom transformation properties:

Table 3-1. Custom Transformation Properties

Option	Description
Language	Language used for the procedure code. You define the language when you create the Custom transformation. If you need to change the language, create a new Custom transformation.
Module Identifier	Module name. Applies to Custom transformation procedures developed using C or C++. Enter only ASCII characters in this field. You cannot enter multibyte characters. This property is the base name of the DLL or the shared library that contains the procedure. The Designer uses this name to create the C file when you generate the external procedure code.
Function Identifier	Name of the procedure in the module. Applies to Custom transformation procedures developed using C. Enter only ASCII characters in this field. You cannot enter multibyte characters. The Designer uses this name to create the C file where you enter the procedure code.

Table 3-1. Custom Transformation Properties

Option	Description
Class Name	Class name of the Custom transformation procedure. Applies to Custom transformation procedures developed using C++ or Java. Enter only ASCII characters in this field. You cannot enter multibyte characters.
Runtime Location	Location that contains the DLL or shared library. Default is \$PMExtProcDir. Enter a path relative to the Integration Service machine that runs the session using the Custom transformation. If you make this property blank, the Integration Service uses the environment variable defined on the Integration Service machine to locate the DLL or shared library. You must copy all DLLs or shared libraries to the runtime location or to the environment variable defined on the Integration Service machine. The Integration Service fails to load the procedure when it cannot locate the DLL, shared library, or a referenced file.
Tracing Level	Amount of detail displayed in the session log for this transformation. Default is Normal.
Is Partitionable	Indicates if you can create multiple partitions in a pipeline that uses this transformation: <ul style="list-style-type: none">- No. The transformation cannot be partitioned. The transformation and other transformations in the same pipeline are limited to one partition.- Locally. The transformation can be partitioned, but the Integration Service must run all partitions in the pipeline on the same node. Choose Local when different partitions of the Custom transformation must share objects in memory.- Across Grid. The transformation can be partitioned, and the Integration Service can distribute each partition to different nodes. Default is No. For more information about using partitioning with Custom transformations, see "Working with Partition Points" in the <i>Workflow Administration Guide</i> .
Inputs Must Block	Indicates if the procedure associated with the transformation must be able to block incoming data. Default is enabled. For more information about blocking input data, see "Blocking Input Data" on page 70.
Is Active	Indicates if this transformation is an active or passive transformation. You cannot change this property after you create the Custom transformation. If you need to change this property, create a new Custom transformation and select the correct property value.
Update Strategy Transformation	Indicates if this transformation defines the update strategy for output rows. Default is disabled. You can enable this for active Custom transformations. For more information about this property, see "Setting the Update Strategy" on page 66.
Transformation Scope	Indicates how the Integration Service applies the transformation logic to incoming data: <ul style="list-style-type: none">- Row- Transaction- All Input When the transformation is passive, this property is always Row. When the transformation is active, this property is All Input by default. For more information about working with transaction control, see "Working with Transaction Control" on page 68.
Generate Transaction	Indicates if this transformation can generate transactions. When a Custom transformation generates transactions, it generates transactions for all output groups. Default is disabled. You can only enable this for active Custom transformations. For more information about working with transaction control, see "Working with Transaction Control" on page 68.

Table 3-1. Custom Transformation Properties

Option	Description
Output is Ordered	Indicates if the order of the output data is consistent between session runs. <ul style="list-style-type: none">- Never. The order of the output data is inconsistent between session runs. This is the default for active transformations.- Based On Input Order. The output order is consistent between session runs when the input data order is consistent between session runs. This is the default for passive transformations.- Always. The order of the output data is consistent between session runs even if the order of the input data is inconsistent between session runs.
Requires Single Thread Per Partition	Indicates if the Integration Service processes each partition at the procedure with one thread. When you enable this option, the procedure code can use thread-specific operations. Default is enabled. For more information about writing thread-specific operations, see "Working with Thread-Specific Procedure Code" on page 66.
Output is Deterministic	Indicates whether the transformation generates consistent output data between session runs. You must enable this property to perform recovery on sessions that use this transformation. For more information about session recovery, see "Recovering Workflows" in the <i>Workflow Administration Guide</i> .

Setting the Update Strategy

Use an active Custom transformation to set the update strategy for a mapping at the following levels:

- ◆ **Within the procedure.** You can write the external procedure code to set the update strategy for output rows. The external procedure can flag rows for insert, update, delete, or reject. For more information about the functions used to set the update strategy, see "Row Strategy Functions (Row-Based Mode)" on page 128.
- ◆ **Within the mapping.** Use the Custom transformation in a mapping to flag rows for insert, update, delete, or reject. Select the Update Strategy Transformation property for the Custom transformation.
- ◆ **Within the session.** Configure the session to treat the source rows as data driven.

If you do not configure the Custom transformation to define the update strategy, or you do not configure the session as data driven, the Integration Service does not use the external procedure code to flag the output rows. Instead, when the Custom transformation is active, the Integration Service flags the output rows as insert. When the Custom transformation is passive, the Integration Service retains the row type. For example, when a row flagged for update enters a passive Custom transformation, the Integration Service maintains the row type and outputs the row as update.

Working with Thread-Specific Procedure Code

Custom transformation procedures can include thread-specific operations. A thread-specific operation is code that performs an action based on the thread that is processing the procedure.

You can configure the Custom transformation so the Integration Service uses one thread to process the Custom transformation for each partition using the Requires Single Thread Per Partition property.

When you configure a Custom transformation to process each partition with one thread, the Integration Service calls the following functions with the same thread for each partition:

- ◆ p_<proc_name>_partitionInit()
- ◆ p_<proc_name>_partitionDeinit()
- ◆ p_<proc_name>_inputRowNotification()
- ◆ p_<proc_name>_dataBdryRowNotification()
- ◆ p_<proc_name>_eofNotification()

You can include thread-specific operations in these functions because the Integration Service uses the same thread to process these functions for each partition. For example, you might attach and detach threads to a Java Virtual Machine.

Note: When you configure a Custom transformation to process each partition with one thread, the Workflow Manager adds partition points depending on the mapping configuration. For more information, see “Working with Partition Points” in the *Workflow Administration Guide*.

Working with Transaction Control

You can define transaction control for Custom transformations using the following transformation properties:

- ◆ **Transformation Scope.** Determines how the Integration Service applies the transformation logic to incoming data.
- ◆ **Generate Transaction.** Indicates that the procedure generates transaction rows and outputs them to the output groups.

Transformation Scope

You can configure how the Integration Service applies the transformation logic to incoming data. You can choose one of the following values:

- ◆ **Row.** Applies the transformation logic to one row of data at a time. Choose Row when the results of the procedure depend on a single row of data. For example, you might choose Row when a procedure parses a row containing an XML file.
- ◆ **Transaction.** Applies the transformation logic to all rows in a transaction. Choose Transaction when the results of the procedure depend on all rows in the same transaction, but not on rows in other transactions. When you choose Transaction, you must connect all input groups to the same transaction control point. For example, you might choose Transaction when the external procedure performs aggregate calculations on the data in a single transaction.
- ◆ **All Input.** Applies the transformation logic to all incoming data. When you choose All Input, the Integration Service drops transaction boundaries. Choose All Input when the results of the procedure depend on all rows of data in the source. For example, you might choose All Input when the external procedure performs aggregate calculations on all incoming data, or when it sorts all incoming data.

For more information about transformation scope, see “Understanding Commit Points” in the *Workflow Administration Guide*.

Generate Transaction

You can write the external procedure code to output transactions, such as commit and rollback rows. When the external procedure outputs commit and rollback rows, configure the Custom transformation to generate transactions. Select the Generate Transaction transformation property. You can enable this property for active Custom transformations. For information about the functions you use to generate transactions, see “Data Boundary Output Notification Function” on page 121.

When the external procedure outputs a commit or rollback row, it outputs or rolls back the row for all output groups.

When you configure the transformation to generate transactions, the Integration Service treats the Custom transformation like a Transaction Control transformation. Most rules that apply to a Transaction Control transformation in a mapping also apply to the Custom

transformation. For example, when you configure a Custom transformation to generate transactions, you cannot concatenate pipelines or pipeline branches containing the transformation. For more information about working with Transaction Control transformations, see “Transaction Control Transformation” on page 519.

When you edit or create a session using a Custom transformation configured to generate transactions, configure it for user-defined commit.

Working with Transaction Boundaries

The Integration Service handles transaction boundaries entering and leaving Custom transformations based on the mapping configuration and the Custom transformation properties.

Table 3-2 describes how the Integration Service handles transaction boundaries at Custom transformations:

Table 3-2. Transaction Boundary Handling with Custom Transformations

Transformation Scope	Generate Transactions Enabled	Generate Transactions Disabled
Row	Integration Service drops incoming transaction boundaries and does not call the data boundary notification function. It outputs transaction rows according to the procedure logic across all output groups.	When the incoming data for all input groups comes from the same transaction control point, the Integration Service preserves incoming transaction boundaries and outputs them across all output groups. However, it does not call the data boundary notification function.
Transaction	Integration Service preserves incoming transaction boundaries and calls the data boundary notification function. However, it outputs transaction rows according to the procedure logic across all output groups.	When the incoming data for the input groups comes from different transaction control points, the Integration Service drops incoming transaction boundaries. It does not call the data boundary notification function. The Integration Service outputs all rows in one open transaction.
All Input	Integration Service drops incoming transaction boundaries and does not call the data boundary notification function. The Integration Service outputs transaction rows according to the procedure logic across all output groups.	Integration Service preserves incoming transaction boundaries and calls the data boundary notification function. It outputs the transaction rows across all output groups.

Blocking Input Data

By default, the Integration Service concurrently reads sources in a target load order group. However, you can write the external procedure code to block input data on some input groups. Blocking is the suspension of the data flow into an input group of a multiple input group transformation. For more information about blocking source data, see “Integration Service Architecture” in the *Administrator Guide*.

To use a Custom transformation to block input data, you must write the procedure code to block and unblock data. You must also enable blocking on the Properties tab for the Custom transformation.

Writing the Procedure Code to Block Data

You can write the procedure to block and unblock incoming data. To block incoming data, use the INFA_CTBLOCKINPUTFLOW() function. To unblock incoming data, use the INFA_CTNBLOCKINPUTFLOW() function. For more information about the blocking functions, see “Blocking Functions” on page 125.

You might want to block input data if the external procedure needs to alternate reading from input groups. Without the blocking functionality, you would need to write the procedure code to buffer incoming data. You can block input data instead of buffering it which usually increases session performance.

For example, you need to create an external procedure with two input groups. The external procedure reads a row from the first input group and then reads a row from the second input group. If you use blocking, you can write the external procedure code to block the flow of data from one input group while it processes the data from the other input group. When you write the external procedure code to block data, you increase performance because the procedure does not need to copy the source data to a buffer. However, you could write the external procedure to allocate a buffer and copy the data from one input group to the buffer until it is ready to process the data. Copying source data to a buffer decreases performance.

Configuring Custom Transformations as Blocking Transformations

When you create a Custom transformation, the Designer enables the Inputs Must Block transformation property by default. This property affects data flow validation when you save or validate a mapping. When you enable this property, the Custom transformation is a blocking transformation. When you clear this property, the Custom transformation is not a blocking transformation. For more information about blocking transformations, see “Multi-Group Transformations” on page 9.

Configure the Custom transformation as a blocking transformation when the external procedure code *must* be able to block input data.

You can configure the Custom transformation as a non-blocking transformation when one of the following conditions is true:

- ♦ The procedure code does not include the blocking functions.

- ◆ The procedure code includes two algorithms, one that uses blocking and the other that copies the source data to a buffer allocated by the procedure instead of blocking data. The code checks whether or not the Integration Service allows the Custom transformation to block data. The procedure uses the algorithm with the blocking functions when it can block, and uses the other algorithm when it cannot block. You might want to do this to create a Custom transformation that you use in multiple mapping configurations.

For more information about verifying whether the Integration Service allows a Custom transformation to block data, see “[Validating Mappings with Custom Transformations](#)” on page 71.

Note: When the procedure blocks data and you configure the Custom transformation as a non-blocking transformation, the Integration Service fails the session.

Validating Mappings with Custom Transformations

When you include a Custom transformation in a mapping, both the Designer and Integration Service validate the mapping. The Designer validates the mapping you save or validate and the Integration Service validates the mapping when you run the session.

Validating at Design Time

When you save or validate a mapping, the Designer performs data flow validation. When the Designer does this, it verifies that the data can flow from all sources in a target load order group to the targets without blocking transformations blocking *all* sources. Some mappings with blocking transformations are invalid. For more information about data flow validation, see “[Mappings](#)” in the *Designer Guide*.

Validating at Runtime

When you run a session, the Integration Service validates the mapping against the procedure code at runtime. When the Integration Service does this, it tracks whether or not it allows the Custom transformations to block data:

- ◆ **Configure the Custom transformation as a blocking transformation.** The Integration Service always allows the Custom transformation to block data.
- ◆ **Configure the Custom transformation as a non-blocking transformation.** The Integration Service allows the Custom transformation to block data depending on the mapping configuration. If the Integration Service can block data at the Custom transformation without blocking all sources in the target load order group simultaneously, it allows the Custom transformation to block data.

You can write the procedure code to check whether or not the Integration Service allows a Custom transformation to block data. Use the INFA_CT_getInternalProperty() function to access the INFA_CT_TRANS_MAY_BLOCK_DATA property ID. The Integration Service returns TRUE when the Custom transformation can block data, and it returns FALSE when the Custom transformation cannot block data. For more information about the INFA_CT_getInternalProperty() function, see “[Property Functions](#)” on page 108.

Working with Procedure Properties

You can define property name and value pairs in the Custom transformation that the procedure can use when the Integration Service runs the procedure, such as during initialization time. You can create user-defined properties on the following tabs of the Custom transformation:

- ◆ **Metadata Extensions.** You can specify the property name, datatype, precision, and value. Use metadata extensions for passing information to the procedure. For more information about creating metadata extensions, see “Metadata Extensions” in the *Repository Guide*.
- ◆ **Initialization Properties.** You can specify the property name and value.

While you can define properties on both tabs in the Custom transformation, the Metadata Extensions tab lets you provide more detail for the property. Use metadata extensions to pass properties to the procedure.

For example, you create a Custom transformation external procedure that sorts data after transforming it. You could create a boolean metadata extension named Sort_Ascending. When you use the Custom transformation in a mapping, you can choose True or False for the metadata extension, depending on how you want the procedure to sort the data.

When you define a property in the Custom transformation, use the get all property names functions, such as INFA_CTGetAllPropertyNamesM(), to access the names of all properties defined on the Initialization Properties and Metadata Extensions tab. Use the get external property functions, such as INFA_CT_getExternalPropertyM(), to access the property name and value of a property ID you specify.

Note: When you define a metadata extension and an initialization property with the same name, the property functions only return information for the metadata extension.

Creating Custom Transformation Procedures

You can create Custom transformation procedures that run on 32-bit or 64-bit Integration Service machines. Use the following steps as a guideline when you create a Custom transformation procedure:

1. In the Transformation Developer, create a reusable Custom transformation. Or, in the Mapplet Designer or Mapping Designer, create a non-reusable Custom transformation.
2. Generate the template code for the procedure.

When you generate the procedure code, the Designer uses the information from the Custom transformation to create C source code files and makefiles.

3. Modify the C files to add the procedure logic.
4. Use a C/C++ compiler to compile and link the source code files into a DLL or shared library and copy it to the Integration Service machine.
5. Create a mapping with the Custom transformation.
6. Run the session in a workflow.

This section includes an example, the “Union example,” to demonstrate this process. The steps in this section create a Custom transformation that contains two input groups and one output group. The Custom transformation procedure verifies that the Custom transformation uses two input groups and one output group. It also verifies that the number of ports in all groups are equal and that the port datatypes are the same for all groups. The procedure takes rows of data from each input group and outputs all rows to the output group.

Step 1. Create the Custom Transformation

The first step is to create a Custom transformation.

To create a Custom transformation:

1. In the Transformation Developer, click Transformation > Create.
2. In the Create Transformation dialog box, choose Custom transformation, enter a transformation name, and click Create.

In the Union example, enter `CT_Inf_Union` as the transformation name.

3. In the Active or Passive dialog box, create the transformation as a passive or active transformation, and click OK.

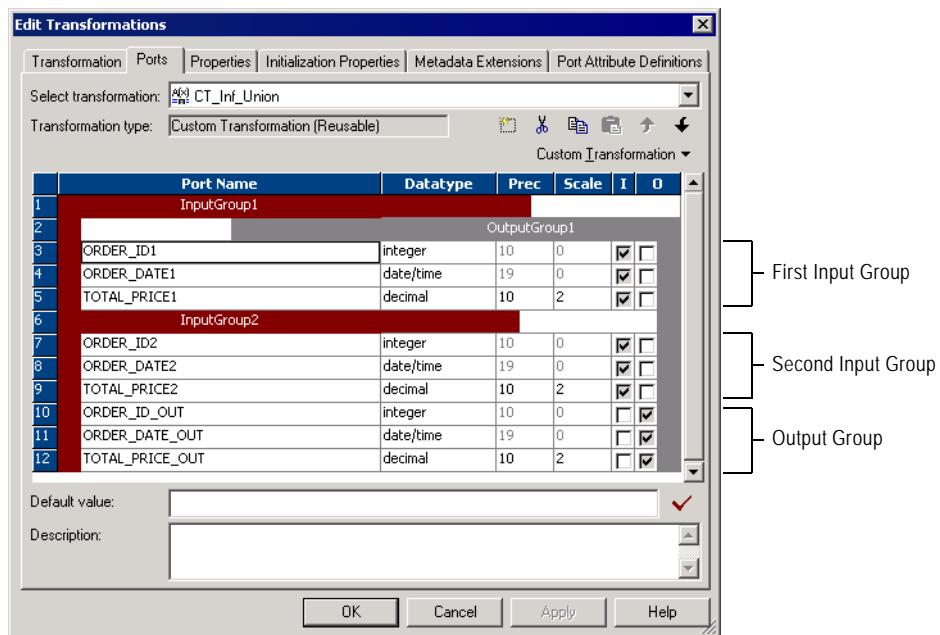
In the Union example, choose Active.

4. Click Done to close the Create Transformation dialog box.
5. Open the transformation and click the Ports tab. Create groups and ports.

You can edit the groups and ports later, if necessary. For more information about creating groups and ports, see “Working with Groups and Ports” on page 59.

In the Union example, create the groups and ports shown in Figure 3-6:

Figure 3-6. Custom Transformation Ports Tab - Union Example

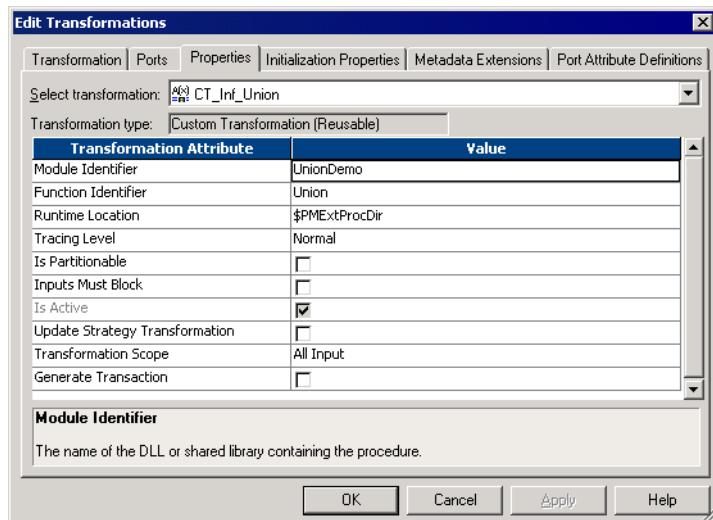


6. Select the Properties tab and enter a module and function identifier and the runtime location. Edit other transformation properties.

For more information about Custom transformation properties, see “Custom Transformation Properties” on page 64.

In the Union example, enter the properties shown in Figure 3-7:

Figure 3-7. Custom Transformation Properties Tab - Union Example



7. Click the Metadata Extensions tab to enter metadata extensions, such as properties the external procedure might need for initialization. For more information about using metadata extensions for procedure properties, see “Working with Procedure Properties” on page 72.

In the Union example, do not create metadata extensions.

8. Click the Port Attribute Definitions tab to create port attributes, if necessary. For more information about creating port attributes, see “Working with Port Attributes” on page 62.

In the Union example, do not create port attributes.

9. Click OK.

10. Click Repository > Save.

After you create the Custom transformation that calls the procedure, the next step is to generate the C files.

Step 2. Generate the C Files

After you create a Custom transformation, you generate the source code files. The Designer generates file names in lower case.

To generate the code for a Custom transformation procedure:

1. In the Transformation Developer, select the transformation and click Transformation > Generate Code.
2. Select the procedure you just created. The Designer lists the procedures as <module_name>.<procedure_name>.

In the Union example, select UnionDemo.Union.

3. Specify the directory where you want to generate the files, and click Generate.

In the Union example, select <client_installation_directory>/TX.

The Designer creates a subdirectory, <module_name>, in the directory you specified. In the Union example, the Designer creates <client_installation_directory>/TX/UnionDemo. It also creates the following files:

- ◆ m_UnionDemo.c
- ◆ m_UnionDemo.h
- ◆ p_Union.c
- ◆ p_Union.h
- ◆ makefile.aix (32-bit), makefile.aix64 (64-bit), makefile.hp (32-bit), makefile.hp64 (64-bit), makefile.hpparisc64, makefile.linux (32-bit), and makefile.sol (32-bit).

Step 3. Fill Out the Code with the Transformation Logic

You must code the procedure C file. Optionally, you can also code the module C file. In the Union example, you fill out the procedure C file only. You do not need to fill out the module C file.

To code the procedure C file:

1. Open p-<procedure_name>.c for the procedure.

In the Union example, open p_Union.c.

2. Enter the C code for the procedure.

3. Save the modified file.

In the Union example, use the following code:

```
*****
*
* Copyright (c) 2005 Informatica Corporation. This file contains
* material proprietary to Informatica Corporation and may not be copied
* or distributed in any form without the written permission of Informatica
* Corporation
*
*****
```

```

*****
* Custom Transformation p_union Procedure File
*
* This file contains code that functions that will be called by the main
* server executable.
*
* for more information on these files,
* see $(INFA_HOME)/ExtProc/include/Readme.txt
*****
*/

/*
* INFORMATICA 'UNION DEMO' developed using the API for custom
* transformations.

* File Name: p_Union.c
*
* An example of a custom transformation ('Union') using PowerCenter8.0
*
* The purpose of the 'Union' transformation is to combine pipelines with the
* same row definition into one pipeline (i.e. union of multiple pipelines).
* [ Note that it does not correspond to the mathematical definition of union
* since it does not eliminate duplicate rows.]
*
* This example union transformation allows N input pipelines ( each
* corresponding to an input group) to be combined into one pipeline.
*
* To use this transformation in a mapping, the following attributes must be
* true:
* a. The transformation must have >= 2 input groups and only one output group.
* b. In the Properties tab set the following properties:
*     i. Module Identifier: UnionDemo
*     ii. Function Identifier: Union
*     iii. Inputs May Block: Unchecked
*     iv. Is Active: Checked
*     v. Update Strategy Transformation: Unchecked *
*     vi. Transformation Scope: All
*     vii. Generate Transaction: Unchecked *
*
*     * This version of the union transformation does not provide code for
*     changing the update strategy or for generating transactions.
* c. The input groups and the output group must have the same number of ports
*     and the same datatypes. This is verified in the initialization of the
*     module and the session is failed if this is not true.
* d. The transformation can be used in multiple number of times in a Target

```

```

*      Load Order Group and can also be contained within multiple partitions.
*
*/

```

```

/***** Includes *****/

```

```

#include <stdlib.h>
#include "p_union.h"

```

```

/***** Forward Declarations *****/

```

```

INFA_STATUS validateProperties(const INFA_CT_PARTITION_HANDLE* partition);

```

```

/***** Functions *****/

```

```

Function: p_union_procInit

Description: Initialization for the procedure. Returns INFA_SUCCESS if
procedure initialization succeeds, else return INFA_FAILURE.

Input: procedure - the handle for the procedure
Output: None
Remarks: This function will get called once for the session at
initialization time. It will be called after the moduleInit function.

```

```

INFA_STATUS p_union_procInit( INFA_CT_PROCEDURE_HANDLE procedure)
{
    const INFA_CT_TRANSFORMATION_HANDLE* transformation = NULL;
    const INFA_CT_PARTITION_HANDLE* partition = NULL;
    size_t nTransformations = 0, nPartitions = 0, i = 0;

    /* Log a message indicating beginning of the procedure initialization */
    INFA_CTLLogMessageM( eESL_LOG,
                        "union_demo: Procedure initialization started ..." );

```

```

    INFA_CTChangeStringMode( procedure, eASM_MBCS );

```

```

/* Get the transformation handles */
transformation = INFA_CTGetChildrenHandles( procedure,
                                             &nTransformations,
                                             TRANSFORMATIONTYPE);

/* For each transformation verify that the 0th partition has the correct
 * properties. This does not need to be done for all partitions since rest
 * of the partitions have the same information */
for (i = 0; i < nTransformations; i++)
{
    /* Get the partition handle */
    partition = INFA_CTGetChildrenHandles(transformation[i],
                                           &nPartitions, PARTITIONTYPE );

    if (validateProperties(partition) != INFA_SUCCESS)
    {
        INFA_CTLLogMessageM( eESL_ERROR,
                            "union_demo: Failed to validate attributes of "
                            "the transformation");
        return INFA_FAILURE;
    }
}

INFA_CTLLogMessageM( eESL_LOG,
                     "union_demo: Procedure initialization completed." );

return INFA_SUCCESS;
}

*****
Function: p_union_procDeinit

Description: Deinitialization for the procedure. Returns INFA_SUCCESS if
procedure deinitialization succeeds, else return INFA_FAILURE.

Input: procedure - the handle for the procedure
Output: None
Remarks: This function will get called once for the session at
deinitialization time. It will be called before the moduleDeinit
function.
***** */

INFA_STATUS p_union_procDeinit( INFA_CT_PROCEDURE_HANDLE procedure,
                               INFA_STATUS sessionStatus )

```

```

{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

*****
Function: p_union_partitionInit

Description: Initialization for the partition. Returns INFA_SUCCESS if
partition deinitialization succeeds, else return INFA_FAILURE.

Input: partition - the handle for the partition
Output: None
Remarks: This function will get called once for each partition for each
transformation in the session.
*****/




INFA_STATUS p_union_partitionInit( INFA_CT_PARTITION_HANDLE partition )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

*****
Function: p_union_partitionDeinit

Description: Deinitialization for the partition. Returns INFA_SUCCESS if
partition deinitialization succeeds, else return INFA_FAILURE.

Input: partition - the handle for the partition
Output: None
Remarks: This function will get called once for each partition for each
transformation in the session.
*****/




INFA_STATUS p_union_partitionDeinit( INFA_CT_PARTITION_HANDLE partition )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

*****
Function: p_union_inputRowNotification

```

Description: Notification that a row needs to be processed for an input group in a transformation for the given partition. Returns INFA_ROWSUCCESS if the input row was processed successfully, INFA_ROWFAILURE if the input row was not processed successfully and INFA_FATALError if the input row causes the session to fail.

Input: partition - the handle for the partition for the given row
group - the handle for the input group for the given row

Output: None

Remarks: This function is probably where the meat of your code will go, as it is called for every row that gets sent into your transformation.

```
INFA_ROWSTATUS p_union_inputRowNotification( INFA_CT_PARTITION_HANDLE partition,
                                             INFA_CT_INPUTGROUP_HANDLE inputGroup )

{

    const INFA_CT_OUTPUTGROUP_HANDLE* outputGroups = NULL;
    const INFA_CT_INPUTPORT_HANDLE* inputGroupPorts = NULL;
    const INFA_CT_OUTPUTPORT_HANDLE* outputGroupPorts = NULL;
    size_t nNumInputPorts = 0, nNumOutputGroups = 0,
           nNumPortsInOutputGroup = 0, i = 0;

    /* Get the output group port handles */
    outputGroups = INFA_CTGetChildrenHandles(partition,
                                              &nNumOutputGroups,
                                              OUTPUTGROUPTYPE);

    outputGroupPorts = INFA_CTGetChildrenHandles(outputGroups[0],
                                                &nNumPortsInOutputGroup,
                                                OUTPUTPORTTYPE);

    /* Get the input groups port handles */
    inputGroupPorts = INFA_CTGetChildrenHandles(inputGroup,
                                                &nNumInputPorts,
                                                INPUTPORTTYPE);

    /* For the union transformation, on receiving a row of input, we need to
     * output that row on the output group. */
    for (i = 0; i < nNumInputPorts; i++)
    {
        INFA_CTSGetData(outputGroupPorts[i],
                        INFA_CTCGetDataVoid(inputGroupPorts[i]));
    }
}
```

```

    INFA_CTSetsIndicator(outputGroupPorts[i],
        INFA_CTGetsIndicator(inputGroupPorts[i]) );

    INFA_CTSetsLength(outputGroupPorts[i],
        INFA_CTGetsLength(inputGroupPorts[i]) );
}

/* We know there is only one output group for each partition */
return INFA_CTOOutputNotification(outputGroups[0]);
}

```

Function: p_union_eofNotification

Description: Notification that the last row for an input group has already been seen. Return INFA_FAILURE if the session should fail as a result of seeing this notification, INFA_SUCCESS otherwise.

Input: partition - the handle for the partition for the notification
group - the handle for the input group for the notification

Output: None

```

INFA_STATUS p_union_eofNotification( INFA_CT_PARTITION_HANDLE partition,
                                     INFA_CT_INPUTGROUP_HANDLE group)
{
    INFA_CTLLogMessageM( eESL_LOG,
        "union_demo: An input group received an EOF notification");

    return INFA_SUCCESS;
}

```

Function: p_union_dataBdryNotification

Description: Notification that a transaction has ended. The data boundary type can either be commit or rollback.

Return INFA_FAILURE if the session should fail as a result of seeing this notification, INFA_SUCCESS otherwise.

Input: partition - the handle for the partition for the notification
transactionType - commit or rollback

Output: None

```

INFA_STATUS p_union_dataBdryNotification ( INFA_CT_PARTITION_HANDLE partition,
                                         INFA_CT_DATABDRY_TYPE transactionType)
{
    /* Do nothing */
    return INFA_SUCCESS;
}

/* Helper functions */

/**************************************************************************
Function: validateProperties

Description: Validate that the transformation has all properties expected
by a union transformation, such as at least one input group, and only
one output group. Return INFA_FAILURE if the session should fail since the
transformation was invalid, INFA_SUCCESS otherwise.

Input: partition - the handle for the partition
Output: None
**************************************************************************/

INFA_STATUS validateProperties(const INFA_CT_PARTITION_HANDLE* partition)
{
    const INFA_CT_INPUTGROUP_HANDLE* inputGroups = NULL;
    const INFA_CT_OUTPUTGROUP_HANDLE* outputGroups = NULL;
    size_t nNumInputGroups = 0, nNumOutputGroups = 0;
    const INFA_CT_INPUTPORT_HANDLE** allInputGroupsPorts = NULL;
    const INFA_CT_OUTPUTPORT_HANDLE* outputGroupPorts = NULL;
    size_t nNumPortsInOutputGroup = 0;
    size_t i = 0, nTempNumInputPorts = 0;

    /* Get the input and output group handles */
    inputGroups = INFA_CTCGetChildrenHandles(partition[0],
                                             &nNumInputGroups,
                                             INPUTGROUPTYPE);

    outputGroups = INFA_CTCGetChildrenHandles(partition[0],
                                              &nNumOutputGroups,
                                              OUTPUTGROUPTYPE);

    /* 1. Number of input groups must be >= 2 and number of output groups must
     *      be equal to one. */
    if (nNumInputGroups < 1 || nNumOutputGroups != 1)

```

```

{
    INFA_CTLLogMessageM( eESL_ERROR,
        "UnionDemo: There must be at least two input groups "
        "and only one output group");
    return INFA_FAILURE;
}

/* 2. Verify that the same number of ports are in each group (including
 * output group). */
outputGroupPorts = INFA_CTGetChildrenHandles(outputGroups[0],
                                             &nNumPortsInOutputGroup,
                                             OUTPUTPORTTYPE);

/* Allocate an array for all input groups ports */
allInputGroupsPorts = malloc(sizeof(INFA_CT_INPUTPORT_HANDLE*) *
                             nNumInputGroups);

for (i = 0; i < nNumInputGroups; i++)
{
    allInputGroupsPorts[i] = INFA_CTGetChildrenHandles(inputGroups[i],
                                                       &nTempNumInputPorts,
                                                       INPUTPORTTYPE);

    if ( nNumPortsInOutputGroup != nTempNumInputPorts)
    {
        INFA_CTLLogMessageM( eESL_ERROR,
            "UnionDemo: The number of ports in all input and "
            "the output group must be the same.");
        return INFA_FAILURE;
    }
}

free(allInputGroupsPorts);

/* 3. Datatypes of ports in input group 1 must match data types of all other
 *     groups.
TODO:*/
return INFA_SUCCESS;
}

```

Step 4. Build the Module

You can build the module on a Windows or UNIX platform.

Table 3-3 lists the library file names for each platform when you build the module:

Table 3-3. Module File Names

Platform	Module File Name
Windows	<module_identifier>.dll
AIX	lib<module_identifier>.a
HP-UX	lib<module_identifier>.sl
Linux	lib<module_identifier>.so
Solaris	lib<module_identifier>.so

Building the Module on Windows

On Windows, use Microsoft Visual C++ to build the module.

To build the module on Windows:

- 1.** Start Visual C++.
- 2.** Click File > New.
- 3.** In the New dialog box, click the Projects tab and select the Win32 Dynamic-Link Library option.
- 4.** Enter its location.

In the Union example, enter <client_installation_directory>/TX/UnionDemo.

- 5.** Enter the name of the project.

You must use the module name specified for the Custom transformation as the project name. In the Union example, enter UnionDemo.

- 6.** Click OK.

Visual C++ creates a wizard to help you define the project components.

- 7.** In the wizard, select An empty DLL project and click Finish. Click OK in the New Project Information dialog box.

Visual C++ creates the project files in the directory you specified.

- 8.** Click Project > Add To Project > Files.

- 9.** Navigate up a directory level. This directory contains the procedure files you created. Select all .c files and click OK.

In the Union example, add the following files:

- ◆ m_UnionDemo.c
- ◆ p_Union.c

- 10.** Click Project > Settings.

- 11.** Click the C/C++ tab, and select Preprocessor from the Category field.

- 12.** In the Additional Include Directories field, enter the following path and click OK:

...; <PowerCenter_install_dir>\extproc\include\ct

- 13.** Click Build > Build <module_name>.dll or press F7 to build the project.

Visual C++ creates the DLL and places it in the debug or release directory under the project directory.

Building the Module on UNIX

On UNIX, use any C compiler to build the module.

To build the module on UNIX:

- 1.** Copy all C files and makefiles generated by the Designer to the UNIX machine.

Note: If you build the shared library on a machine other than the Integration Service machine, you must also copy the files in the following directory to the build machine:

<PowerCenter_install_dir>\ExtProc\include\ct

In the Union example, copy all files in <client_installation_directory>/TX/UnionDemo.

- 2.** Set the environment variable INFA_HOME to the Integration Service installation directory.

Note: If you specify an incorrect directory path for the INFA_HOME environment variable, the Integration Service cannot start.

- 3.** Enter a command from Table 3-4 to make the project.

Table 3-4. UNIX Commands to Build the Shared Library

UNIX Version	Command
AIX (32-bit)	make -f makefile.aix
AIX (64-bit)	make -f makefile.aix64
HP-UX (32-bit)	make -f makefile.hp
HP-UX (64-bit)	make -f makefile.hp64
HP-UX PA-RISC	make -f makefile.hpparisc64

Table 3-4. UNIX Commands to Build the Shared Library

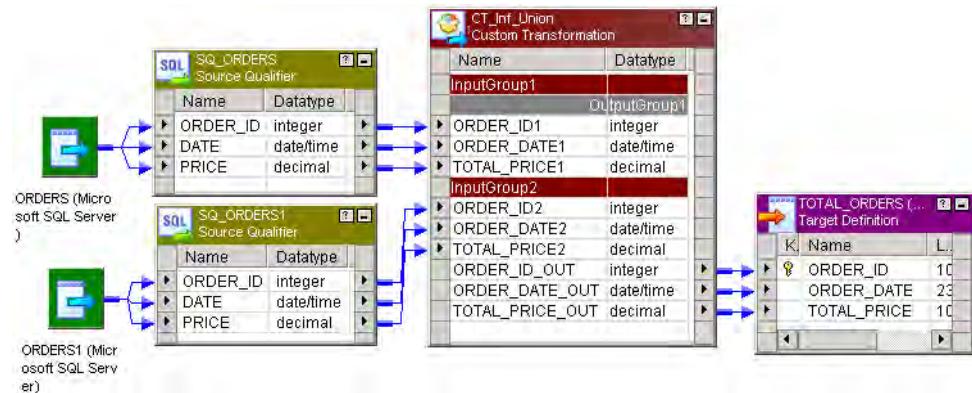
UNIX Version	Command
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

Step 5. Create a Mapping

In the Mapping Designer, create a mapping that uses the Custom transformation.

In the Union example, create a mapping similar to the one in Figure 3-8:

Figure 3-8. Mapping with a Custom Transformation - Union Example



In this mapping, two sources with the same ports and datatypes connect to the two input groups in the Custom transformation. The Custom transformation takes the rows from both sources and outputs them all through its one output group. The output group has the same ports and datatypes as the input groups.

Step 6. Run the Session in a Workflow

When you run the session, the Integration Service looks for the shared library or DLL in the runtime location you specify in the Custom transformation.

To run a session in a workflow:

1. In the Workflow Manager, create a workflow.
2. Create a session for this mapping in the workflow.
3. Copy the shared library or DLL to the runtime location directory.
4. Run the workflow containing the session.

When the Integration Service loads a Custom transformation bound to a procedure, it loads the DLL or shared library and calls the procedure you define.

Chapter 4

Custom Transformation Functions

This chapter includes the following topics:

- ◆ Overview, 90
- ◆ Function Reference, 92
- ◆ Working with Rows, 96
- ◆ Generated Functions, 98
- ◆ API Functions, 104
- ◆ Array-Based API Functions, 130
- ◆ Java API Functions, 138
- ◆ C++ API Functions, 139

Overview

Custom transformations operate in conjunction with procedures you create outside of the Designer to extend PowerCenter functionality. The Custom transformation functions allow you to develop the transformation logic in a procedure you associate with a Custom transformation. PowerCenter provides two sets of functions called generated and API functions. The Integration Service uses generated functions to interface with the procedure. When you create a Custom transformation and generate the source code files, the Designer includes the generated functions in the files. Use the API functions in the procedure code to develop the transformation logic.

When you write the procedure code, you can configure it to receive a block of rows from the Integration Service or a single row at a time. You can increase the procedure performance when it receives and processes a block of rows. For more information about receiving rows from the Integration Service, see “Working with Rows” on page 96.

Working with Handles

Most functions are associated with a handle, such as INFA_CT_PARTITION_HANDLE. The first parameter for these functions is the handle the function affects. Custom transformation handles have a hierarchical relationship to each other. A parent handle has a $1:n$ relationship to its child handle.

Figure 4-1 shows the Custom transformation handles:

Figure 4-1. Custom Transformation Handles

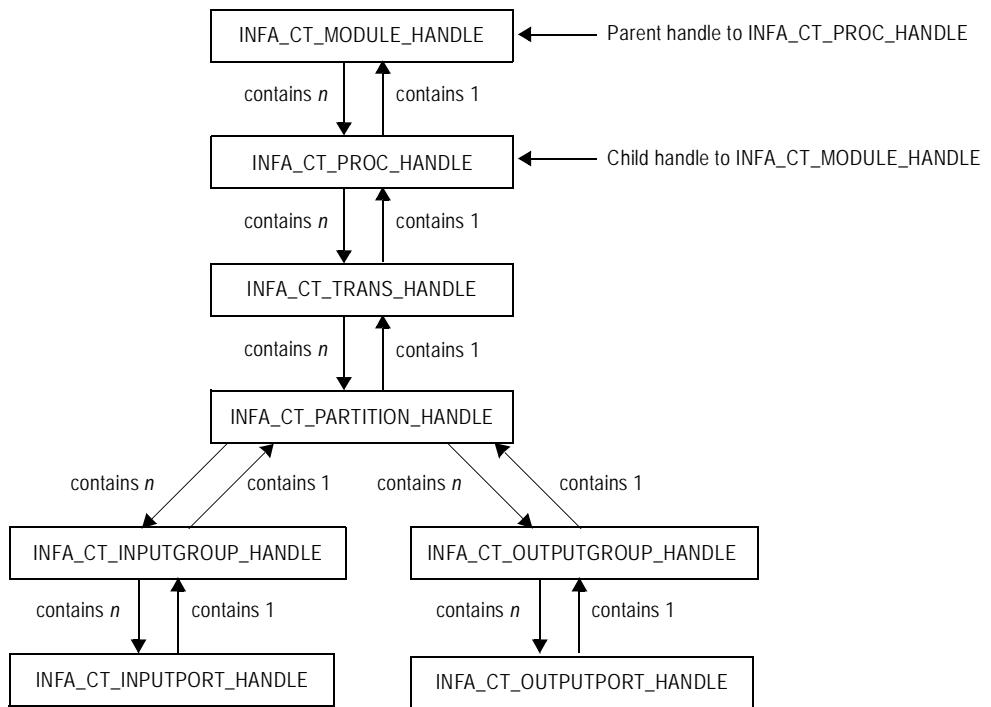


Table 4-1 describes the Custom transformation handles:

Table 4-1. Custom Transformation Handles

Handle Name	Description
INFA_CT_MODULE_HANDLE	Represents the shared library or DLL. The external procedure can only access the module handle in its own shared library or DLL. It cannot access the module handle in any other shared library or DLL.
INFA_CT_PROC_HANDLE	Represents a specific procedure within the shared library or DLL. You might use this handle when you need to write a function to affect a procedure referenced by multiple Custom transformations.
INFA_CT_TRANS_HANDLE	Represents a specific Custom transformation instance in the session.
INFA_CT_PARTITION_HANDLE	Represents a specific partition in a specific Custom transformation instance.
INFA_CT_INPUTGROUP_HANDLE	Represents an input group in a partition.
INFA_CT_INPUTPORT_HANDLE	Represents an input port in an input group in a partition.
INFA_CT_OUTPUTGROUP_HANDLE	Represents an output group in a partition.
INFA_CT_OUTPUTPORT_HANDLE	Represents an output port in an output group in a partition.

Function Reference

The Custom transformation functions include generated and API functions.

Table 4-2 lists the Custom transformation generated functions:

Table 4-2. Custom Transformation Generated Functions

Function	Description
m_<module_name>_moduleInit()	Module initialization function. For more information, see "Module Initialization Function" on page 98.
p_<proc_name>_procInit()	Procedure initialization function. For more information, see "Procedure Initialization Function" on page 99.
p_<proc_name>_partitionInit()	Partition initialization function. For more information, see "Partition Initialization Function" on page 99.
p_<proc_name>_inputRowNotification()	Input row notification function. For more information, see "Input Row Notification Function" on page 100.
p_<proc_name>_dataBdryNotification()	Data boundary notification function. For more information, see "Data Boundary Notification Function" on page 101.
p_<proc_name>_eofNotification()	End of file notification function. For more information, see "End Of File Notification Function" on page 101.
p_<proc_name>_partitionDeinit()	Partition deinitialization function. For more information, see "Partition Deinitialization Function" on page 102.
p_<proc_name>_procedureDeinit()	Procedure deinitialization function. For more information, see "Procedure Deinitialization Function" on page 102.
m_<module_name>_moduleDeinit()	Module deinitialization function. For more information, see "Module Deinitialization Function" on page 103.

Table 4-3 lists the Custom transformation API functions:

Table 4-3. Custom Transformation API Functions

Function	Description
INFA_CTSetsDataAccessMode()	Set data access mode function. For more information, see "Set Data Access Mode Function" on page 104.
INFA_CTCGetAncestorHandle()	Get ancestor handle function. For more information, see "Get Ancestor Handle Function" on page 105.
INFA_CTCGetChildrenHandles()	Get children handles function. For more information, see "Get Children Handles Function" on page 106.
INFA_CTCGetInputPortHandle()	Get input port handle function. For more information, see "Get Port Handle Functions" on page 107.
INFA_CTCGetOutputPortHandle()	Get output port handle function. For more information, see "Get Port Handle Functions" on page 107.

Table 4-3. Custom Transformation API Functions

Function	Description
INFA_CTGetInternalProperty<datatype>()	Get internal property function. For more information, see "Get Internal Property Function" on page 108.
INFA_CTGetAllPropertyNamesM()	Get all property names in MBCS mode function. For more information, see "Get All External Property Names (MBCS or Unicode)" on page 114.
INFA_CTGetAllPropertyNamesU()	Get all property names in Unicode mode function. For more information, see "Get All External Property Names (MBCS or Unicode)" on page 114.
INFA_CTGetExternalProperty<datatype>M()	Get external property in MBCS function. For more information, see "Get External Properties (MBCS or Unicode)" on page 114.
INFA_CTGetExternalProperty<datatype>U()	Get external property in Unicode function. For more information, see "Get External Properties (MBCS or Unicode)" on page 114.
INFA_CTRbindInputDataType()	Rebind input port datatype function. For more information, see "Rebind Datatype Functions" on page 115.
INFA_CTRbindOutputDataType()	Rebind output port datatype function. For more information, see "Rebind Datatype Functions" on page 115.
INFA_CTGetData<datatype>()	Get data functions. For more information, see "Get Data Functions (Row-Based Mode)" on page 118.
INFA_CTSetsData()	Set data functions. For more information, see "Set Data Function (Row-Based Mode)" on page 118.
INFA_CTGetIndicator()	Get indicator function. For more information, see "Indicator Functions (Row-Based Mode)" on page 119.
INFA_CTSetsIndicator()	Set indicator function. For more information, see "Indicator Functions (Row-Based Mode)" on page 119.
INFA_CTGetLength()	Get length function. For more information, see "Length Functions" on page 120.
INFA_CTSetsLength()	Set length function. For more information, see "Length Functions" on page 120.
INFA_CTSetsPassThruPort()	Set pass-through port function. For more information, see "Set Pass-Through Port Function" on page 120.
INFA_CTOutputNotification()	Output notification function. For more information, see "Output Notification Function" on page 121.
INFA_CTDatasBdryOutputNotification()	Data boundary output notification function. For more information, see "Data Boundary Output Notification Function" on page 121.
INFA_CTGetsErrorMsgU()	Get error message in Unicode function. For more information, see "Error Functions" on page 122.
INFA_CTGetsErrorMsgM()	Get error message in MBCS function. For more information, see "Error Functions" on page 122.
INFA_CTLLogMessageU()	Log message in the session log in Unicode function. For more information, see "Session Log Message Functions" on page 123.

Table 4-3. Custom Transformation API Functions

Function	Description
INFA_CTCLogMessageM0()	Log message in the session log in MBCS function. For more information, see "Session Log Message Functions" on page 123.
INFA_CTIIncrementErrorCount()	Increment error count function. For more information, see "Increment Error Count Function" on page 124.
INFA_CTIIsTerminateRequested()	Is terminate requested function. For more information, see "Is Terminated Function" on page 124.
INFA_CTBBlockInputFlow()	Block input groups function. For more information, see "Blocking Functions" on page 125.
INFA_CTUnblockInputFlow()	Unblock input groups function. For more information, see "Blocking Functions" on page 125.
INFA_CTSUserDefinedPtr()	Set user-defined pointer function. For more information, see "Pointer Functions" on page 126.
INFA_CTGUserDefinedPtr()	Get user-defined pointer function. For more information, see "Pointer Functions" on page 126.
INFA_CTCChangeStringMode()	Change the string mode function. For more information, see "Change String Mode Function" on page 126.
INFA_CTSDataCodePageID()	Set the data code page ID function. For more information, see "Set Data Code Page Function" on page 127.
INFA_CTGGetRowStrategy()	Get row strategy function. For more information, see "Row Strategy Functions (Row-Based Mode)" on page 128.
INFA_CTSRowStrategy()	Set the row strategy function. For more information, see "Row Strategy Functions (Row-Based Mode)" on page 128.
INFA_CTCChangeDefaultRowStrategy()	Change the default row strategy of a transformation. For more information, see "Change Default Row Strategy Function" on page 129.

Table 4-4 lists the Custom transformation array-based functions:

Table 4-4. Custom Transformation Array-Based API Functions

Function	Description
INFA_CTAGetInputRowMax()	Get maximum number of input rows function. For more information, see "Maximum Number of Rows Functions" on page 130.
INFA_CTAGetOutputRowMax()	Get maximum number of output rows function. For more information, see "Maximum Number of Rows Functions" on page 130.
INFA_CTASetOutputRowMax()	Set maximum number of output rows function. For more information, see "Maximum Number of Rows Functions" on page 130.
INFA_CTAGetNumRows()	Get number of rows function. For more information, see "Number of Rows Functions" on page 131.
INFA_CTASetNumRows()	Set number of rows function. For more information, see "Number of Rows Functions" on page 131.

Table 4-4. Custom Transformation Array-Based API Functions

Function	Description
INFA_CTAIsRowValid()	Is row valid function. For more information, see "Is Row Valid Function" on page 132.
INFA_CTAGetData<datatype>()	Get data functions. For more information, see "Get Data Functions (Array-Based Mode)" on page 133.
INFA_CTAGetIndicator()	Get indicator function. For more information, see "Get Indicator Function (Array-Based Mode)" on page 134.
INFA_CTASetData()	Set data function. For more information, see "Set Data Function (Array-Based Mode)" on page 134.
INFA_CTAGetRowStrategy()	Get row strategy function. For more information, see "Row Strategy Functions (Array-Based Mode)" on page 135.
INFA_CTASetRowStrategy()	Set row strategy function. For more information, see "Row Strategy Functions (Array-Based Mode)" on page 135.
INFA_CTASetInputErrorRowM()	Set input error row function for MBCS. For more information, see "Set Input Error Row Functions" on page 136.
INFA_CTASetInputErrorRowU()	Set input error row function for Unicode. For more information, see "Set Input Error Row Functions" on page 136.

Working with Rows

The Integration Service can pass a single row to a Custom transformation procedure or a block of rows in an array. You can write the procedure code to specify whether the procedure receives one row or a block of rows. You can increase performance when the procedure receives a block of rows:

- ◆ You can decrease the number of function calls the Integration Service and procedure make. The Integration Service calls the input row notification function fewer times, and the procedure calls the output notification function fewer times.
- ◆ You can increase the locality of memory access space for the data.
- ◆ You can write the procedure code to perform an algorithm on a block of data instead of each row of data.

By default, the procedure receives a row of data at a time. To receive a block of rows, you must include the INFA_CTSetsDataAccessMode() function to change the data access mode to array-based. When the data access mode is array-based, you must use the array-based data handling and row strategy functions to access and output the data. When the data access mode is row-based, you must use the row-based data handling and row strategy functions to access and output the data.

All array-based functions use the prefix INFA_CTA. All other functions use the prefix INFA_CT. For more information about the array-based functions, see “Array-Based API Functions” on page 130.

Use the following steps to write the procedure code to access a block of rows:

1. Call INFA_CTSetsDataAccessMode() during the procedure initialization, to change the data access mode to array-based.
2. When you create a passive Custom transformation, you can also call INFA_CTSetsPassThruPort() during procedure initialization to pass through the data for input/output ports.

When a block of data reaches the Custom transformation procedure, the Integration Service calls p_<proc_name>_inputRowNotification() for each block of data. Perform the rest of the steps inside this function.

3. Call INFA_CTAGetNumRows() using the input group handle in the input row notification function to find the number of rows in the current block.
4. Call one of the INFA_CTAGetData<datatype>() functions using the input port handle to get the data for a particular row in the block.
5. Call INFA_CTASetData to output rows in a block.
6. Before calling INFA_CTOOutputNotification(), call INFA_CTASetNumRows() to notify the Integration Service of the number of rows the procedure is outputting in the block.
7. Call INFA_CTOOutputNotification().

Rules and Guidelines

Use the following rules and guidelines when you write the procedure code to use either row-based or array-based data access mode:

- ◆ In row-based mode, you can return INFA_ROWERROR in the input row notification function to indicate the function encountered an error for the row of data on input. The Integration Service increments the internal error count.
- ◆ In array-based mode, do not return INFA_ROWERROR in the input row notification function. The Integration Service treats that as a fatal error. If you need to indicate a row in a block has an error, call the INFA_CTASetInputErrorRowM() or INFA_CTASetInputErrorRowU() function.
- ◆ In row-based mode, the Integration Service only passes valid rows to the procedure.
- ◆ In array-based mode, an input block may contain invalid rows, such as dropped, filtered, or error rows. Call INFA_CTAIsRowValid() to determine if a row in a block is valid.
- ◆ In array-based mode, do not call INFA_CTASetNumRows() for a passive Custom transformation. You can call this function for active Custom transformations.
- ◆ In array-based mode, call INFA_CTOOutputNotification() once.
- ◆ In array-based mode, you can call INFA_CTSetPassThruPort() only for passive Custom transformations.
- ◆ In array-based mode for passive Custom transformations, you must output all rows in an output block, including any error row.

Generated Functions

When you use the Designer to generate the procedure code, the Designer includes a set of functions called generated functions in the m_<module_name>.c and p_<procedure_name>.c files. The Integration Service uses the generated functions to interface with the procedure. When you run a session, the Integration Service calls these generated functions in the following order for each target load order group in the mapping:

1. Initialization functions
2. Notification functions
3. Deinitialization functions

Initialization Functions

The Integration Service first calls the initialization functions. Use the initialization functions to write processes you want the Integration Service to run before it passes data to the Custom transformation. Writing code in the initialization functions reduces processing overhead because the Integration Service runs these processes only once for a module, procedure, or partition.

The Designer generates the following initialization functions:

- ◆ **m_<module_name>_moduleInit()**. For more information, see “Module Initialization Function” on page 98.
- ◆ **p_<proc_name>_procInit()**. For more information, see “Procedure Initialization Function” on page 99.
- ◆ **p_<proc_name>_partitionInit()**. For more information, see “Partition Initialization Function” on page 99.

Module Initialization Function

The Integration Service calls the m_<module_name>_moduleInit() function during session initialization, before it runs the pre-session tasks. It calls this function, once for a module, before all other functions.

If you want the Integration Service to run a specific process when it loads the module, you must include it in this function. For example, you might write code to create global structures that procedures within this module access.

Use the following syntax:

```
INFA_STATUS m_<module_name>_moduleInit(INFA_CT_MODULE_HANDLE module);
```

Argument	Datatype	Input/ Output	Description
module	INFA_CT_MODULE_HANDLE	Input	Module handle.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

Procedure Initialization Function

The Integration Service calls p_<proc_name>_procInit() function during session initialization, before it runs the pre-session tasks and after it runs the module initialization function. The Integration Service calls this function once for each procedure in the module.

Write code in this function when you want the Integration Service to run a process for a particular procedure. You can also enter some API functions in the procedure initialization function, such as navigation and property functions.

Use the following syntax:

```
INFA_STATUS p_<proc_name>_procInit(INFA_CT_PROCEDURE_HANDLE procedure);
```

Argument	Datatype	Input/ Output	Description
procedure	INFA_CT_PROCEDURE_HANDLE	Input	Procedure handle.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

Partition Initialization Function

The Integration Service calls p_<proc_name>_partitionInit() function before it passes data to the Custom transformation. The Integration Service calls this function once for each partition at a Custom transformation instance.

If you want the Integration Service to run a specific process before it passes data through a partition of the Custom transformation, you must include it in this function.

Use the following syntax:

```
INFA_STATUS p_<proc_name>_partitionInit(INFA_CT_PARTITION_HANDLE  
transformation);
```

Argument	Datatype	Input/ Output	Description
transformation	INFA_CT_PARTITION_HANDLE	Input	Partition handle.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

Note: When the Custom transformation requires one thread for each partition, you can include thread-specific operations in the partition initialization function. For more

information about working with thread-specific procedure code, see “Working with Thread-Specific Procedure Code” on page 66.

Notification Functions

The Integration Service calls the notification functions when it passes a row of data to the Custom transformation.

The Designer generates the following notification functions:

- ♦ `p_<proc_name>_inputRowNotification()`. For more information, see “Input Row Notification Function” on page 100.
- ♦ `p_<proc_name>_dataBdryRowNotification()`. For more information, see “Data Boundary Notification Function” on page 101.
- ♦ `p_<proc_name>_eofNotification()`. For more information, see “End Of File Notification Function” on page 101.

Note: When the Custom transformation requires one thread for each partition, you can include thread-specific operations in the notification functions. For more information about working with thread-specific procedure code, see “Working with Thread-Specific Procedure Code” on page 66.

Input Row Notification Function

The Integration Service calls the `p_<proc_name>_inputRowNotification()` function when it passes a row or a block of rows to the Custom transformation. It notes which input group and partition receives data through the input group handle and partition handle.

Use the following syntax:

```
INFA_ROWSTATUS  
p_<proc_name>_inputRowNotification(INFA_CT_PARTITION_HANDLE Partition,  
INFA_CT_INPUTGROUP_HANDLE group);
```

Argument	Datatype	Input/ Output	Description
partition	INFA_CT_PARTITION_HANDLE	Input	Partition handle.
group	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.

The datatype of the return value is INFA_ROWSTATUS. Use the following values for the return value:

- ♦ `INFA_ROWSUCCESS`. Indicates the function successfully processed the row of data.
- ♦ `INFA_ROWERROR`. Indicates the function encountered an error for the row of data. The Integration Service increments the internal error count. Only return this value when the data access mode is row.

If the input row notification function returns `INFA_ROWERROR` in array-based mode, the Integration Service treats it as a fatal error. If you need to indicate a row in a block has

an error, call the INFA_CTASetInputErrorRowM() or INFA_CTASetInputErrorRowU() function.

- ◆ **INFA_FATALError.** Indicates the function encountered a fatal error for the row of data or the block of data. The Integration Service fails the session.

Data Boundary Notification Function

The Integration Service calls the p_<proc_name>_dataBdryNotification() function when it passes a commit or rollback row to a partition.

Use the following syntax:

```
INFA_STATUS p_<proc_name>_dataBdryNotification(INFA_CT_PARTITION_HANDLE  
transformation, INFA_CTDaDataBdryType dataBoundaryType);
```

Argument	Datatype	Input/ Output	Description
transformation	INFA_CT_PARTITION_HANDLE	Input	Partition handle.
dataBoundaryType	INFA_CTDaDataBdryType	Input	Integration Service uses one of the following values for the dataBoundaryType parameter: - eBT_COMMIT - eBT_ROLLBACK

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

End Of File Notification Function

The Integration Service calls the p_<proc_name>_eofNotification() function after it passes the last row to a partition in an input group.

Use the following syntax:

```
INFA_STATUS p_<proc_name>_eofNotification(INFA_CT_PARTITION_HANDLE  
transformation, INFA_CT_INPUTGROUP_HANDLE group);
```

Argument	Datatype	Input/ Output	Description
transformation	INFA_CT_PARTITION_HANDLE	Input	Partition handle.
group	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

Deinitialization Functions

The Integration Service calls the deinitialization functions after it processes data for the Custom transformation. Use the deinitialization functions to write processes you want the Integration Service to run after it passes all rows of data to the Custom transformation.

The Designer generates the following deinitialization functions:

- ◆ `p_<proc_name>_partitionDeinit()`. For more information, see “Partition Deinitialization Function” on page 102.
- ◆ `p_<proc_name>_procDeinit()`. For more information, see “Procedure Deinitialization Function” on page 102.
- ◆ `m_<module_name>_moduleDeinit()`. For more information, see “Module Deinitialization Function” on page 103.

Note: When the Custom transformation requires one thread for each partition, you can include thread-specific operations in the initialization and deinitialization functions. For more information about working with thread-specific procedure code, see “Working with Thread-Specific Procedure Code” on page 66.

Partition Deinitialization Function

The Integration Service calls the `p_<proc_name>_partitionDeinit()` function after it calls the `p_<proc_name>_eofNotification()` or `p_<proc_name>_abortNotification()` function. The Integration Service calls this function once for each partition of the Custom transformation.

Use the following syntax:

```
INFA_STATUS p_<proc_name>_partitionDeinit(INFA_CT_PARTITION_HANDLE  
partition);
```

Argument	Datatype	Input/ Output	Description
partition	INFA_CT_PARTITION_HANDLE	Input	Partition handle.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

Note: When the Custom transformation requires one thread for each partition, you can include thread-specific operations in the partition deinitialization function. For more information about working with thread-specific procedure code, see “Working with Thread-Specific Procedure Code” on page 66.

Procedure Deinitialization Function

The Integration Service calls the `p_<proc_name>_procDeinit()` function after it calls the `p_<proc_name>_partitionDeinit()` function for all partitions of each Custom transformation instance that uses this procedure in the mapping.

Use the following syntax:

```
INFA_STATUS p_<proc_name>_procDeinit(INFA_CT_PROCEDURE_HANDLE procedure,  
INFA_STATUS sessionStatus);
```

Argument	Datatype	Input/ Output	Description
procedure	INFA_CT_PROCEDURE_HANDLE	Input	Procedure handle.
sessionStatus	INFA_STATUS	Input	Integration Service uses one of the following values for the sessionStatus parameter: <ul style="list-style-type: none">- INFA_SUCCESS. Indicates the session succeeded.- INFA_FAILURE. Indicates the session failed.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

Module Deinitialization Function

The Integration Service calls the m_<module_name>_moduleDeinit() function after it runs the post-session tasks. It calls this function, once for a module, after all other functions.

Use the following syntax:

```
INFA_STATUS m_<module_name>_moduleDeinit(INFA_CT_MODULE_HANDLE module,  
INFA_STATUS sessionStatus);
```

Argument	Datatype	Input/ Output	Description
module	INFA_CT_MODULE_HANDLE	Input	Module handle.
sessionStatus	INFA_STATUS	Input	Integration Service uses one of the following values for the sessionStatus parameter: <ul style="list-style-type: none">- INFA_SUCCESS. Indicates the session succeeded.- INFA_FAILURE. Indicates the session failed.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value. When the function returns INFA_FAILURE, the Integration Service fails the session.

API Functions

PowerCenter provides a set of API functions that you use to develop the transformation logic. When the Designer generates the source code files, it includes the generated functions in the source code. Add API functions to the code to implement the transformation logic. The procedure uses the API functions to interface with the Integration Service. You must code API functions in the procedure C file. Optionally, you can also code the module C file.

Informatica provides the following groups of API functions:

- ◆ **Set data access mode.** See “Set Data Access Mode Function” on page 104.
- ◆ **Navigation.** See “Navigation Functions” on page 105.
- ◆ **Property.** See “Property Functions” on page 108.
- ◆ **Rebind datatype.** See “Rebind Datatype Functions” on page 115.
- ◆ **Data handling (row-based mode).** See “Data Handling Functions (Row-Based Mode)” on page 117.
- ◆ **Set pass-through port.** See “Set Pass-Through Port Function” on page 120.
- ◆ **Output notification.** See “Output Notification Function” on page 121.
- ◆ **Data boundary output notification.** See “Data Boundary Output Notification Function” on page 121.
- ◆ **Error.** See “Error Functions” on page 122.
- ◆ **Session log message.** See “Session Log Message Functions” on page 123.
- ◆ **Increment error count.** See “Increment Error Count Function” on page 124.
- ◆ **Is terminated.** See “Is Terminated Function” on page 124.
- ◆ **Blocking.** See “Blocking Functions” on page 125.
- ◆ **Pointer.** See “Pointer Functions” on page 126.
- ◆ **Change string mode.** See “Change String Mode Function” on page 126.
- ◆ **Set data code page.** See “Set Data Code Page Function” on page 127.
- ◆ **Row strategy (row-based mode).** See “Row Strategy Functions (Row-Based Mode)” on page 128.
- ◆ **Change default row strategy.** See “Change Default Row Strategy Function” on page 129.

Informatica also provides array-based API Functions. For more information about array-based API functions, see “Array-Based API Functions” on page 130.

Set Data Access Mode Function

By default, the Integration Service passes data to the Custom transformation procedure one row at a time. However, use the INFA_CTSetsDataAccessMode() function to change the data access mode to array-based. When you set the data access mode to array-based, the Integration Service passes multiple rows to the procedure as a block in an array.

When you set the data access mode to array-based, you must use the array-based versions of the data handling functions and row strategy functions. When you use a row-based data handling or row strategy function and you switch to array-based mode, you will get unexpected results. For example, the DLL or shared library might crash.

You can only use this function in the procedure initialization function.

If you do not use this function in the procedure code, the data access mode is row-based. However, when you want the data access mode to be row-based, include this function and set the access mode to row-based.

For more information about the array-based functions, see “Array-Based API Functions” on page 130.

Use the following syntax:

```
INFA_STATUS INFA_CTSetDataAccessMode( INFA_CT PROCEDURE_HANDLE procedure,  
INFA_CT DATA_ACCESS_MODE mode );
```

Argument	Datatype	Input/ Output	Description
procedure	INFA_CT PROCEDURE_HANDLE	Input	Procedure name.
mode	INFA_CT DATA_ACCESS_MODE	Input	Data access mode. Use the following values for the mode parameter: - eDA_ROW - eDA_ARRAY

Navigation Functions

Use the navigation functions when you want the procedure to navigate through the handle hierarchy. For more information about handles, see “Working with Handles” on page 90.

PowerCenter provides the following navigation functions:

- ◆ **INFA_CTGetAncestorHandle()**. For more information, see “Get Ancestor Handle Function” on page 105.
- ◆ **INFA_CTGetChildrenHandles()**. For more information, see “Get Children Handles Function” on page 106.
- ◆ **INFA_CTGetInputPortHandle()**. For more information, see “Get Port Handle Functions” on page 107.
- ◆ **INFA_CTGetOutputPortHandle()**. For more information, see “Get Port Handle Functions” on page 107.

Get Ancestor Handle Function

Use the INFA_CTGetAncestorHandle() function when you want the procedure to access a parent handle of a given handle.

Use the following syntax:

```
INFA_CT_HANDLE INFA_CHttpGetAncestorHandle(INFA_CT_HANDLE handle,  
INFA_CTHandleType returnHandleType);
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_HANDLE	Input	Handle name.
returnHandleType	INFA_CTHandleType	Input	Return handle type. Use the following values for the returnHandleType parameter: - PROCEDURETYPE - TRANSFORMATIONTYPE - PARTITIONTYPE - INPUTGROUPTYPE - OUTPUTGROUPTYPE - INPUTPORTTYPE - OUTPUTPORTTYPE

The handle parameter specifies the handle whose parent you want the procedure to access. The Integration Service returns INFA_CT_HANDLE if you specify a valid handle in the function. Otherwise, it returns a null value.

To avoid compilation errors, you must code the procedure to set a handle name to the return value.

For example, you can enter the following code:

```
INFA_CT_MODULE_HANDLE module = INFA_CHttpGetAncestorHandle(procedureHandle,  
INFA_CTHandleType);
```

Get Children Handles Function

Use the INFA_CHttpGetChildrenHandles() function when you want the procedure to access the children handles of a given handle.

Use the following syntax:

```
INFA_CT_HANDLE* INFA_CHttpGetChildrenHandles(INFA_CT_HANDLE handle, size_t*  
pnChildrenHandles, INFA_CTHandleType returnHandleType);
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_HANDLE	Input	Handle name.

Argument	Datatype	Input/ Output	Description
pnChildrenHandles	size_t*	Output	Integration Service returns an array of children handles. The pnChildrenHandles parameter indicates the number of children handles in the array.
returnHandleType	INFA_CTHandleType	Input	Use the following values for the returnHandleType parameter: - PROCEDURETYPE - TRANSFORMATIONTYPE - PARTITIONTYPE - INPUTGROUPTYPE - OUTPUTGROUPTYPE - INPUTPORTTYPE - OUTPUTPORTTYPE

The handle parameter specifies the handle whose children you want the procedure to access. The Integration Service returns INFA_CT_HANDLE* when you specify a valid handle in the function. Otherwise, it returns a null value.

To avoid compilation errors, you must code the procedure to set a handle name to the returned value.

For example, you can enter the following code:

```
INFA_CT_PARTITION_HANDLE partition =
INFA_CTGetChildrenHandles(procedureHandle, pnChildrenHandles,
INFA_CT_PARTITION_HANDLE_TYPE);
```

Get Port Handle Functions

The Integration Service associates the INFA_CT_INPUTPORT_HANDLE with input and input/output ports, and the INFA_CT_OUTPUTPORT_HANDLE with output and input/output ports.

PowerCenter provides the following get port handle functions:

- ◆ **INFA_CTGetInputPortHandle()**. Use this function when the procedure knows the output port handle for an input/output port and needs the input port handle.

Use the following syntax:

```
INFA_CTIINFA_CT_INPUTPORT_HANDLE
INFA_CTGetInputPortHandle(INFA_CT_OUTPUTPORT_HANDLE outputPortHandle);
```

Argument	Datatype	Input/ Output	Description
outputPortHandle	INFA_CT_OUTPUTPORT_HANDLE	input	Output port handle.

- ◆ **INFA_CTGetOutputPortHandle()**. Use this function when the procedure knows the input port handle for an input/output port and needs the output port handle.

Use the following syntax:

```
INFA_CT_OUTPUTPORT_HANDLE  
INFA_CTGetOutputPortHandle(INFA_CT_INPUTPORT_HANDLE inputPortHandle);
```

Argument	Datatype	Input/ Output	Description
inputPortHandle	INFA_CT_INPUTPORT_HANDLE	input	Input port handle.

The Integration Service returns NULL when you use the get port handle functions with input or output ports.

Property Functions

Use the property functions when you want the procedure to access the Custom transformation properties. The property functions access properties on the following tabs of the Custom transformation:

- ◆ Ports
- ◆ Properties
- ◆ Initialization Properties
- ◆ Metadata Extensions
- ◆ Port Attribute Definitions

Use the following property functions in initialization functions:

- ◆ INFA_CTGetProperty<datatype>(). For more information, see “Get Internal Property Function” on page 108.
- ◆ INFA_CTCGetAllPropertyNamesM(). For more information, see “Get All External Property Names (MBCS or Unicode)” on page 114.
- ◆ INFA_CTCGetAllPropertyNamesU(). For more information, see “Get All External Property Names (MBCS or Unicode)” on page 114.
- ◆ INFA_CTCGetExternalProperty<datatype>M(). For more information, see “Get External Properties (MBCS or Unicode)” on page 114.
- ◆ INFA_CTCGetExternalProperty<datatype>U(). For more information, see “Get External Properties (MBCS or Unicode)” on page 114.

Get Internal Property Function

PowerCenter provides functions to access the port attributes specified on the ports tab, and properties specified for attributes on the Properties tab of the Custom transformation.

The Integration Service associates each port and property attribute with a property ID. You must specify the property ID in the procedure to access the values specified for the attributes. For more information about property IDs, see “Port and Property Attribute Property IDs” on page 109. For the handle parameter, specify a handle name from the handle hierarchy. The Integration Service fails the session if the handle name is invalid.

Use the following functions when you want the procedure to access the properties:

- ◆ **INFA_CTGetInternalPropertyStringM()**. Accesses a value of type string in MBCS for a given property ID.

Use the following syntax:

```
INFA_STATUS INFA_CTGetInternalPropertyStringM( INFA_CT_HANDLE handle,  
size_t propId, const char** psPropValue );
```

- ◆ **INFA_CTGetInternalPropertyStringU()**. Accesses a value of type string in Unicode for a given property ID.

Use the following syntax:

```
INFA_STATUS INFA_CTGetInternalPropertyStringU( INFA_CT_HANDLE handle,  
size_t propId, const INFA_UNICHAR** psPropValue );
```

- ◆ **INFA_CTGetInternalPropertyInt32()**. Accesses a value of type integer for a given property ID.

Use the following syntax:

```
INFA_STATUS INFA_CTGetInternalPropertyInt32( INFA_CT_HANDLE handle,  
size_t propId, INFA_INT32* pnPropValue );
```

- ◆ **INFA_CTGetInternalPropertyBool()**. Accesses a value of type Boolean for a given property ID.

Use the following syntax:

```
INFA_STATUS INFA_CTGetInternalPropertyBool( INFA_CT_HANDLE handle, size_t  
propId, INFA_BOOL* pbPropValue );
```

- ◆ **INFA_CTGetInternalPropertyINFA_PTR()**. Accesses a pointer to a value for a given property ID.

Use the following syntax:

```
INFA_STATUS INFA_CTGetInternalPropertyINFA_PTR( INFA_CT_HANDLE handle,  
size_t propId, INFA_PTR* pvPropValue );
```

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Port and Property Attribute Property IDs

The following tables list the property IDs for the port and property attributes in the Custom transformation. Each table lists a Custom transformation handle and the property IDs you can access with the handle in a property function.

Table 4-5 lists INFA_CT_MODULE _HANDLE property IDs:

Table 4-5. INFA_CT_MODULE Property IDs

Handle	Property ID	Datatype	Description
INFA_CT_MODULE_NAME		String	Specifies the module name.
INFA_CT_SESSION_INFA_VERSION		String	Specifies the Informatica version.

Table 4-5. INFA_CT_MODULE Property IDs

Handle Property ID	Datatype	Description
INFA_CT_SESSION_CODE_PAGE	Integer	Specifies the Integration Service code page.
INFA_CT_SESSION_DATAMOVEMENT_MODE	Integer	Specifies the data movement mode. The Integration Service returns one of the following values: <ul style="list-style-type: none">- eASM_MBCS- eASM_UNICODE
INFA_CT_SESSION_VALIDATE_CODEPAGE	Boolean	Specifies whether the Integration Service enforces code page validation.
INFA_CT_SESSION_PROD_INSTALL_DIR	String	Specifies the Integration Service installation directory.
INFA_CT_SESSION_HIGH_PRECISION_MODE	Boolean	Specifies whether session is configured for high precision.
INFA_CT_MODULE_RUNTIME_DIR	String	Specifies the runtime directory for the DLL or shared library.
INFA_CT_SESSION_IS_UPD_STR_ALLOWED	Boolean	Specifies whether the Update Strategy Transformation property is selected in the transformation.
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	Integer	Specifies whether the Custom transformation produces data in the same order in every session run. The Integration Service returns one of the following values: <ul style="list-style-type: none">- eOUTREPEAT_NEVER = 1- eOUTREPEAT_ALWAYS = 2- eOUTREPEAT_BASED_ON_INPUT_ORDER = 3
INFA_CT_TRANS_FATAL_ERROR	Boolean	Specifies if the Custom Transformation caused a fatal error. The Integration Service returns one of the following values: <ul style="list-style-type: none">- INFA_TRUE- INFA_FALSE

Table 4-6 lists INFA_CT_PROC_HANDLE property IDs:

Table 4-6. INFA_CT_PROC_HANDLE Property IDs

Handle Property ID	Datatype	Description
INFA_CT_PROCEDURE_NAME	String	Specifies the Custom transformation procedure name.

Table 4-7 lists INFA_CT_TRANS_HANDLE property IDs:

Table 4-7. INFA_CT_TRANS_HANDLE Property IDs

Handle Property ID	Datatype	Description
INFA_CT_TRANS_INSTANCE_NAME	String	Specifies the Custom transformation instance name.
INFA_CT_TRANS_TRACE_LEVEL	Integer	Specifies the tracing level. The Integration Service returns one of the following values: - eTRACE_TERSE - eTRACE_NORMAL - eTRACE_VERBOSE_INIT - eTRACE_VERBOSE_DATA
INFA_CT_TRANS_MAY_BLOCK_DATA	Boolean	Specifies if the Integration Service allows the procedure to block input data in the current session.
INFA_CT_TRANS_MUST_BLOCK_DATA	Boolean	Specifies if the Inputs Must Block Custom transformation property is selected.
INFA_CT_TRANS_ISACTIVE	Boolean	Specifies whether the Custom transformation is an active or passive transformation.
INFA_CT_TRANS_ISPARTITIONABLE	Boolean	Specifies if you can partition sessions that use this Custom transformation.
INFA_CT_TRANS_IS_UPDATE_STRATEGY	Boolean	Specifies if the Custom transformation behaves like an Update Strategy transformation.
INFA_CT_TRANS_DEFAULT_UPDATE_STRATEGY	Integer	Specifies the default update strategy. - eDUS_INSERT - eDUS_UPDATE - eDUS_DELETE - eDUS_REJECT - eDUS_PASSTHROUGH
INFA_CT_TRANS_NUM_PARTITIONS	Integer	Specifies the number of partitions in the sessions that use this Custom transformation.
INFA_CT_TRANS_DATACODEPAGE	Integer	Specifies the code page in which the Integration Service passes data to the Custom transformation. Use the set data code page function if you want the Custom transformation to access data in a different code page. For more information, see "Set Data Code Page Function" on page 127.
INFA_CT_TRANS_TRANSFORM_SCOPE	Integer	Specifies the transformation scope in the Custom transformation. The Integration Service returns one of the following values: - eTS_ROW - eTS_TRANSACTION - eTS_ALLINPUT

Table 4-7. INFA_CT_TRANS_HANDLE Property IDs

Handle Property ID	Datatype	Description
INFA_CT_TRANS_GENERATE_TRANSACT	Boolean	Specifies if the Generate Transaction property is enabled. The Integration Service returns one of the following values: - INFA_TRUE - INFA_FALSE
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	Integer	Specifies whether the Custom transformation produces data in the same order in every session run. The Integration Service returns one of the following values: - eOUTREPEAT_NEVER = 1 - eOUTREPEAT_ALWAYS = 2 - eOUTREPEAT_BASED_ON_INPUT_ORDER = 3
INFA_CT_TRANS_FATAL_ERROR	Boolean	Specifies if the Custom Transformation caused a fatal error. The Integration Service returns one of the following values: - INFA_TRUE - INFA_FALSE

Table 4-8 lists INFA_CT_INPUT_GROUP_HANDLE and INFA_CT_OUTPUT_GROUP_HANDLE property IDs:

Table 4-8. INFA_CT_INPUT_GROUP and INFA_CT_OUTPUT_GROUP Handle Property IDs

Handle Property ID	Datatype	Description
INFA_CT_GROUP_NAME	String	Specifies the group name.
INFA_CT_GROUP_NUM_PORTS	Integer	Specifies the number of ports in the group.
INFA_CT_GROUP_ISCONNECTED	Boolean	Specifies if all ports in a group are connected to another transformation.
INFA_CT_PORT_NAME	String	Specifies the port name.
INFA_CT_PORT_CDATATYPE	Integer	Specifies the port datatype. The Integration Service returns one of the following values: - eINFA_CTYPE_SHORT - eINFA_CTYPE_INT32 - eINFA_CTYPE_CHAR - eINFA_CTYPE_RAW - eINFA_CTYPE_UNICHAR - eINFA_CTYPE_TIME - eINFA_CTYPE_FLOAT - eINFA_CTYPE_DOUBLE - eINFA_CTYPE_DECIMAL18_FIXED - eINFA_CTYPE_DECIMAL28_FIXED - eINFA_CTYPE_INFA_CTDATETIME
INFA_CT_PORT_PRECISION	Integer	Specifies the port precision.
INFA_CT_PORT_SCALE	Integer	Specifies the port scale (if applicable).

Table 4-8. INFA_CT_INPUT_GROUP and INFA_CT_OUTPUT_GROUP Handle Property IDs

Handle Property ID	Datatype	Description
INFA_CT_PORT_IS_MAPPED	Boolean	Specifies whether the port is linked to other transformations in the mapping.
INFA_CT_PORT_STORAGESIZE	Integer	Specifies the internal storage size of the data for a port. The storage size depends on the datatype of the port.
INFA_CT_PORT_BOUNDDATATYPE	Integer	Specifies the port datatype. Use instead of INFA_CT_PORT_CDATATYPE if you rebind the port and specify a datatype other than the default. For more information about rebinding a port, see "Rebind Datatype Functions" on page 115.

Table 4-9 lists INFA_CT_INPUTPORT_HANDLE and INFA_CT_OUTPUTPORT_HANDLE property IDs:

Table 4-9. INFA_CT_INPUTPORT and INFA_CT_OUTPUTPORT_HANDLE Handle Property IDs

Handle Property ID	Datatype	Description
INFA_CT_PORT_NAME	String	Specifies the port name.
INFA_CT_PORT_CDATATYPE	Integer	Specifies the port datatype. The Integration Service returns one of the following values: - eINFA_CTYPE_SHORT - eINFA_CTYPE_INT32 - eINFA_CTYPE_CHAR - eINFA_CTYPE_RAW - eINFA_CTYPE_UNICHAR - eINFA_CTYPE_TIME - eINFA_CTYPE_FLOAT - eINFA_CTYPE_DOUBLE - eINFA_CTYPE_DECIMAL18_FIXED - eINFA_CTYPE_DECIMAL28_FIXED - eINFA_CTYPE_INFA_CTDATETIME
INFA_CT_PORT_PRECISION	Integer	Specifies the port precision.
INFA_CT_PORT_SCALE	Integer	Specifies the port scale (if applicable).
INFA_CT_PORT_IS_MAPPED	Boolean	Specifies whether the port is linked to other transformations in the mapping.
INFA_CT_PORT_STORAGESIZE	Integer	Specifies the internal storage size of the data for a port. The storage size depends on the datatype of the port.
INFA_CT_PORT_BOUNDDATATYPE	Integer	Specifies the port datatype. Use instead of INFA_CT_PORT_CDATATYPE if you rebind the port and specify a datatype other than the default. For more information about rebinding a port, see "Rebind Datatype Functions" on page 115.

Get All External Property Names (MBCS or Unicode)

PowerCenter provides two functions to access the property names defined on the Metadata Extensions tab, Initialization Properties tab, and Port Attribute Definitions tab of the Custom transformation.

Use the following functions when you want the procedure to access the property names:

- ◆ **INFA_CTGetAllPropertyNamesM()**. Accesses the property names in MBCS.

Use the following syntax:

```
INFA_STATUS INFA_CTGetAllPropertyNamesM(INFA_CT_HANDLE handle, const  
char*const** paPropertyNames, size_t* pnProperties);
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_HANDLE	Input	Specify the handle name.
paPropertyNames	const char*const**	Output	Specifies the property name. The Integration Service returns an array of property names in MBCS.
pnProperties	size_t*	Output	Indicates the number of properties in the array.

- ◆ **INFA_CTGetAllPropertyNamesU()**. Accesses the property names in Unicode.

Use the following syntax:

```
INFA_STATUS INFA_CTGetAllPropertyNamesU(INFA_CT_HANDLE handle, const  
INFA_UNICHAR*const** pasPropertyNames, size_t* pnProperties);
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_HANDLE	Input	Specify the handle name.
paPropertyNames	const INFA_UNICHAR*const**	Output	Specifies the property name. The Integration Service returns an array of property names in Unicode.
pnProperties	size_t*	Output	Indicates the number of properties in the array.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Get External Properties (MBCS or Unicode)

PowerCenter provides functions to access the values of the properties defined on the Metadata Extensions tab, Initialization Properties tab, or Port Attribute Definitions tab of the Custom transformation.

You must specify the property names in the functions if you want the procedure to access the values. Use the INFA_CTGetAllPropertyNamesM() or INFA_CTGetAllPropertyNamesU()

functions to access property names. For the handle parameter, specify a handle name from the handle hierarchy. The Integration Service fails the session if the handle name is invalid.

Note: If you define an initialization property with the same name as a metadata extension, the Integration Service returns the metadata extension value.

Use the following functions when you want the procedure to access the values of the properties:

- ◆ **INFA_CTGetExternalProperty<datatype>M()**. Accesses the value of the property in MBCS. Use the syntax as shown in Table 4-10:

Table 4-10. Property Functions (MBCS)

Syntax	Property Datatype
INFA_STATUS INFA_CTGetExternalPropertyStringM(INFA_CT_HANDLE handle, const char* sPropName, const char** psPropValue);	String
INFA_STATUS INFA_CTGetExternalPropertyINT32M(INFA_CT_HANDLE handle, const char* sPropName, INFA_INT32* pnPropValue);	Integer
INFA_STATUS INFA_CTGetExternalPropertyBoolM(INFA_CT_HANDLE handle, const char* sPropName, INFA_BOOLEN* pbPropValue);	Boolean

- ◆ **INFA_CTGetExternalProperty<datatype>U()**. Accesses the value of the property in Unicode. Use the syntax as shown in Table 4-11:

Table 4-11. Property Functions (Unicode)

Syntax	Property Datatype
INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_UNICHAR** psPropValue);	String
INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_INT32* pnPropValue);	Integer
INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_BOOLEN* pbPropValue);	Boolean

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Rebind Datatype Functions

You can rebind a port with a datatype other than the default datatype with PowerCenter. Use the rebind datatype functions if you want the procedure to access data in a datatype other than the default datatype. You must rebind the port with a compatible datatype.

You can only use these functions in the initialization functions.

Consider the following rules when you rebind the datatype for an output or input/output port:

- ◆ You must use the data handling functions to set the data and the indicator for that port. Use the INFA_CTSethData() and INFA_CTSethIndicator() functions in row-based mode, and use the INFA_CTASetData() function in array-based mode.
- ◆ Do not call the INFA_CTSethPassThruPort() function for the output port.

Table 4-12 lists compatible datatypes:

Table 4-12. Compatible Datatypes

Default Datatype	Compatible With
Char	Unichar
Unichar	Char
Date	INFA_DATETIME Use the following syntax: <pre>struct INFA_DATETIME { int nYear; int nMonth; int nDay; int nHour; int nMinute; int nSecond; int nNanoSecond; }</pre>
Dec18	Char, Unichar
Dec28	Char, Unichar

PowerCenter provides the following rebind datatype functions:

- ◆ **INFA_CTRerebindInputDataType()**. Rebinds the input port. Use the following syntax:

```
INFA_STATUS INFA_CTRerebindInputDataType(INFA_CT_INPUTPORT_HANDLE  
portHandle, INFA_CDATATYPE datatype);
```
- ◆ **INFA_CTRerebindOutputDataType()**. Rebinds the output port. Use the following syntax:

```
INFA_STATUS INFA_CTRerebindOutputDataType(INFA_CT_OUTPUTPORT_HANDLE  
portHandle, INFA_CDATATYPE datatype);
```

Argument	Datatype	Input/ Output	Description
portHandle	INFA_CT_OUTPUTPORT_HANDLE	Input	Output port handle.
datatype	INFA_CDATATYPE	Input	The datatype with which you rebind the port. Use the following values for the datatype parameter: - eINFA_CTYPE_SHORT - eINFA_CTYPE_INT32 - eINFA_CTYPE_CHAR - eINFA_CTYPE_RAW - eINFA_CTYPE_UNICHAR - eINFA_CTYPE_TIME - eINFA_CTYPE_FLOAT - eINFA_CTYPE_DOUBLE - eINFA_CTYPE_DECIMAL18_FIXED - eINFA_CTYPE_DECIMAL28_FIXED - eINFA_CTYPE_INFA_CTDATETIME

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Data Handling Functions (Row-Based Mode)

When the Integration Service calls the input row notification function, it notifies the procedure that the procedure can access a row or block of data. However, to get data from the input port, modify it, and set data in the output port, you must use the data handling functions in the input row notification function. When the data access mode is row-based, use the row-based data handling functions.

Include the INFA_CTCGetData<datatype>() function to get the data from the input port and INFA_CTCSetData() function to set the data in the output port. Include the INFA_CTCGetIndicator() or INFA_CTCGetLength() function if you want the procedure to verify before you get the data if the port has a null value or an empty string.

PowerCenter provides the following data handling functions:

- ◆ **INFA_CTCGetData<datatype>()**. For more information, see “Get Data Functions (Row-Based Mode)” on page 118.
- ◆ **INFA_CTCSetData()**. For more information, see “Set Data Function (Row-Based Mode)” on page 118.

- ◆ `INFA_CTGetIndicator()`. For more information, see “Indicator Functions (Row-Based Mode)” on page 119.
- ◆ `INFA_CTSetsIndicator()`. For more information, see “Indicator Functions (Row-Based Mode)” on page 119.
- ◆ `INFA_CTGetLength()`. For more information, see “Length Functions” on page 120.
- ◆ `INFA_CTSetsLength()`. For more information, see “Length Functions” on page 120.

Get Data Functions (Row-Based Mode)

Use the `INFA_CTDGetData<datatype>()` functions to retrieve data for the port the function specifies.

You must modify the function name depending on the datatype of the port you want the procedure to access.

Table 4-13 lists the `INFA_CTDGetData<datatype>()` function syntax and the datatype of the return value:

Table 4-13. Get Data Functions

Syntax	Return Value Datatype
<code>void* INFA_CTDGetDataVoid(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Data void pointer to the return value
<code>char* INFA_CTDGetDataStringM(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	String (MBCS)
<code>IUNICHAR* INFA_CTDGetDataStringU(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	String (Unicode)
<code>INFA_INT32 INFA_CTDGetDataINT32(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Integer
<code>double INFA_CTDGetDataDouble(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Double
<code>INFA_CT_RAWDATE INFA_CTDGetDataDate(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Raw date
<code>INFA_CT_RAWDEC18 INFA_CTDGetDataRawDec18(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Decimal BLOB (precision 18)
<code>INFA_CT_RAWDEC28 INFA_CTDGetDataRawDec28(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Decimal BLOB (precision 28)
<code>INFA_CT_DATETIME INFA_CTDGetDataDateTime(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Datetime

Set Data Function (Row-Based Mode)

Use the `INFA_CTSetsData()` function when you want the procedure to pass a value to an output port.

Use the following syntax:

```
INFA_STATUS INFA_CTSetData(INFA_CT_OUTPUTPORT_HANDLE dataHandle, void*  
data);
```

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Note: If you use the INFA_CTSetPassThruPort() function on an input/output port, do not use set the data or indicator for that port.

Indicator Functions (Row-Based Mode)

Use the indicator functions when you want the procedure to get the indicator for an input port or to set the indicator for an output port. The indicator for a port indicates whether the data is valid, null, or truncated.

PowerCenter provides the following indicator functions:

- ◆ **INFA_CHttpGetIndicator()**. Gets the indicator for an input port. Use the following syntax:

```
INFA_INDICATOR INFA_CHttpGetIndicator(INFA_CT_INPUTPORT_HANDLE dataHandle);
```

The return value datatype is INFA_INDICATOR. Use the following values for INFA_INDICATOR:

- **INFA_DATA_VALID**. Indicates the data is valid.
- **INFA_NULL_DATA**. Indicates a null value.
- **INFA_DATA_TRUNCATED**. Indicates the data has been truncated.

- ◆ **INFA_CTSetIndicator()**. Sets the indicator for an output port. Use the following syntax:

```
INFA_STATUS INFA_CTSetIndicator(INFA_CT_OUTPUTPORT_HANDLE dataHandle,  
INFA_INDICATOR indicator);
```

Argument	Datatype	Input/ Output	Description
dataHandle	INFA_CT_OUTPUTPORT_HANDLE	Input	Output port handle.
indicator	INFA_INDICATOR	Input	The indicator value for the output port. Use one of the following values: - INFA_DATA_VALID. Indicates the data is valid. - INFA_NULL_DATA. Indicates a null value. - INFA_DATA_TRUNCATED. Indicates the data has been truncated.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Note: If you use the INFA_CTSetPassThruPort() function on an input/output port, do not set the data or indicator for that port.

Length Functions

Use the length functions when you want the procedure to access the length of a string or binary input port, or to set the length of a binary or string output port.

Use the following length functions:

- ◆ **INFA_CTGetLength()**. Use this function for string and binary ports only. The Integration Service returns the length as the number of characters including trailing spaces. Use the following syntax:

```
INFA_UINT32 INFA_CTGetLength(INFA_CT_INPUTPORT_HANDLE dataHandle);
```

The return value datatype is INFA_UINT32. Use a value between zero and 2GB for the return value.

- ◆ **INFA_CTSLength()**. When the Custom transformation contains a binary or string output port, you must use this function to set the length of the data, including trailing spaces. Verify you the length you set for string and binary ports is not greater than the precision for that port. If you set the length greater than the port precision, you get unexpected results. For example, the session may fail.

Use the following syntax:

```
INFA_STATUS INFA_CTSLength(INFA_CT_OUTPUTPORT_HANDLE dataHandle,  
    IUINT32 length);
```

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Set Pass-Through Port Function

Use the INFA_CTSSetPassThruPort() function when you want the Integration Service to pass data from an input port to an output port without modifying the data. When you use the INFA_CTSSetPassThruPort() function, the Integration Service passes the data to the output port when it calls the input row notification function.

Consider the following rules and guidelines when you use the set pass-through port function:

- ◆ Only use this function in an initialization function.
- ◆ If the procedure includes this function, do not include the INFA_CTSGetData(), INFA_CTSLength, INFA_CTSSetIndicator(), or INFA_CTASetData() functions to pass data to the output port.
- ◆ In row-based mode, you can only include this function when the transformation scope is Row. When the transformation scope is Transaction or All Input, this function returns INFA_FAILURE.
- ◆ In row-based mode, when you use this function to output multiple rows for a given input row, every output row contains the data that is passed through from the input port.
- ◆ In array-based mode, you can only use this function for passive Custom transformations.

You must verify that the datatype, precision, and scale are the same for the input and output ports. The Integration Service fails the session if the datatype, precision, or scale are not the same for the input and output ports you specify in the INFA_CTSSetPassThruPort() function.

Use the following syntax:

```
INFA_STATUS INFA_CTSethPassThruPort(INFA_CT_OUTPUTPORT_HANDLE outputport,  
INFA_CT_INPUTPORT_HANDLE inputport)
```

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Output Notification Function

When you want the procedure to output a row to the Integration Service, use the INFA_CTOOutputNotification() function. Only include this function for active Custom transformations. For passive Custom transformations, the procedure outputs a row to the Integration Service when the input row notification function gives a return value. If the procedure calls this function for a passive Custom transformation, the Integration Service ignores the function.

Note: When the transformation scope is Row, you can only include this function in the input row notification function. If you include it somewhere else, it returns a failure.

Use the following syntax:

```
INFA_ROWSTATUS INFA_CTOOutputNotification(INFA_CT_OUTPUTGROUP_HANDLE  
group);
```

Argument	Datatype	Input/ Output	Description
group	INFA_CT_OUTPUT_GROUP_HANDLE	Input	Output group handle.

The return value datatype is INFA_ROWSTATUS. Use the following values for the return value:

- ◆ INFA_ROWSUCCESS. Indicates the function successfully processed the row of data.
- ◆ INFA_ROWERROR. Indicates the function encountered an error for the row of data. The Integration Service increments the internal error count.
- ◆ INFA_FATALError. Indicates the function encountered a fatal error for the row of data. The Integration Service fails the session.

Note: When the procedure code calls the INFA_CTOOutputNotification() function, you must verify that all pointers in an output port handle point to valid data. When a pointer does not point to valid data, the Integration Service might shut down unexpectedly.

Data Boundary Output Notification Function

Include the INFA_CTDatadbryOutputNotification() function when you want the procedure to output a commit or rollback transaction.

When you use this function, you must select the Generate Transaction property for this Custom transformation. If you do not select this property, the Integration Service fails the session.

Use the following syntax:

```
INFA_STATUS INFA_CTDaDataBdryOutputNotification(INFA_CT_PARTITION_HANDLE  
handle, INFA_CTDaDataBdryType dataBoundaryType);
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_PARTITION_HANDLE	Input	Handle name.
dataBoundaryType	INFA_CTDaDataBdryType	Input	The transaction type. Use the following values for the dataBoundaryType parameter: - eBT_COMMIT - eBT_ROLLBACK

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Error Functions

Use the error functions to access procedure errors. The Integration Service returns the most recent error.

PowerCenter provides the following error functions:

- ◆ **INFA_CTGetErrorMsgM()**. Gets the error message in MBCS. Use the following syntax:

```
const char* INFA_CTGetErrorMsg();
```
- ◆ **INFA_CTGetErrorMsgU()**. Gets the error message in Unicode. Use the following syntax:

```
const IUNICHAR* INFA_CTGetErrorMsgU();
```

Session Log Message Functions

Use the session log message functions when you want the procedure to log a message in the session log in either Unicode or MBCS.

PowerCenter provides the following session log message functions:

- ◆ **INFA_CTLLogMessageU()**. Logs a message in Unicode.

Use the following syntax:

```
void INFA_CTLLogMessageU(INFA_CT_ErrorSeverityLevel errorSeverityLevel,  
                         INFA_UNICHAR* msg)
```

Argument	Datatype	Input/ Output	Description
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	Input	Severity level of the error message that you want the Integration Service to write in the session log. Use the following values for the errorSeverityLevel parameter: - EESL_LOG - EESL_DEBUG - EESL_ERROR
msg	INFA_UNICHAR*	Input	Enter the text of the message in Unicode in quotes.

- ◆ **INFA_CTLLogMessageM()**. Logs a message in MBCS.

Use the following syntax:

```
void INFA_CTLLogMessageM(INFA_CT_ErrorSeverityLevel errorSeverityLevel,  
                         char* msg)
```

Argument	Datatype	Input/ Output	Description
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	Input	Severity level of the error message that you want the Integration Service to write in the session log. Use the following values for the errorSeverityLevel parameter: - EESL_LOG - EESL_DEBUG - EESL_ERROR
msg	char*	Input	Enter the text of the message in MBCS in quotes.

Increment Error Count Function

Use the INFA_CTIIncrementErrorCount() function when you want to increase the error count for the session.

Use the following syntax:

```
INFA_STATUS INFA_CTIIncrementErrorCount(INFA_CT_PARTITION_HANDLE  
transformation, size_t nErrors, INFA_STATUS* pStatus);
```

Argument	Datatype	Input/ Output	Description
transformation	INFA_CT_PARTITION_HANDLE	Input	Partition handle.
nErrors	size_t	Input	Integration Service increments the error count by nErrors for the given transformation instance.
pStatus	INFA_STATUS*	Input	Integration Service uses INFA_FAILURE for the pStatus parameter when the error count exceeds the error threshold and fails the session.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Is Terminated Function

Use the INFA_CTIIsTerminated() function when you want the procedure to check if the PowerCenter Client has requested the Integration Service to stop the session. You might call this function if the procedure includes a time-consuming process.

Use the following syntax:

```
INFA_CTTerminateType INFA_CTIIsTerminated(INFA_CT_PARTITION_HANDLE  
handle);
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_PARTITION_HANDLE	input	Partition handle.

The return value datatype is INFA_CTTerminateType. The Integration Service returns one of the following values:

- ◆ **eTT_NOTTERMINATED.** Indicates the PowerCenter Client has not requested to stop the session.
- ◆ **eTT_ABORTED.** Indicates the Integration Service aborted the session.
- ◆ **eTT_STOPPED.** Indicates the Integration Service failed the session.

Blocking Functions

When the Custom transformation contains multiple input groups, you can write code to block the incoming data on an input group. For more information about blocking data, see “Blocking Input Data” on page 70.

Consider the following rules when you use the blocking functions:

- ◆ You can block at most $n-1$ input groups.
- ◆ You cannot block an input group that is already blocked.
- ◆ You cannot block an input group when it receives data from the same source as another input group.
- ◆ You cannot unblock an input group that is already unblocked.

PowerCenter provides the following blocking functions:

- ◆ **INFA_CTBlockInputFlow()**. Allows the procedure to block an input group.

Use the following syntax:

```
INFA_STATUS INFA_CTBlockInputFlow(INFA_CT_INPUTGROUP_HANDLE group);
```

- ◆ **INFA_CTUnblockInputFlow()**. Allows the procedure to unblock an input group.

Use the following syntax:

```
INFA_STATUS INFA_CTUnblockInputFlow(INFA_CT_INPUTGROUP_HANDLE group);
```

Argument	Datatype	Input/ Output	Description
group	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Verify Blocking

When you use the INFA_CTBlockInputFlow() and INFA_CTUnblockInputFlow() functions in the procedure code, verify the procedure checks whether or not the Integration Service allows the Custom transformation to block incoming data. To do this, check the value of the INFA_CT_TRANS_MAY_BLOCK_DATA propID using the INFA_CTCGetInternalPropertyBool() function.

When the value of the INFA_CT_TRANS_MAY_BLOCK_DATA propID is FALSE, the procedure should either not use the blocking functions, or it should return a fatal error and stop the session.

If the procedure code uses the blocking functions when the Integration Service does not allow the Custom transformation to block data, the Integration Service might fail the session.

Pointer Functions

Use the pointer functions when you want the Integration Service to create and access pointers to an object or a structure.

PowerCenter provides the following pointer functions:

- ◆ **INFA_CTGetUserDefinedPtr()**. Allows the procedure to access an object or structure during run time.

Use the following syntax:

```
void* INFA_CTGetUserDefinedPtr(INFA_CT_HANDLE handle)
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_HANDLE	Input	Handle name.

- ◆ **INFA_CTSUserDefinedPtr()**. Allows the procedure to associate an object or a structure with any handle the Integration Service provides. To reduce processing overhead, include this function in the initialization functions.

Use the following syntax:

```
void INFA_CTSUserDefinedPtr(INFA_CT_HANDLE handle, void* pPtr)
```

Argument	Datatype	Input/ Output	Description
handle	INFA_CT_HANDLE	Input	Handle name.
pPtr	void*	Input	User pointer.

You must substitute a valid handle for INFA_CT_HANDLE.

Change String Mode Function

When the Integration Service runs in Unicode mode, it passes data to the procedure in UCS-2 by default. When it runs in ASCII mode, it passes data in ASCII by default. Use the INFA_CTCChangeStringMode() function if you want to change the default string mode for the procedure. When you change the default string mode to MBCS, the Integration Service passes data in the Integration Service code page. Use the INFA_CTSSetDataCodePageID() function if you want to change the code page. For more information about changing the code page ID, see “Set Data Code Page Function” on page 127.

When a procedure includes the INFA_CTCChangeStringMode() function, the Integration Service changes the string mode for all ports in each Custom transformation that use this particular procedure.

Use the change string mode function in the initialization functions.

Use the following syntax:

```
INFA_STATUS INFA_CTChangeStringMode(INFA_CT_PROCEDURE_HANDLE procedure,  
INFA_CTSStringMode stringMode);
```

Argument	Datatype	Input/ Output	Description
procedure	INFA_CT_PROCEDURE_HANDLE	Input	Procedure handle name.
stringMode	INFA_CTSStringMode	Input	Specifies the string mode that you want the Integration Service to use. Use the following values for the stringMode parameter: - eASM_UNICODE. Use this when the Integration Service runs in ASCII mode and you want the procedure to access data in Unicode. - eASM_MBCS. Use this when the Integration Service runs in Unicode mode and you want the procedure to access data in MBCS.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Set Data Code Page Function

Use the INFA_CTSGetDataCodePageID() when you want the Integration Service to pass data to the Custom transformation in a code page other than the Integration Service code page.

Use the set data code page function in the procedure initialization function.

Use the following syntax:

```
INFA_STATUS INFA_CTSGetDataCodePageID(INFA_CT_TRANSFORMATION_HANDLE  
transformation, int dataCodePageID);
```

Argument	Datatype	Input/ Output	Description
transformation	INFA_CT_TRANSFORMATION_HANDLE	Input	Transformation handle name.
dataCodePageID	int	Input	Specifies the code page you want the Integration Service to pass data in. For valid values for the dataCodePageID parameter, see "Code Pages" in the <i>Administrator Guide</i> .

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Row Strategy Functions (Row-Based Mode)

The row strategy functions allow you to access and configure the update strategy for each row.

PowerCenter provides the following row strategy functions:

- ◆ **INFA_CTGetRowStrategy()**. Allows the procedure to get the update strategy for a row.

Use the following syntax:

```
INFA_STATUS INFA_CTGetRowStrategy(INFA_CT_INPUTGROUP_HANDLE group,  
INFA_CTCUpdateStrategy updateStrategy);
```

Argument	Datatype	Input/ Output	Description
group	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.
updateStrategy	INFA_CT_UPDATESTRATEGY	Input	Update strategy for the input port. The Integration Service uses the following values: - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

- ◆ **INFA_CTSRowStrategy()**. Sets the update strategy for each row. This overrides the INFA_CTCChangeDefaultRowStrategy function.

Use the following syntax:

```
INFA_STATUS INFA_CTSRowStrategy(INFA_CT_OUTPUTGROUP_HANDLE group,  
INFA_CTC_UPDATESTRATEGY updateStrategy);
```

Argument	Datatype	Input/ Output	Description
group	INFA_CT_OUTPUTGROUP_HANDLE	Input	Output group handle.
updateStrategy	INFA_CT_UPDATESTRATEGY	Input	Update strategy you want to set for the output port. Use one of the following values: - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Change Default Row Strategy Function

By default, the row strategy for a Custom transformation is pass-through when the transformation scope is Row. When the transformation scope is Transaction or All Input, the row strategy is the same value as the Treat Source Rows As session property by default.

For example, in a mapping you have an Update Strategy transformation followed by a Custom transformation with Row transformation scope. The Update Strategy transformation flags the rows for update, insert, or delete. When the Integration Service passes a row to the Custom transformation, the Custom transformation retains the flag since its row strategy is pass-through.

However, you can change the row strategy of a Custom transformation with PowerCenter. Use the INFA_CTChangeDefaultRowStrategy() function to change the default row strategy at the transformation level. For example, when you change the default row strategy of a Custom transformation to insert, the Integration Service flags all the rows that pass through this transformation for insert.

Note: The Integration Service returns INFA_FAILURE if the session is not in data-driven mode.

Use the following syntax:

```
INFA_STATUS INFA_CTChangeDefaultRowStrategy(INFA_CT_TRANSFORMATION_HANDLE  
transformation, INFA_CT_DefaultUpdateStrategy defaultUpdateStrategy);
```

Argument	Datatype	Input/ Output	Description
transformation	INFA_CT_TRANSFORMATION_HANDLE	Input	Transformation handle.
defaultUpdateStrategy	INFA_CT_DefaultUpdateStrategy	Input	Specifies the row strategy you want the Integration Service to use for the Custom transformation. - eDUS_PASSTHROUGH. Flags the row for passthrough. - eDUS_INSERT. Flags rows for insert. - eDUS_UPDATE. Flags rows for update. - eDUS_DELETE. Flags rows for delete.

The return value datatype is INFA_STATUS. Use INFA_SUCCESS and INFA_FAILURE for the return value.

Array-Based API Functions

The array-based functions are API functions you use when you change the data access mode to array-based. For more information about changing the data access mode, see “Set Data Access Mode Function” on page 104.

Informatica provides the following groups of array-based API functions:

- ◆ **Maximum number of rows.** See “Maximum Number of Rows Functions” on page 130.
- ◆ **Number of rows.** See “Number of Rows Functions” on page 131.
- ◆ **Is row valid.** See “Is Row Valid Function” on page 132.
- ◆ **Data handling (array-based mode).** See “Data Handling Functions (Array-Based Mode)” on page 132.
- ◆ **Row strategy.** See “Row Strategy Functions (Array-Based Mode)” on page 135.
- ◆ **Set input error row.** See “Set Input Error Row Functions” on page 136.

Maximum Number of Rows Functions

By default, the Integration Service allows a maximum number of rows in an input block and an output block. However, you can change the maximum number of rows allowed in an output block.

Use the INFA_CTAGetInputNumRowsMax() and INFA_CTAGetOutputNumRowsMax() functions to determine the maximum number of rows in input and output blocks. Use the values these functions return to determine the buffer size if the procedure needs a buffer.

You can set the maximum number of rows in the output block using the INFA_CTASetOutputRowMax() function. You might use this function if you want the procedure to use a larger or smaller buffer.

You can only call these functions in an initialization function.

PowerCenter provides the following functions to determine and set the maximum number of rows in blocks:

- ◆ **INFA_CTAGetInputNumRowsMax().** Use this function to determine the maximum number of rows allowed in an input block.

Use the following syntax:

```
IINT32 INFA_CTAGetInputRowMax( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

Argument	Datatype	Input/ Output	Description
inputgroup	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.

- ◆ **INFA_CTAGetOutputNumRowsMax().** Use this function to determine the maximum number of rows allowed in an output block.

Use the following syntax:

```
IINT32 INFA_CTAGetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE outputgroup );
```

Argument	Datatype	Input/ Output	Description
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	Input	Output group handle.

- ◆ **INFA_CTASetOutputRowMax()**. Use this function to set the maximum number of rows allowed in an output block.

Use the following syntax:

```
INFA_STATUS INFA_CTASetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE  
outputgroup, INFA_INT32 nRowMax );
```

Argument	Datatype	Input/ Output	Description
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	Input	Output group handle.
nRowMax	INFA_INT32	Input	Maximum number of rows you want to allow in an output block. You must enter a positive number. The function returns a fatal error when you use a non-positive number, including zero.

Number of Rows Functions

Use the number of rows functions to determine the number of rows in an input block, or to set the number of rows in an output block for the specified input or output group.

PowerCenter provides the following number of rows functions:

- ◆ **INFA_CTAGetNumRows()**. You can determine the number of rows in an input block.

Use the following syntax:

```
INFA_INT32 INFA_CTAGetNumRows( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

Argument	Datatype	Input/ Output	Description
inputgroup	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.

- ◆ **INFA_CTASetNumRows()**. You can set the number of rows in an output block. Call this function before you call the output notification function.

Use the following syntax:

```
void INFA_CTASetNumRows( INFA_CT_OUTPUTGROUP_HANDLE outputgroup,  
INFA_INT32 nRows );
```

Argument	Datatype	Input/ Output	Description
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	Input	Output port handle.
nRows	INFA_INT32	Input	Number of rows you want to define in the output block. You must enter a positive number. The Integration Service fails the output notification function when specify a non-positive number.

Is Row Valid Function

Some rows in a block may be dropped, filter, or error rows. Use the INFA_CTAIsRowValid() function to determine if a row in a block is valid. This function returns INFA_TRUE when a row is valid.

Use the following syntax:

```
INFA_BOOLINFA_CTAIsRowValid( INFA_CT_INPUTGROUP_HANDLE inputgroup,  
INFA_INT32 iRow );
```

Argument	Datatype	Input/ Output	Description
inputgroup	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.
iRow	INFA_INT32	Input	Index number of the row in the block. The index is zero-based. You must verify the procedure only passes an index number that exists in the data block. If you pass an invalid value, the Integration Service shuts down unexpectedly.

Data Handling Functions (Array-Based Mode)

When the Integration Service calls the p_<proc_name>_inputRowNotification() function, it notifies the procedure that the procedure can access a row or block of data. However, to get data from the input port, modify it, and set data in the output port in array-based mode, you must use the array-based data handling functions in the input row notification function.

Include the INFA_CTAGetData<datatype>() function to get the data from the input port and INFA_CTASetData() function to set the data in the output port. Include the INFA_CTAGetIndicator() function if you want the procedure to verify before you get the data if the port has a null value or an empty string.

PowerCenter provides the following data handling functions for the array-based data access mode:

- ◆ **INFA_CTAGetData<datatype>()**. For more information, see “Get Data Functions (Array-Based Mode)” on page 133.
- ◆ **INFA_CTAGetIndicator()**. For more information, see “Get Indicator Function (Array-Based Mode)” on page 134.
- ◆ **INFA_CTASetData()**. For more information, see “Set Data Function (Array-Based Mode)” on page 134.

Get Data Functions (Array-Based Mode)

Use the INFA_CTAGetData<datatype>() functions to retrieve data for the port the function specifies. You must modify the function name depending on the datatype of the port you want the procedure to access. The Integration Service passes the length of the data in the array-based get data functions.

Table 4-14 lists the INFA_CTAGetData<datatype>() function syntax and the datatype of the return value:

Table 4-14. Get Data Functions (Array-Based Mode)

Syntax	Return Value Datatype
<code>void* INFA_CTAGetDataVoid(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	Data void pointer to the return value
<code>char* INFA_CTAGetDataStringM(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	String (MBCS)
<code>IUNICHAR* INFA_CTAGetDataStringU(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	String (Unicode)
<code>INFA_INT32 INFA_CTAGetDataINT32(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Integer
<code>double INFA_CTAGetDataDouble(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Double
<code>INFA_CT_RAWDATETIME INFA_CTAGetDataRawDate(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Raw date
<code>INFA_CT_DATETIME INFA_CTAGetDataDateTime(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Datetime
<code>INFA_CT_RAWDEC18 INFA_CTAGetDataRawDec18(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Decimal BLOB (precision 18)
<code>INFA_CT_RAWDEC28 INFA_CTAGetDataRawDec28(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Decimal BLOB (precision 28)

Get Indicator Function (Array-Based Mode)

Use the get indicator function when you want the procedure to verify if the input port has a null value.

Use the following syntax:

```
INFA_INDICATOR INFA_CTAGetIndicator( INFA_CT_INPUTPORT_HANDLE inputport,  
INFA_INT32 iRow );
```

Argument	Datatype	Input/ Output	Description
inputport	INFA_CT_INPUTPORT_HANDLE	Input	Input port handle.
iRow	INFA_INT32	Input	Index number of the row in the block. The index is zero-based. You must verify the procedure only passes an index number that exists in the data block. If you pass an invalid value, the Integration Service shuts down unexpectedly.

The return value datatype is INFA_INDICATOR. Use the following values for INFA_INDICATOR:

- ◆ **INFA_DATA_VALID.** Indicates the data is valid.
- ◆ **INFA_NULL_DATA.** Indicates a null value.
- ◆ **INFA_DATA_TRUNCATED.** Indicates the data has been truncated.

Set Data Function (Array-Based Mode)

Use the set data function when you want the procedure to pass a value to an output port. You can set the data, the length of the data, if applicable, and the indicator for the output port you specify. You do not use separate functions to set the length or indicator for the output port.

Use the following syntax:

```
void INFA_CTASetData( INFA_CT_OUTPUTPORT_HANDLE outputport, INFA_INT32  
iRow, void* pData, INFA_UINT32 nLength, INFA_INDICATOR indicator);
```

Argument	Datatype	Input/ Output	Description
outputport	INFA_CT_OUTPUTPORT_HANDLE	Input	Output port handle.
iRow	INFA_INT32	Input	Index number of the row in the block. The index is zero-based. You must verify the procedure only passes an index number that exists in the data block. If you pass an invalid value, the Integration Service shuts down unexpectedly.
pData	void*	Input	Pointer to the data.

Argument	Datatype	Input/ Output	Description
nLength	INFA_UINT32	Input	<p>Length of the port. Use for string and binary ports only.</p> <p>You must verify the function passes the correct length of the data. If the function passes a different length, the output notification function returns failure for this port.</p> <p>Verify the length you set for string and binary ports is not greater than the precision for the port. If you set the length greater than the port precision, you get unexpected results. For example, the session may fail.</p>
indicator	INFA_INDICATOR	Input	<p>Indicator value for the output port. Use one of the following values:</p> <ul style="list-style-type: none"> - INFA_DATA_VALID. Indicates the data is valid. - INFA_NULL_DATA. Indicates a null value. - INFA_DATA_TRUNCATED. Indicates the data has been truncated.

Row Strategy Functions (Array-Based Mode)

The array-based row strategy functions allow you to access and configure the update strategy for each row in a block.

PowerCenter provides the following row strategy functions:

- ◆ **INFA_CTAGetRowStrategy()**. Allows the procedure to get the update strategy for a row in a block.

Use the following syntax:

```
INFA_CT_UPDATESTRATEGY INFA_CTAGetRowStrategy( INFA_CT_INPUTGROUP_HANDLE
inputgroup, INFA_INT32 iRow);
```

Argument	Datatype	Input/ Output	Description
inputgroup	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.
iRow	INFA_INT32	Input	<p>Index number of the row in the block. The index is zero-based.</p> <p>You must verify the procedure only passes an index number that exists in the data block. If you pass an invalid value, the Integration Service shuts down unexpectedly.</p>

- ◆ **INFA_CTASetRowStrategy()**. Sets the update strategy for a row in a block.

Use the following syntax:

```
void INFA_CTASetRowStrategy( INFA_CT_OUTPUTGROUP_HANDLE outputgroup,
    INFA_INT32 iRow, INFA_CT_UPDATESTRATEGY updateStrategy );
```

Argument	Datatype	Input/ Output	Description
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	Input	Output group handle.
iRow	INFA_INT32	Input	Index number of the row in the block. The index is zero-based. You must verify the procedure only passes an index number that exists in the data block. If you pass an invalid value, the Integration Service shuts down unexpectedly.
updateStrategy	INFA_CT_UPDATESTRATEGY	Input	Update strategy for the port. Use one of the following values: - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

Set Input Error Row Functions

When you use array-based access mode, you cannot return INFA_ROWERROR in the input row notification function. Instead, use the set input error row functions to notify the Integration Service that a particular input row has an error.

PowerCenter provides the following set input row functions in array-based mode:

- ◆ **INFA_CTASetInputErrorRowM()**. You can notify the Integration Service that a row in the input block has an error and to output an MBCS error message to the session log.

Use the following syntax:

```
INFA_STATUS INFA_CTASetInputErrorRowM( INFA_CT_INPUTGROUP_HANDLE
    inputGroup, INFA_INT32 iRow, size_t nErrors, INFA_MBCSCHAR* sErrMsg );
```

Argument	Datatype	Input/ Output	Description
inputGroup	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.
iRow	INFA_INT32	Input	Index number of the row in the block. The index is zero-based. You must verify the procedure only passes an index number that exists in the data block. If you pass an invalid value, the Integration Service shuts down unexpectedly.

Argument	Datatype	Input/ Output	Description
nErrors	size_t	Input	Use this parameter to specify the number of errors this input row has caused.
sErrMsg	INFA_MBCSCHAR*	Input	MBCS string containing the error message you want the function to output. You must enter a null-terminated string. This parameter is optional. When you include this argument, the Integration Service prints the message in the session log, even when you enable row error logging.

- ◆ **INFA_CTASetInputErrorRowU()**. You can notify the Integration Service that a row in the input block has an error and to output a Unicode error message to the session log.

Use the following syntax:

```
INFA_STATUS INFA_CTASetInputErrorRowU( INFA_CT_INPUTGROUP_HANDLE
inputGroup, INFA_INT32 iRow, size_t nErrors, INFA_UNICHAR* sErrMsg );
```

Argument	Datatype	Input/ Output	Description
inputGroup	INFA_CT_INPUTGROUP_HANDLE	Input	Input group handle.
iRow	INFA_INT32	Input	Index number of the row in the block. The index is zero-based. You must verify the procedure only passes an index number that exists in the data block. If you pass an invalid value, the Integration Service shuts down unexpectedly.
nErrors	size_t	Input	Use this parameter to specify the number of errors this output row has caused.
sErrMsg	INFA_UNICHAR*	Input	Unicode string containing the error message you want the function to output. You must enter a null-terminated string. This parameter is optional. When you include this argument, the Integration Service prints the message in the session log, even when you enable row error logging.

Java API Functions

Information forthcoming.

C++ API Functions

Information forthcoming.

Chapter 5

Expression Transformation

This chapter includes the following topics:

- ◆ Overview, 142
- ◆ Creating an Expression Transformation, 143

Overview

Transformation type:

Passive
Connected

Use the Expression transformation to calculate values in a single row before you write to the target. For example, you might need to adjust employee salaries, concatenate first and last names, or convert strings to numbers. Use the Expression transformation to perform any non-aggregate calculations. You can also use the Expression transformation to test conditional statements before you output the results to target tables or other transformations.

Note: To perform calculations involving multiple rows, such as sums or averages, use the Aggregator transformation. Unlike the Expression transformation, the Aggregator lets you group and sort data. For more information, see “Aggregator Transformation” on page 37.

Calculating Values

To use the Expression transformation to calculate values for a single row, you must include the following ports:

- ◆ **Input or input/output ports for each value used in the calculation.** For example, when calculating the total price for an order, determined by multiplying the unit price by the quantity ordered, the input or input/output ports. One port provides the unit price and the other provides the quantity ordered.
- ◆ **Output port for the expression.** You enter the expression as a configuration option for the output port. The return value for the output port needs to match the return value of the expression. For more information about entering expressions, see “Working with Expressions” on page 10. Expressions use the transformation language, which includes SQL-like functions, to perform calculations.

Adding Multiple Calculations

You can enter multiple expressions in a single Expression transformation. As long as you enter only one expression for each output port, you can create any number of output ports in the transformation. In this way, use one Expression transformation rather than creating separate transformations for each calculation that requires the same set of data.

For example, you might want to calculate several types of withholding taxes from each employee paycheck, such as local and federal income tax, Social Security and Medicare. Since all of these calculations require the employee salary, the withholding category, and/or the corresponding tax rate, you can create one Expression transformation with the salary and withholding category as input/output ports and a separate output port for each necessary calculation.

Creating an Expression Transformation

Use the following procedure to create an Expression transformation.

To create an Expression transformation:

1. In the Mapping Designer, click Transformation > Create. Select the Expression transformation. Enter a name for it (the convention is EXP_*TransformationName*) and click OK.
2. Create the input ports.

If you have the input transformation available, you can select Link Columns from the Layout menu and then drag each port used in the calculation into the Expression transformation. With this method, the Designer copies the port into the new transformation and creates a connection between the two ports. Or, you can open the transformation and create each port manually.

Note: If you want to make this transformation reusable, you must create each port manually within the transformation.

3. Repeat the previous step for each input port you want to add to the expression.
4. Create the output ports (O) you need, making sure to assign a port datatype that matches the expression return value. The naming convention for output ports is OUT_*PORTNAME*.
5. Click the small button that appears in the Expression section of the dialog box and enter the expression in the Expression Editor.

To prevent typographic errors, where possible, use the listed port names and functions.

If you select a port name that is not connected to the transformation, the Designer copies the port into the new transformation and creates a connection between the two ports.

Port names used as part of an expression in an Expression transformation follow stricter rules than port names in other types of transformations:

- ♦ A port name must begin with a single- or double-byte letter or single- or double-byte underscore (_).
- ♦ It can contain any of the following single- or double-byte characters: a letter, number, underscore (_), \$, #, or @.

6. Check the expression syntax by clicking Validate.

If necessary, make corrections to the expression and check the syntax again. Then save the expression and exit the Expression Editor.

7. Connect the output ports to the next transformation or target.
8. Select a tracing level on the Properties tab to determine the amount of transaction detail reported in the session log file.
9. Click Repository > Save.

Chapter 6

External Procedure Transformation

This chapter includes the following topics:

- ◆ Overview, 146
- ◆ Developing COM Procedures, 149
- ◆ Developing Informatica External Procedures, 159
- ◆ Distributing External Procedures, 169
- ◆ Development Notes, 171
- ◆ Service Process Variables in Initialization Properties, 180
- ◆ External Procedure Interfaces, 181

Overview

Transformation type:

Passive

Connected/Unconnected

External Procedure transformations operate in conjunction with procedures you create outside of the Designer interface to extend PowerCenter functionality.

Although the standard transformations provide you with a wide range of options, there are occasions when you might want to extend the functionality provided with PowerCenter. For example, the range of standard transformations, such as Expression and Filter transformations, may not provide the functionality you need. If you are an experienced programmer, you may want to develop complex functions within a dynamic link library (DLL) or UNIX shared library, instead of creating the necessary Expression transformations in a mapping.

To obtain this kind of extensibility, use the Transformation Exchange (TX) dynamic invocation interface built into PowerCenter. Using TX, you can create an Informatica External Procedure transformation and bind it to an external procedure that you have developed. You can bind External Procedure transformations to two kinds of external procedures:

- ◆ COM external procedures (available on Windows only)
- ◆ Informatica external procedures (available on Windows, AIX, HP-UX, Linux, and Solaris)

To use TX, you must be an experienced C, C++, or Visual Basic programmer.

Use multi-threaded code in external procedures.

Code Page Compatibility

When the Integration Service runs in ASCII mode, the external procedure can process data in 7-bit ASCII.

When the Integration Service runs in Unicode mode, the external procedure can process data that is two-way compatible with the Integration Service code page. For information about accessing the Integration Service code page, see “Code Page Access Functions” on page 185.

Configure the Integration Service to run in Unicode mode if the external procedure DLL or shared library contains multibyte characters. External procedures must use the same code page as the Integration Service to interpret input strings from the Integration Service and to create output strings that contain multibyte characters.

Configure the Integration Service to run in either ASCII or Unicode mode if the external procedure DLL or shared library contains ASCII characters only.

External Procedures and External Procedure Transformations

There are two components to TX: *external procedures* and *External Procedure transformations*.

As its name implies, an *external procedure* exists separately from the Integration Service. It consists of C, C++, or Visual Basic code written by a user to define a transformation. This code is compiled and linked into a DLL or shared library, which is loaded by the Integration Service at runtime. An external procedure is “bound” to an External Procedure transformation.

An *External Procedure transformation* is created in the Designer. It is an object that resides in the Informatica repository and serves several purposes:

1. It contains the metadata describing the following external procedure. It is through this metadata that the Integration Service knows the “signature” (number and types of parameters, type of return value, if any) of the external procedure.
2. It allows an external procedure to be referenced in a mapping. By adding an instance of an External Procedure transformation to a mapping, you call the external procedure bound to that transformation.

Note: You can create a connected or unconnected External Procedure.

3. When you develop Informatica external procedures, the External Procedure transformation provides the information required to generate Informatica external procedure stubs.

External Procedure Transformation Properties

Create reusable External Procedure transformations in the Transformation Developer, and add instances of the transformation to mappings. You cannot create External Procedure transformations in the Mapping Designer or Mapplet Designer.

External Procedure transformations return one or no output rows per input row.

On the Properties tab of the External Procedure transformation, only enter ASCII characters in the Module/Programmatic Identifier and Procedure Name fields. You cannot enter multibyte characters in these fields. On the Ports tab of the External Procedure transformation, only enter ASCII characters for the port names. You cannot enter multibyte characters for External Procedure transformation port names.

Pipeline Partitioning

If you purchase the Partitioning option with PowerCenter, you can increase the number of partitions in a pipeline to improve session performance. Increasing the number of partitions allows the Integration Service to create multiple connections to sources and process partitions of source data concurrently.

When you create a session, the Workflow Manager validates each pipeline in the mapping for partitioning. You can specify multiple partitions in a pipeline if the Integration Service can maintain data consistency when it processes the partitioned data.

Use the Is Partitionable property on the Properties tab to specify whether or not you can create multiple partitions in the pipeline. For more information about partitioning External Procedure transformations, see “Working with Partition Points” in the *Workflow Administration Guide*.

COM Versus Informatica External Procedures

Table 6-1 describes the differences between COM and Informatica external procedures:

Table 6-1. Differences Between COM and Informatica External Procedures

COM	Informatica
Technology	Uses COM technology
Operating System	Runs on Windows only
Language	C, C++, VC++, VB, Perl, VJ++
	Only C++

The BankSoft Example

The following sections use an example called BankSoft to illustrate how to develop COM and Informatica procedures. The BankSoft example uses a financial function, FV, to illustrate how to develop and call an external procedure. The FV procedure calculates the future value of an investment based on regular payments and a constant interest rate.

Developing COM Procedures

You can develop COM external procedures using Microsoft Visual C++ or Visual Basic. The following sections describe how to create COM external procedures using Visual C++ and how to create COM external procedures using Visual Basic.

Steps for Creating a COM Procedure

To create a COM external procedure, complete the following steps:

1. Using Microsoft Visual C++ or Visual Basic, create a project.
2. Define a class with an *IDispatch* interface.
3. Add a method to the interface. This method is the external procedure that will be invoked from inside the Integration Service.
4. Compile and link the class into a dynamic link library.
5. Register the class in the local Windows registry.
6. Import the COM procedure in the Transformation Developer.
7. Create a mapping with the COM procedure.
8. Create a session using the mapping.

COM External Procedure Server Type

The Integration Service only supports in-process COM servers (that is, COM servers with *Server Type: Dynamic Link Library*). This is done to enhance performance. It is more efficient when processing large amounts of data to process the data in the same process, instead of forwarding it to a separate process on the same machine or a remote machine.

Using Visual C++ to Develop COM Procedures

C++ developers can use Visual C++ version 5.0 or later to develop COM procedures. The first task is to create a project.

Step 1. Create an ATL COM AppWizard Project

1. Launch Visual C++ and click File > New.
2. In the dialog box that appears, select the Projects tab.
3. Enter the project name and location.

In the BankSoft example, you enter *COM_VC_Banksoft* as the project name, and *c:\COM_VC_Banksoft* as the directory.

4. Select the *ATL COM AppWizard* option in the projects list box and click OK.

A wizard used to create COM projects in Visual C++ appears.

5. Set the Server Type to Dynamic Link Library, select the *Support MFC* option, and click Finish.
The final page of the wizard appears.
6. Click OK to return to Visual C++.
7. Add a class to the new project.
8. On the next page of the wizard, click the OK button. The Developer Studio creates the basic project files.

Step 2. Add an ATL Object to a Project

1. In the Workspace window, select the Class View tab, right-click the tree item *COM_VC_BankSoft.BSoftFin* classes, and choose New ATL Object from the local menu that appears.
2. Highlight the Objects item in the left list box and select *Simple Object* from the list of object types.
3. Click Next.
4. In the Short Name field, enter a short name for the class you want to create.

In the BankSoft example, use the name *BSoftFin*, since you are developing a financial function for the fictional company BankSoft. As you type into the Short Name field, the wizard fills in suggested names in the other fields.

5. Enter the programmatic identifier for the class.

In the BankSoft example, change the ProgID (programmatic identifier) field to *COM_VC_BankSoft.BSoftFin*.

A programmatic identifier, or ProgID, is the human-readable name for a class. Internally, classes are identified by numeric CLSID's. For example:

{33B17632-1D9F-11D1-8790-0000C044ACF9}

The standard format of a ProgID is *Project.Class[.Version]*. In the Designer, you refer to COM classes through ProgIDs.

6. Select the Attributes tab and set the threading model to *Free*, the interface to *Dual*, and the aggregation setting to *No*.
7. Click OK.

Now that you have a basic class definition, you can add a method to it.

Step 3. Add the Required Methods to the Class

1. Return to the Classes View tab of the Workspace Window.
2. Expand the tree view.

For the BankSoft example, you expand *COM_VC_BankSoft*.

3. Right-click the newly-added class.

In the BankSoft example, you right-click the *IBSoftFin* tree item.

4. Click the Add Method menu item and enter the name of the method.

In the BankSoft example, you enter *FV*.

5. In the Parameters field, enter the signature of the method.

For *FV*, enter the following:

```
[in] double Rate,  
[in] long nPeriods,  
[in] double Payment,  
[in] double PresentValue,  
[in] long PaymentType,  
[out, retval] double* FV
```

This signature is expressed in terms of the Microsoft Interface Description Language (MIDL). For a complete description of MIDL, see the MIDL language reference. Note that:

- ♦ [in] indicates that the parameter is an input parameter.
- ♦ [out] indicates that the parameter is an output parameter.
- ♦ [out, retval] indicates that the parameter is the return value of the method.

Also, note that all [out] parameters are passed by reference. In the BankSoft example, the parameter *FV* is a double.

6. Click OK.

The Developer Studio adds to the project a stub for the method you added.

Step 4. Fill Out the Method Stub with an Implementation

1. In the BankSoft example, return to the Class View tab of the Workspace window and expand the *COM_VC_BankSoft* classes item.
2. Expand the *CBSofFin* item.
3. Expand the *IBSoftFin* item under the above item.
4. Right-click the *FV* item and choose Go to Definition.
5. Position the cursor in the edit window on the line after the TODO comment and add the following code:

```
double v = pow((1 + Rate), nPeriods);  
*FV = -(  
    (PresentValue * v) +  
    (Payment * (1 + (Rate * PaymentType))) * ((v - 1) / Rate)  
);
```

Since you refer to the pow function, you have to add the following preprocessor statement after all other include statements at the beginning of the file:

```
#include <math.h>
```

The final step is to build the DLL. When you build it, you register the COM procedure with the Windows registry.

Step 5. Build the Project

1. Pull down the Build menu.
2. Select Rebuild All.

As Developer Studio builds the project, it generates the following output:

```
-----Configuration: COM_VC_BankSoft - Win32 Debug-----
Performing MIDL step
Microsoft (R) MIDL Compiler Version 3.01.75
Copyright (c) Microsoft Corp 1991-1997. All rights reserved.
Processing .\COM_VC_BankSoft.idl
COM_VC_BankSoft.idl
Processing C:\msdev\VC\INCLUDE\oaidl.idl
oaidl.idl
Processing C:\msdev\VC\INCLUDE\objidl.idl
objidl.idl
Processing C:\msdev\VC\INCLUDE\unknwn.idl
unknwn.idl
Processing C:\msdev\VC\INCLUDE\wtypes.idl
wtypes.idl
Processing C:\msdev\VC\INCLUDE\ocidl.idl
ocidl.idl
Processing C:\msdev\VC\INCLUDE\oleidl.idl
oleidl.idl
Compiling resources...
Compiling...
StdAfx.cpp
Compiling...
COM_VC_BankSoft.cpp
BSoftFin.cpp
Generating Code...
Linking...
Creating library Debug/COM_VC_BankSoft.lib and object Debug/
COM_VC_BankSoft.exp
Registering ActiveX Control...
RegSvr32: DllRegisterServer in .\Debug\COM_VC_BankSoft.dll succeeded.

COM_VC_BankSoft.dll - 0 error(s), 0 warning(s)
```

Notice that Visual C++ compiles the files in the project, links them into a dynamic link library (DLL) called COM_VC_BankSoft.DLL, and registers the COM (ActiveX) class COM_VC_BankSoft.BSoftFin in the local registry.

Once the component is registered, it is accessible to the Integration Service running on that host.

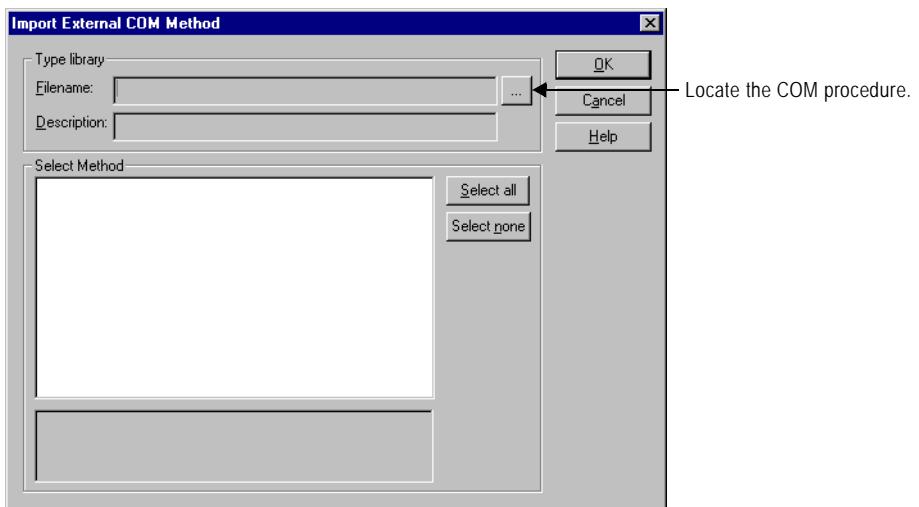
For more information about how to package COM classes for distribution to other Integration Services, see “Distributing External Procedures” on page 169.

For more information about how to use COM external procedures to call functions in a preexisting library of C or C++ functions, see “Wrapper Classes for Pre-Existing C/C++ Libraries or VB Functions” on page 173.

For more information about how to use a class factory to initialize COM objects, see “Initializing COM and Informatica Modules” on page 175.

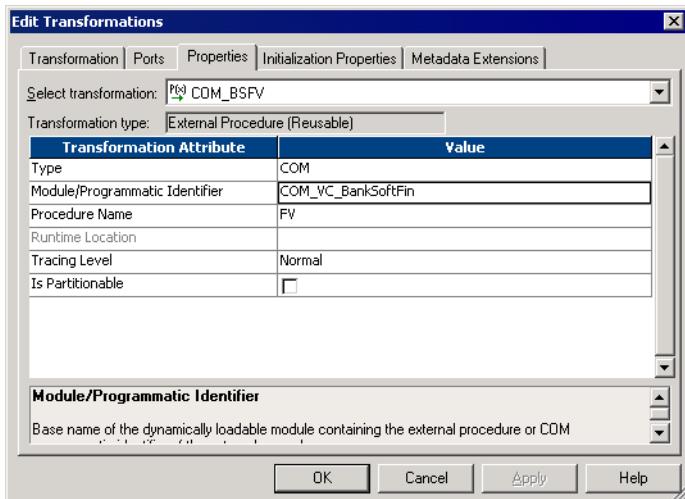
Step 6. Register a COM Procedure with the Repository

1. Open the Transformation Developer.
 2. Click Transformation > Import External Procedure.
- The Import External COM Method dialog box appears.
3. Click the Browse button.



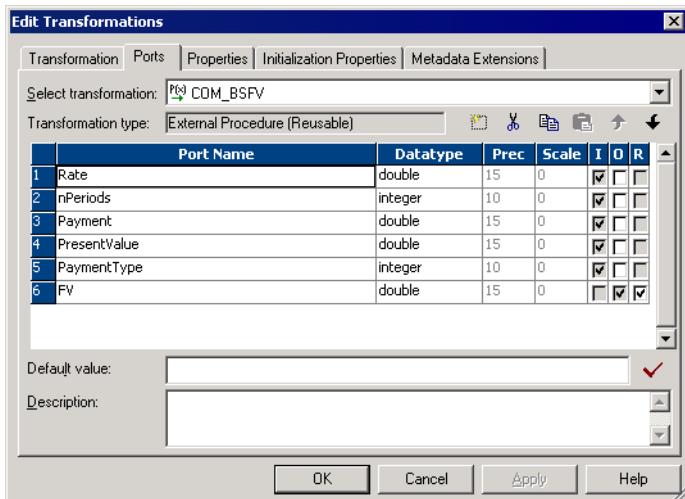
4. Select the COM DLL you created and click OK.
In the Banksoft example, select *COM_VC_Banksoft.DLL*.
5. Under Select Method tree view, expand the class node (in this example, BSoftFin).
6. Expand Methods.
7. Select the method you want (in this example, FV) and press OK.
The Designer creates an External Procedure transformation.
8. Open the External Procedure transformation, and select the Properties tab.

The transformation properties display:



Enter ASCII characters in the Module/Programmatic Identifier and Procedure Name fields.

9. Click the Ports tab.



Enter ASCII characters in the Port Name fields. For more information about mapping Visual C++ and Visual Basic datatypes to COM datatypes, see "COM Datatypes" on page 171.

10. Click OK, and then click Repository > Save.

The repository now contains the reusable transformation, so you can add instances of this transformation to mappings.

Step 7. Create a Source and a Target for a Mapping

Use the following SQL statements to create a source table and to populate this table with sample data:

```
create table FVInputs (
    Rate float,
    nPeriods int,
    Payment float,
    PresentValue float,
    PaymentType int
)
insert into FVInputs values (.005,10,-200.00,-500.00,1)
insert into FVInputs values (.01,12,-1000.00,0.00,0)
insert into FVInputs values (.11/12,35,-2000.00,0.00,1)
insert into FVInputs values (.005,12,-100.00,-1000.00,1)
```

Use the following SQL statement to create a target table:

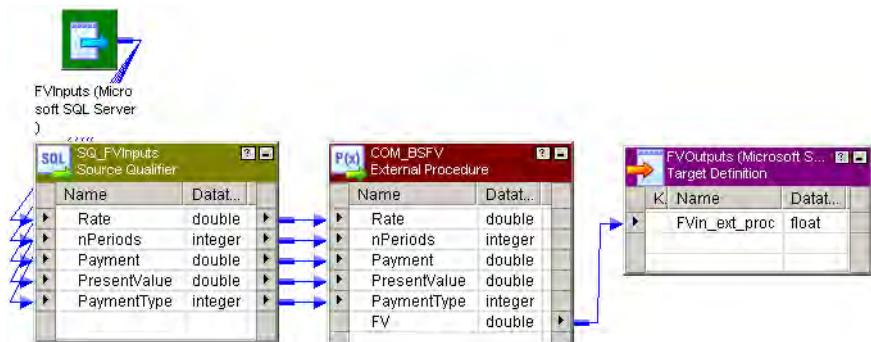
```
create table FVOutputs(
    FVin_ext_proc float,
)
```

Use the Source Analyzer and the Target Designer to import FVInputs and FVOutputs into the same folder as the one in which you created the COM_BSFV transformation.

Step 8. Create a Mapping to Test the External Procedure Transformation

Now create a mapping to test the External Procedure transformation:

1. In the Mapping Designer, create a new mapping named Test_BSFV.
2. Drag the source table FVInputs into the mapping.
3. Drag the target table FVOutputs into the mapping.
4. Drag the transformation COM_BSFV into the mapping.



5. Connect the Source Qualifier transformation ports to the External Procedure transformation ports as appropriate.

6. Connect the FV port in the External Procedure transformation to the FVIn_ext_proc target column.
7. Validate and save the mapping.

Step 9. Start the Integration Service

Start the Integration Service. Note that the service must be started on the same host as the one on which the COM component was registered.

Step 10. Run a Workflow to Test the Mapping

When the Integration Service runs the session in a workflow, it performs the following functions:

- ◆ Uses the COM runtime facilities to load the DLL and create an instance of the class.
- ◆ Uses the COM IDispatch interface to call the external procedure you defined once for every row that passes through the mapping.

Note: Multiple classes, each with multiple methods, can be defined within a single project. Each of these methods can be invoked as an external procedure.

To run a workflow to test the mapping:

1. In the Workflow Manager, create the session s_Test_BSFV from the Test_BSFV mapping.
2. Create a workflow that contains the session s_Test_BSFV.
3. Run the workflow. The Integration Service searches the registry for the entry for the COM_VC_BankSoft.BSoftFin class. This entry has information that allows the Integration Service to determine the location of the DLL that contains that class. The Integration Service loads the DLL, creates an instance of the class, and invokes the FV function for every row in the source table.

When the workflow finishes, the FVOutputs table should contain the following results:

```
FVIn_ext_proc
2581.403374
12682.503013
82846.246372
2301.401830
```

Developing COM Procedures with Visual Basic

Microsoft Visual Basic offers a different development environment for creating COM procedures. While the Basic language has different syntax and conventions, the development procedure has the same broad outlines as developing COM procedures in Visual C++.

Step 1. Create a Visual Basic Project with a Single Class

1. Launch Visual Basic and click File > New Project.
2. In the dialog box that appears, select ActiveX DLL as the project type and click OK.
Visual Basic creates a new project named *Project1*.
If the Project window does not display, type Ctrl+R, or click View > Project Explorer.
If the Properties window does not display, press F4, or click View > Properties.
3. In the Project Explorer window for the new project, right-click the project and choose Project1 Properties from the menu that appears.
4. Enter the name of the new project.
In the Project window, select *Project1* and change the name in the Properties window to *COM_VB_BankSoft*.

Step 2. Change the Names of the Project and Class

1. Inside the Project Explorer, select the “Project – Project1” item, which should be the root item in the tree control. The project properties display in the Properties Window.
2. Select the Alphabetic tab in the Properties Window and change the Name property to COM_VB_BankSoft. This renames the root item in the Project Explorer to COM_VB_BankSoft (COM_VB_BankSoft).
3. Expand the COM_VB_BankSoft (COM_VB_BankSoft) item in the Project Explorer.
4. Expand the Class Modules item.
5. Select the Class1 (Class1) item. The properties of the class display in the Properties Window.
6. Select the Alphabetic tab in the Properties Window and change the Name property to BSoftFin.

By changing the name of the project and class, you specify that the programmatic identifier for the class you create is “COM_VB_BankSoft.BSoftFin.” Use this ProgID to refer to this class inside the Designer.

Step 3. Add a Method to the Class

Place the pointer inside the Code window and enter the following text:

```
Public Function FV( _
    Rate As Double, _
    nPeriods As Long, _
    Payment As Double, _
    PresentValue As Double, _
    PaymentType As Long _
) As Double

    Dim v As Double
    v = (1 + Rate) ^ nPeriods
```

```

FV = -( 
    (PresentValue * v) + -
    (Payment * (1 + (Rate * PaymentType))) * ((v - 1) / Rate) -
)
End Function

```

This Visual Basic FV function, of course, performs the same operation as the C++ FV function in “Developing COM Procedures with Visual Basic” on page 156.

Step 4. Build the Project

To build the project:

1. From the File menu, select the Make COM_VB_BankSoft.DLL. A dialog box prompts you for the file location.
2. Enter the file location and click OK.

Visual Basic compiles the source code and creates the COM_VB_BankSoft.DLL in the location you specified. It also registers the class COM_VB_BankSoft.BSoftFin in the local registry.

Once the component is registered, it is accessible to the Integration Service running on that host.

For more information about how to package Visual Basic COM classes for distribution to other machines hosting the Integration Service, see “Distributing External Procedures” on page 169.

For more information about how to use Visual Basic external procedures to call preexisting Visual Basic functions, see “Wrapper Classes for Pre-Existing C/C++ Libraries or VB Functions” on page 173.

To create the procedure, follow steps 6 to 9 of “Using Visual C++ to Develop COM Procedures” on page 149.

Developing Informatica External Procedures

You can create external procedures that run on 32-bit or 64-bit Integration Service machines. Complete the following steps to create an Informatica-style external procedure:

1. In the Transformation Developer, create an External Procedure transformation.

The External Procedure transformation defines the signature of the procedure. The names of the ports, datatypes and port type (input or output) must match the signature of the external procedure.

2. Generate the template code for the external procedure.

When you execute this command, the Designer uses the information from the External Procedure transformation to create several C++ source code files and a makefile. One of these source code files contains a “stub” for the function whose signature you defined in the transformation.

3. Modify the code to add the procedure logic. Fill out the stub with an implementation and use a C++ compiler to compile and link the source code files into a dynamic link library or shared library.

When the Integration Service encounters an External Procedure transformation bound to an Informatica procedure, it loads the DLL or shared library and calls the external procedure you defined.

4. Build the library and copy it to the Integration Service machine.
5. Create a mapping with the External Procedure transformation.
6. Run the session in a workflow.

We use the BankSoft example to illustrate how to implement this feature.

Step 1. Create the External Procedure Transformation

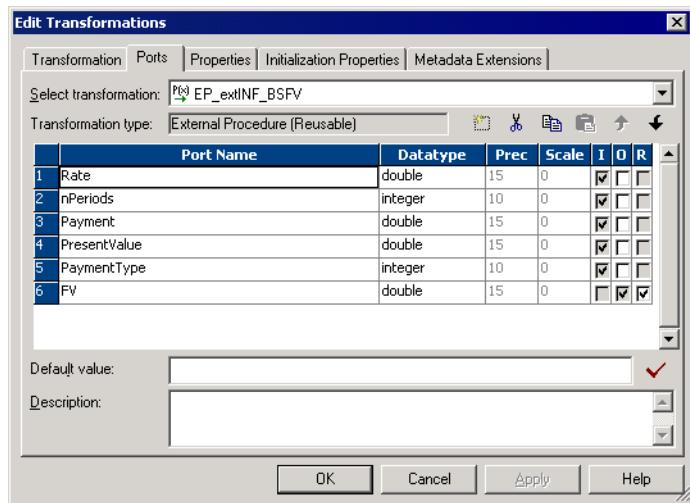
1. Open the Transformation Developer and create an External Procedure transformation.
2. Open the transformation and enter a name for it.

In the BankSoft example, enter `EP_extINF_BSFV`.

3. Create a port for each argument passed to the procedure you plan to define.

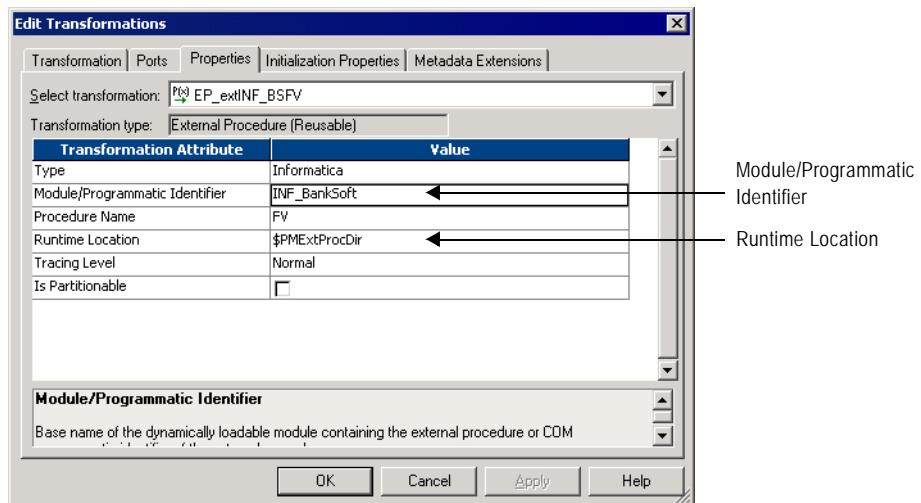
Be sure that you use the correct datatypes.

To use the FV procedure as an example, you create the following ports. The last port, FV, captures the return value from the procedure:



- Select the Properties tab and configure the procedure as an Informatica procedure.

In the BankSoft example, enter the following:



Module/Programmatic Identifier
Runtime Location

Note on Module/Programmatic Identifier:

- The module name is the base name of the dynamic link library (on Windows) or the shared object (on UNIX) that contains the external procedures. The following table

describes how the module name determines the name of the DLL or shared object on the various platforms:

Operating System	Module Identifier	Library File Name
Windows	INF_BankSoft	INF_BankSoft.DLL
AIX	INF_BankSoft	libINF_BankSoftsh.a
HPUX	INF_BankSoft	libINF_BankSoft.sl
Linux	INF_BankSoft	libINF_BankSoft.so
Solaris	INF_BankSoft	libINF_BankSoft.so.1

Notes on Runtime Location:

- ◆ If you set the Runtime Location to \$PMExtProcDir, then the Integration Service looks in the directory specified by the process variable \$PMExtProcDir to locate the library.
- ◆ If you leave the Runtime Location property blank, the Integration Service uses the environment variable defined on the server platform to locate the dynamic link library or shared object. The following table describes the environment variables used to locate the DLL or shared object on the various platforms:

Operating System	Environment Variable
Windows	PATH
AIX	LIBPATH
HPUX	SHLIB_PATH
Linux	LD_LIBRARY_PATH
Solaris	LD_LIBRARY_PATH

- ◆ You can hard code a path as the Runtime Location. This is not recommended since the path is specific to a single machine only.

Note: You must copy all DLLs or shared libraries to the Runtime Location or to the environment variable defined on the Integration Service machine. The Integration Service fails to load the external procedure when it cannot locate the DLL, shared library, or a referenced file.

5. Click OK.

6. Click Repository > Save.

After you create the External Procedure transformation that calls the procedure, the next step is to generate the C++ files.

Step 2. Generate the C++ Files

After you create an External Procedure transformation, you generate the code. The Designer generates file names in lower case since files created on UNIX-mapped drives are always in lower case. The following rules apply to the generated files:

- ♦ **File names.** A prefix ‘tx’ is used for TX module files.
- ♦ **Module class names.** The generated code has class declarations for the module that contains the TX procedures. A prefix *Tx* is used for TX module classes. For example, if an External Procedure transformation has a module name Mymod, then the class name is TxMymod.

To generate the code for an external procedure:

1. Select the transformation and click Transformation > Generate Code.
2. Select the check box next to the name of the procedure you just created.
In the BankSoft example, select *INF_BankSoft.FV*.
3. Specify the directory where you want to generate the files, and click Generate.

The Designer creates a subdirectory, INF_BankSoft, in the directory you specified.

Each External Procedure transformation created in the Designer must specify a module and a procedure name. The Designer generates code in a single directory for all transformations sharing a common module name. Building the code in one directory creates a single shared library.

The Designer generates the following files:

- ♦ **tx<moduleName>.h.** Defines the external procedure module class. This class is derived from a base class TINFExternalModule60. No data members are defined for this class in the generated code. However, you can add new data members and methods here.
- ♦ **tx<moduleName>.cpp.** Implements the external procedure module class. You can expand the InitDerived() method to include initialization of any new data members you add. The Integration Service calls the derived class InitDerived() method only when it successfully completes the base class Init() method.

This file defines the signatures of all External Procedure transformations in the module. Any modification of these signatures leads to inconsistency with the External Procedure transformations defined in the Designer. Therefore, you should not change the signatures.

This file also includes a C function CreateExternalModuleObject, which creates an object of the external procedure module class using the constructor defined in this file. The Integration Service calls CreateExternalModuleObject instead of directly calling the constructor.

- ♦ **<procedureName>.cpp.** The Designer generates one of these files for each external procedure in this module. This file contains the code that implements the procedure logic, such as data cleansing and filtering. For data cleansing, create code to read in

values from the input ports and generate values for output ports. For filtering, create code to suppress generation of output rows by returning INF_NO_OUTPUT_ROW.

- ◆ **stdafx.h.** Stub file used for building on UNIX systems. The various *.cpp files include this file. On Windows systems, the Visual Studio generates an stdafx.h file, which should be used instead of the Designer generated file.
- ◆ **version.cpp.** This is a small file that carries the version number of this implementation. In earlier releases, external procedure implementation was handled differently. This file allows the Integration Service to determine the version of the external procedure module.
- ◆ **makefile.aix, makefile.aix64, makefile.hp, makefile.hp64, makefile.hpparisc64, makefile.linux, makefile.sol.** Make files for UNIX platforms. Use makefile.aix, makefile.hp, makefile.linux, and makefile.sol for 32-bit platforms. Use makefile.aix64 for 64-bit AIX platforms and makefile.hp64 for 64-bit HP-UX (Itanium) platforms.

Example 1

In the BankSoft example, the Designer generates the following files:

- ◆ **txinf_banksoft.h.** Contains declarations for module class TxINF_BankSoft and external procedure FV.
- ◆ **txinf_banksoft.cpp.** Contains code for module class TxINF_BankSoft.
- ◆ **fv.cpp.** Contains code for procedure FV.
- ◆ **version.cpp.** Returns TX version.
- ◆ **stdafx.h.** Required for compilation on UNIX. On Windows, stdafx.h is generated by Visual Studio.
- ◆ **readme.txt.** Contains general help information.

Example 2

If you create two External Procedure transformations with procedure names ‘Myproc1’ and ‘Myproc2,’ both with the module name Mymod, the Designer generates the following files:

- ◆ **txmymod.h.** Contains declarations for module class TxMymod and external procedures Myproc1 and Myproc2.
- ◆ **txmymod.cpp.** Contains code for module class TxMymod.
- ◆ **myproc1.cpp.** Contains code for procedure Myproc1.
- ◆ **myproc2.cpp.** Contains code for procedure Myproc2.
- ◆ **version.cpp.**
- ◆ **stdafx.h.**
- ◆ **readme.txt.**

Step 3. Fill Out the Method Stub with Implementation

The final step is coding the procedure.

1. Open the *<Procedure_Name>.cpp* stub file generated for the procedure.

In the BankSoft example, you open fv.cpp to code the TxINF_BankSoft::FV procedure.

2. Enter the C++ code for the procedure.

The following code implements the FV procedure:

```
INF_RESULT TxINF_BankSoft::FV()
{
    // Input port values are mapped to the m_pInParamVector array in
    // the InitParams method. Use m_pInParamVector[i].IsValid() to check
    // if they are valid. Use m_pInParamVector[i].GetLong or GetDouble,
    // etc. to get their value. Generate output data into m_pOutParamVector.

    // TODO: Fill in implementation of the FV method here.

    ostrstream ss;
    char* s;

    INF_Boolean bVal;
    double v;

    TINFParam* Rate = &m_pInParamVector[0];
    TINFParam* nPeriods = &m_pInParamVector[1];
    TINFParam* Payment = &m_pInParamVector[2];
    TINFParam* PresentValue = &m_pInParamVector[3];
    TINFParam* PaymentType = &m_pInParamVector[4];
    TINFParam* FV = &m_pOutParamVector[0];

    bVal =
        INF_Boolean(
            Rate->IsValid() &&
            nPeriods->IsValid() &&
            Payment->IsValid() &&
            PresentValue->IsValid() &&
            PaymentType->IsValid()
        );
    if (bVal == INF_FALSE)
    {
        FV->SetIndicator(INF_SQL_DATA_NULL);
        return INF_SUCCESS;
    }
}
```

```

v = pow((1 + Rate->GetDouble()), (double)nPeriods->GetLong());
FV->SetDouble(
    -
    (PresentValue->GetDouble() * v) +
    (Payment->GetDouble() *
     (1 + (Rate->GetDouble() * PaymentType->GetLong()))) *
     ((v - 1) / Rate->GetDouble())
)
);
ss << "The calculated future value is: " << FV->GetDouble() <<ends;
s = ss.str();
(*m_pfnMessageCallback)(E_MSG_TYPE_LOG, 0, s);
(*m_pfnMessageCallback)(E_MSG_TYPE_ERR, 0, s);
delete [] s;
return INF_SUCCESS;
}

```

The Designer generates the function profile, including the arguments and return value. You need to enter the actual code within the function, as indicated in the comments. Since you referenced the POW function and defined an `ostrstream` variable, you must also include the preprocessor statements:

On Windows:

```
#include <math.h>;
#include <strstrea.h>;
```

On UNIX, the include statements would be the following:

```
#include <math.h>;
#include <strstream.h>;
```

- 3.** Save the modified file.

Step 4. Building the Module

On Windows, use Visual C++ to compile the DLL.

To build a DLL on Windows:

- 1.** Start Visual C++.
- 2.** Click File > New.
- 3.** In the New dialog box, click the Projects tab and select the MFC AppWizard (DLL) option.

4. Enter its location.

In the BankSoft example, you enter `c:\pmclient\tx\INF_BankSoft`, assuming you generated files in `c:\pmclient\tx`.

5. Enter the name of the project.

It must be the same as the module name entered for the External Procedure transformation. In the BankSoft example, it is INF_BankSoft.

6. Click OK.

Visual C++ now steps you through a wizard that defines all the components of the project.

7. In the wizard, click MFC Extension DLL (using shared MFC DLL).

8. Click Finish.

The wizard generates several files.

9. Click Project > Add To Project > Files.

10. Navigate up a directory level. This directory contains the external procedure files you created. Select all .cpp files.

In the BankSoft example, add the following files:

- ◆ fv.cpp
- ◆ txinf_banksoft.cpp
- ◆ version.cpp

11. Click Project > Settings.

12. Click the C/C++ tab, and select Preprocessor from the Category field.

13. In the Additional Include Directories field, enter ..; *<pmserver install dir>\extproc\include*.

14. Click the Link tab, and select General from the Category field.

15. Enter *<pmserver install dir>\bin\pmtx.lib* in the Object/Library Modules field.

16. Click OK.

17. Click Build > Build INF_BankSoft.dll or press F7 to build the project.

The compiler now creates the DLL and places it in the debug or release directory under the project directory. For information about running a workflow with the debug version, see “Running a Session with the Debug Version of the Module on Windows” on page 168.

To build shared libraries on UNIX:

1. If you cannot access the PowerCenter Client tools directly, copy all the files you need for the shared library to the UNIX machine where you plan to perform the build.

For example, in the BankSoft procedure, use ftp or another mechanism to copy everything from the INF_BankSoft directory to the UNIX machine.

2. Set the environment variable INFA_HOME to the PowerCenter installation directory.

Warning: If you specify an incorrect directory path for the INFA_HOME environment variable, the Integration Service cannot start.

3. Enter the command to make the project.

The command depends on the version of UNIX, as summarized below:

UNIX Version	Command
AIX (32-bit)	make -f makefile.aix
AIX (64-bit)	make -f makefile.aix64
HP-UX (32-bit)	make -f makefile.hp
HP-UX (64-bit)	make -f makefile.hp64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

Step 5. Create a Mapping

In the Mapping Designer, create a mapping that uses this External Procedure transformation.

Step 6. Run the Session in a Workflow

When you run the session in a workflow, the Integration Service looks in the directory you specify as the Runtime Location to find the library (DLL) you built in Step 4. The default value of the Runtime Location property in the session properties is \$PMExtProcDir.

To run a session in a workflow:

1. In the Workflow Manager, create a workflow.
2. Create a session for this mapping in the workflow.

Tip: Alternatively, you can create a re-usable session in the Task Developer and use it in the workflow.

3. Copy the library (DLL) to the Runtime Location directory.
4. Run the workflow containing the session.

Running a Session with the Debug Version of the Module on Windows

Informatica ships PowerCenter on Windows with the release build (pmtx.dll) and the debug build (pmtxdbg.dll) of the External Procedure transformation library. These libraries are installed in the server bin directory.

If you build a release version of the module in Step 4, run the session in a workflow to use the release build (pmtx.dll) of the External Procedure transformation library. You do not need to complete the following task.

If you build a debug version of the module in Step 4, follow the procedure below to use the debug build (pmtxdbg.dll) of the External Procedure transformation library.

To run a session using a debug version of the module:

- 1.** In the Workflow Manager, create a workflow.
- 2.** Create a session for this mapping in the workflow.
Or, you can create a re-usable session in the Task Developer and use it in the workflow.
- 3.** Copy the library (DLL) to the Runtime Location directory.
- 4.** To use the debug build of the External Procedure transformation library:
 - ◆ Preserve pmtx.dll by renaming it or moving it from the server bin directory.
 - ◆ Rename pmtxdbg.dll to pmtx.dll.
- 5.** Run the workflow containing the session.
- 6.** To revert the release build of the External Procedure transformation library back to the default library:
 - ◆ Rename pmtx.dll back to pmtxdbg.dll.
 - ◆ Return/rename the original pmtx.dll file to the server bin directory.

Note: If you run a workflow containing this session with the debug version of the module on Windows, you must return the original pmtx.dll file to its original name and location before you can run a non-debug session.

Distributing External Procedures

Suppose you develop a set of external procedures and you want to make them available on multiple servers, each of which is running the Integration Service. The methods for doing this depend on the type of the external procedure and the operating system on which you built it.

You can also use these procedures to distribute external procedures to external customers.

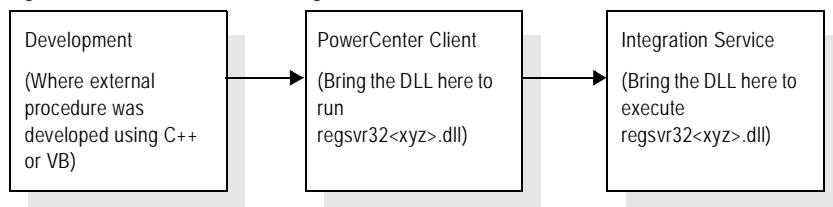
Distributing COM Procedures

Visual Basic and Visual C++ register COM classes in the local registry when you build the project. Once registered, these classes are accessible to the Integration Service running on the machine where you compiled the DLL. For example, if you build a project on HOST1, all the classes in the project will be registered in the HOST1 registry and will be accessible to the Integration Service running on HOST1. Suppose, however, that you also want the classes to be accessible to the Integration Service running on HOST2. For this to happen, the classes must be registered in the HOST2 registry.

Visual Basic provides a utility for creating a setup program that can install COM classes on a Windows machine and register these classes in the registry on that machine. While no utility is available in Visual C++, you can easily register the class yourself.

Figure 6-1 shows the process for distributing external procedures:

Figure 6-1. Process for Distributing External Procedures



To distribute a COM Visual Basic procedure:

1. After you build the DLL, exit Visual Basic and launch the Visual Basic Application Setup wizard.
2. Skip the first panel of the wizard.
3. On the second panel, specify the location of the project and select the *Create a Setup Program* option.
4. In the third panel, select the method of distribution you plan to use.
5. In the next panel, specify the directory to which you want to write the setup files.

For simple ActiveX components, you can continue to the final panel of the wizard. Otherwise, you may need to add more information, depending on the type of file and the method of distribution.

6. Click Finish in the final panel.

Visual Basic then creates the setup program for the DLL. Run this setup program on any Windows machine where the Integration Service is running.

To distribute a COM Visual C++/Visual Basic procedure manually:

1. Copy the DLL to the directory on the new Windows machine anywhere you want it saved.
2. Log in to this Windows machine and open a DOS prompt.
3. Navigate to the directory containing the DLL and execute the following command:

```
REGSVR32 project_name.DLL
```

project_name is the name of the DLL you created. In the BankSoft example, the project name is *COM_VC_BankSoft.DLL*. or *COM_VB_BankSoft.DLL*.

This command line program then registers the DLL and any COM classes contained in it.

Distributing Informatica Modules

You can distribute external procedures between repositories.

To distribute external procedures between repositories:

1. Move the DLL or shared object that contains the external procedure to a directory on a machine that the Integration Service can access.
2. Copy the External Procedure transformation from the original repository to the target repository using the Designer client tool.

-or-

Export the External Procedure transformation to an XML file and import it in the target repository.

For more information, see “Exporting and Importing Objects” in the *Repository Guide*.

Development Notes

This section includes some additional guidelines and information about developing COM and Informatica external procedures.

COM Datatypes

When using either Visual C++ or Visual Basic to develop COM procedures, you need to use COM datatypes that correspond to the internal datatypes that the Integration Service uses when reading and transforming data. These datatype matches are important when the Integration Service attempts to map datatypes between ports in an External Procedure transformation and arguments (or return values) from the procedure the transformation calls.

Table 6-2 compares Visual C++ and transformation datatypes:

Table 6-2. Visual C++ and Transformation Datatypes

Visual C++ COM Datatype	Transformation Datatype
VT_I4	Integer
VT_UI4	Integer
VT_R8	Double
VT_BSTR	String
VT_DECIMAL	Decimal
VT_DATE	Date/Time

Table 6-3 compares Visual Basic and transformation datatypes:

Table 6-3. Visual Basic and Transformation Datatypes

Visual Basic COM Datatype	Transformation Datatype
Long	Integer
Double	Double
String	String
Decimal	Decimal
Date	Date/Time

If you do not correctly match datatypes, the Integration Service may attempt a conversion. For example, if you assign the Integer datatype to a port, but the datatype for the corresponding argument is BSTR, the Integration Service attempts to convert the Integer value to a BSTR.

Row-Level Procedures

All External Procedure transformations call procedures using values from a single row passed through the transformation. You cannot use values from multiple rows in a single procedure call. For example, you could not code the equivalent of the aggregate functions SUM or AVG into a procedure call. In this sense, all external procedures must be *stateless*.

Return Values from Procedures

When you call a procedure, the Integration Service captures an additional return value beyond whatever return value you code into the procedure. This additional value indicates whether the Integration Service successfully called the procedure.

For COM procedures, this return value uses the type HRESULT.

Informatica procedures use the type INF_RESULT. If the value returned is S_OK/INF_SUCCESS, the Integration Service successfully called the procedure. You must return the appropriate value to indicate the success or failure of the external procedure. Informatica procedures return four values:

- ◆ **INF_SUCCESS.** The external procedure processed the row successfully. The Integration Service passes the row to the next transformation in the mapping.
- ◆ **INF_NO_OUTPUT_ROW.** The Integration Service does not write the current row due to external procedure logic. This is not an error. When you use INF_NO_OUTPUT_ROW to filter rows, the External Procedure transformation behaves similarly to the Filter transformation.

Note: When you use INF_NO_OUTPUT_ROW in the external procedure, make sure you connect the External Procedure transformation to another transformation that receives rows from the External Procedure transformation only.

- ◆ **INF_ROW_ERROR.** Equivalent to a transformation error. The Integration Service discards the current row, but may process the next row unless you configure the session to stop on *n* errors.
- ◆ **INF_FATAL_ERROR.** Equivalent to an ABORT() function call. The Integration Service aborts the session and does not process any more rows. For more information, see “Functions” in the *Transformation Language Reference*.

Exceptions in Procedure Calls

The Integration Service captures most exceptions that occur when it calls a COM or Informatica procedure through an External Procedure transformation. For example, if the procedure call creates a divide by zero error, the Integration Service catches the exception.

In a few cases, the Integration Service cannot capture errors generated by procedure calls. Since the Integration Service supports only in-process COM servers, and since all Informatica procedures are stored in shared libraries and DLLs, the code running external procedures exists in the same address space in memory as the Integration Service. Therefore, it is possible for the external procedure code to overwrite the Integration Service memory, causing the

Integration Service to stop. If COM or Informatica procedures cause such stops, review the source code for memory access problems.

Memory Management for Procedures

Since all the datatypes used in Informatica procedures are fixed length, there are no memory management issues for Informatica external procedures. For COM procedures, you need to allocate memory only if an [out] parameter from a procedure uses the BSTR datatype. In this case, you need to allocate memory on every call to this procedure. During a session, the Integration Service releases the memory after calling the function.

Wrapper Classes for Pre-Existing C/C++ Libraries or VB Functions

Suppose that BankSoft has a library of C or C++ functions and wants to plug these functions in to the Integration Service. In particular, the library contains BankSoft's own implementation of the FV function, called PreExistingFV. The general method for doing this is the same for both COM and Informatica external procedures. A similar solution is available in Visual Basic. You need only make calls to preexisting Visual Basic functions or to methods on objects that are accessible to Visual Basic.

Generating Error and Tracing Messages

The implementation of the Informatica external procedure TxINF_BankSoft::FV in “Step 4. Building the Module” on page 165 contains the following lines of code.

```
ostrstream ss;
char* s;
...
ss << "The calculated future value is: " << FV->GetDouble() << endl;
s = ss.str();
(*m_pfnMessageCallback)(E_MSG_TYPE_LOG, 0, s);
(*m_pfnMessageCallback)(E_MSG_TYPE_ERR, 0, s);
delete [] s;
```

When the Integration Service creates an object of type Tx<MODNAME>, it passes to its constructor a pointer to a callback function that can be used to write error or debugging messages to the session log. (The code for the Tx<MODNAME> constructor is in the file Tx<MODNAME>.cpp.) This pointer is stored in the Tx<MODNAME> member variable m_pfnMessageCallback. The type of this pointer is defined in a typedef in the file \$PMExtProcDir/include/infemmsg.h:

```
typedef void (*PFN_MESSAGE_CALLBACK) (
    enum E_MSG_TYPE eMsgType,
    unsigned long Code,
    char* Message
);
```

Also defined in that file is the enumeration E_MSG_TYPE:

```
enum E_MSG_TYPE {
    E_MSG_TYPE_LOG = 0,
    E_MSG_TYPE_WARNING,
```

```
    E_MSG_TYPE_ERR  
};
```

If you specify the eMsgType of the callback function as E_MSG_TYPE_LOG, the callback function will write a *log* message to the session log. If you specify E_MSG_TYPE_ERR, the callback function writes an *error* message to the session log. If you specify E_MSG_TYPE_WARNING, the callback function writes an *warning* message to the session log. Use these messages to provide a simple debugging capability in Informatica external procedures.

To debug COM external procedures, you may use the output facilities available from inside a Visual Basic or C++ class. For example, in Visual Basic use a MsgBox to print out the result of a calculation for each row. Of course, you want to do this only on small samples of data while debugging and make sure to remove the MsgBox before making a production run.

Note: Before attempting to use any output facilities from inside a Visual Basic or C++ class, you must add the following value to the registry:

1. Add the following entry to the Windows registry:

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo\RunInDebugMode=Yes
```

This option starts the Integration Service as a regular application, not a service. You can debug the Integration Service without changing the debug privileges for the Integration Service service while it is running.

2. Start the Integration Service from the command line, using the command PMSERVER.EXE.

The Integration Service is now running in debug mode.

When you are finished debugging, make sure you remove this entry from the registry or set RunInDebugMode to No. Otherwise, when you attempt to start PowerCenter as a service, it will not start.

1. Stop the Integration Service and change the registry entry you added earlier to the following setting:

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo\RunInDebugMode=No
```

2. Restart the Integration Service as a Windows service.

The TINFParam Class and Indicators

The <PROCNAME> method accesses input and output parameters using two parameter arrays, and that each array element is of the TINFParam datatype. The TINFParam datatype is a C++ class that serves as a “variant” data structure that can hold any of the Informatica internal datatypes. The actual data in a parameter of type TINFParam* is accessed through member functions of the form Get<Type> and Set<Type>, where <Type> is one of the Informatica internal datatypes. TINFParam also has methods for getting and setting the indicator for each parameter.

You are responsible for checking these indicators on entry to the external procedure and for setting them on exit. On entry, the indicators of all output parameters are explicitly set to INF_SQL_DATA_NULL, so if you do not reset these indicators before returning from the external procedure, you will just get NULLs for all the output parameters. The TINFParam class also supports functions for obtaining the metadata for a particular parameter. For a complete description of all the member functions of the TINFParam class, see the infemdef.h include file in the tx/include directory.

Note that one of the main advantages of Informatica external procedures over COM external procedures is that Informatica external procedures directly support indicator manipulation. That is, you can check an input parameter to see if it is NULL, and you can set an output parameter to NULL. COM provides no indicator support. Consequently, if a row entering a COM-style external procedure has any NULLs in it, the row cannot be processed. Use the default value facility in the Designer to overcome this shortcoming. However, it is not possible to pass NULLs out of a COM function.

Unconnected External Procedure Transformations

When you add an instance of an External Procedure transformation to a mapping, you can choose to connect it as part of the pipeline or leave it unconnected. Connected External Procedure transformations call the COM or Informatica procedure every time a row passes through the transformation.

To get return values from an unconnected External Procedure transformation, call it in an expression using the following syntax:

```
:EXT.transformation_name(arguments)
```

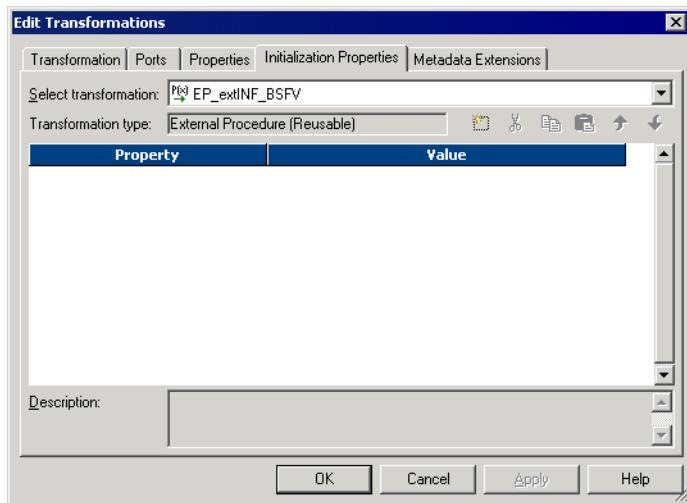
When a row passes through the transformation containing the expression, the Integration Service calls the procedure associated with the External Procedure transformation. The expression captures the return value of the procedure through the External Procedure transformation return port, which should have the Result (R) option checked. For more information about expressions, see “Working with Expressions” on page 10.

Initializing COM and Informatica Modules

Some external procedures must be configured at initialization time. This initialization takes one of two forms, depending on the type of the external procedure:

1. **Initialization of Informatica-style external procedures.** The Tx<MODNAME> class, which contains the external procedure, also contains the initialization function, Tx<MODNAME>::InitDerived. The signature of this initialization function is well-known to the Integration Service and consists of three parameters:
 - ◆ **nInitProps.** This parameter tells the initialization function how many initialization properties are being passed to it.
 - ◆ **Properties.** This parameter is an array of nInitProp strings representing the names of the initialization properties.

- ◆ **Values.** This parameter is an array of nInitProp strings representing the values of the initialization properties.



The Integration Service first calls the Init() function in the base class. When the Init() function successfully completes, the base class calls the Tx<MODNAME>::InitDerived() function.

The Integration Service creates the Tx<MODNAME> object and then calls the initialization function. It is the responsibility of the external procedure developer to supply that part of the Tx<MODNAME>::InitDerived() function that interprets the initialization properties and uses them to initialize the external procedure. Once the object is created and initialized, the Integration Service can call the external procedure on the object for each row.

2. **Initialization of COM-style external procedures.** The object that contains the external procedure (or EP object) does not contain an initialization function. Instead, another object (the CF object) serves as a class factory for the EP object. The CF object has a method that can create an EP object.

The signature of the CF object method is determined from its type library. The Integration Service creates the CF object, and then calls the method on it to create the EP object, passing this method whatever parameters are required. This requires that the signature of the method consist of a set of input parameters, whose types can be determined from the type library, followed by a single output parameter that is an IUnknown** or an IDispatch** or a VARIANT* pointing to an IUnknown* or IDispatch*.

The input parameters hold the values required to initialize the EP object and the output parameter receives the initialized object. The output parameter can have either the [out] or the [out, retval] attributes. That is, the initialized object can be returned either as an

output parameter or as the return value of the method. The datatypes supported for the input parameters are:

- ◆ COM VC type
- ◆ VT_UI1
- ◆ VT_BOOL
- ◆ VT_I2
- ◆ VT_UI2
- ◆ VT_I4
- ◆ VT_UI4
- ◆ VT_R4
- ◆ VT_R8
- ◆ VT_BSTR
- ◆ VT_CY
- ◆ VT_DATE

Setting Initialization Properties in the Designer

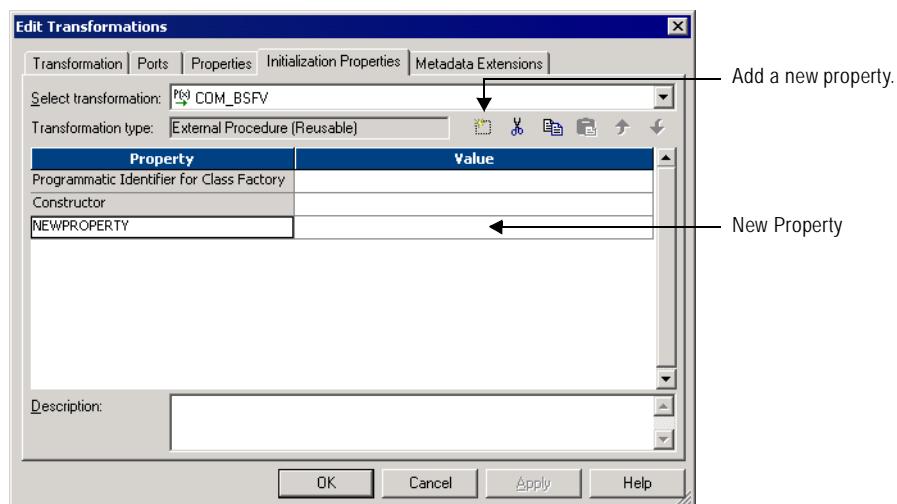
Enter external procedure initialization properties on the Initialization Properties tab of the Edit Transformations dialog box. The tab displays different fields, depending on whether the external procedure is COM-style or Informatica-style.

COM-style External Procedure transformations contain the following fields on the Initialization Properties tab:

- ◆ **Programmatic Identifier for Class Factory.** Enter the programmatic identifier of the class factory.
- ◆ **Constructor.** Specify the method of the class factory that creates the EP object.

Figure 6-2 shows the Initialization Properties tab of a COM-style External Procedure transformation:

Figure 6-2. External Procedure Transformation Initialization Properties



You can enter an unlimited number of initialization properties to pass to the Constructor method for both COM-style and Informatica-style External Procedure transformations.

To add a new initialization property, click the Add button. Enter the name of the parameter in the Property column and enter the value of the parameter in the Value column. For example, you can enter the following parameters:

Parameter	Value
Param1	abc
Param2	100
Param3	3.17

Note: You must create a one-to-one relation between the initialization properties you define in the Designer and the input parameters of the class factory constructor method. For example, if the constructor has n parameters with the last parameter being the output parameter that receives the initialized object, you must define $n - 1$ initialization properties in the Designer, one for each input parameter in the constructor method.

You can also use process variables in initialization properties. For information about process variables support in Initialization properties, see “Service Process Variables in Initialization Properties” on page 180.

Other Files Distributed and Used in TX

Following are the header files located under the path `$PMExtProcDir/include` that are needed for compiling external procedures:

- ◆ `infconfig.h`
- ◆ `infem60.h`
- ◆ `infemdef.h`
- ◆ `infemmsg.h`
- ◆ `infparam.h`
- ◆ `infsigtr.h`

Following are the library files located under the path `<PMInstallDir>` that are needed for linking external procedures and running the session:

- ◆ `libpmtx.a` (AIX)
- ◆ `libpmtx.sl` (HP-UX)
- ◆ `libpmtx.so` (Linux)
- ◆ `libpmtx.so` (Solaris)
- ◆ `pmtx.dll` and `pmtx.lib` (Windows)

Service Process Variables in Initialization Properties

PowerCenter supports built-in process variables in the External Procedure transformation initialization properties list. If the property values contain built-in process variables, the Integration Service expands them before passing them to the external procedure library. This can be very useful for writing portable External Procedure transformations.

Figure 6-3 shows an External Procedure transformation with five user-defined properties:

Figure 6-3. External Procedure Transformation Initialization Properties Tab

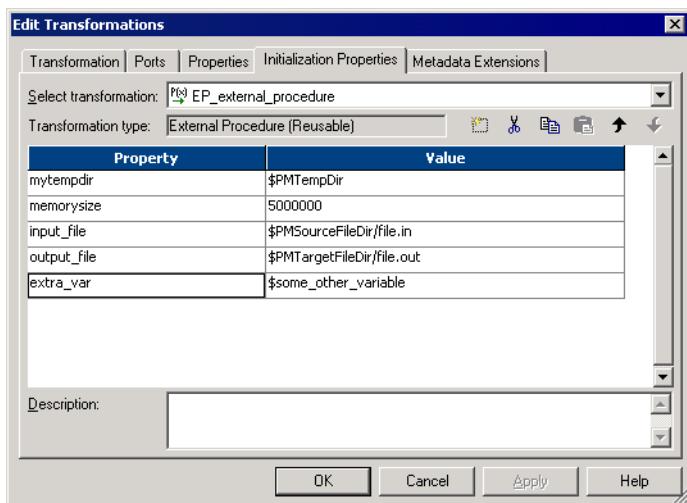


Table 6-4 contains the initialization properties and values for the External Procedure transformation in Figure 6-3:

Table 6-4. External Procedure Initialization Properties

Property	Value	Expanded Value Passed to the External Procedure Library
mytempdir	\$PMTempDir	/tmp
memorysize	5000000	5000000
input_file	\$PMSourceFileDir/file.in	/data/input/file.in
output_file	\$PMTargetFileDir/file.out	/data/output/file.out
extra_var	\$some_other_variable	\$some_other_variable

When you run the workflow, the Integration Service expands the property list and passes it to the external procedure initialization function. Assuming that the values of the built-in process variables \$PMTempDir is /tmp, \$PMSourceFileDir is /data/input, and \$PMTargetFileDir is /data/output, the last column in Table 6-4 contains the property and expanded value information. Note that the Integration Service does not expand the last property “\$some_other_variable” because it is not a built-in process variable.

External Procedure Interfaces

The Integration Service uses the following major functions with External Procedures:

- ◆ Dispatch
- ◆ External procedure
- ◆ Property access
- ◆ Parameter access
- ◆ Code page access
- ◆ Transformation name access
- ◆ Procedure access
- ◆ Partition related
- ◆ Tracing level

Dispatch Function

The Integration Service calls the dispatch function to pass each input row to the external procedure module. The dispatch function, in turn, calls the external procedure function you specify.

External procedures access the ports in the transformation directly using the member variable `m_pInParamVector` for input ports and `m_pOutParamVector` for output ports.

Signature

The dispatch function has a fixed signature which includes one index parameter.

```
virtual INF_RESULT Dispatch(unsigned long ProcedureIndex) = 0
```

External Procedure Function

The external procedure function is the main entry point into the external procedure module, and is an attribute of the External Procedure transformation. The dispatch function calls the external procedure function for every input row. For External Procedure transformations, use the external procedure function for input and output from the external procedure module. The function can access the IN and IN-OUT port values for every input row, and can set the OUT and IN-OUT port values. The external procedure function contains all the input and output processing logic.

Signature

The external procedure function has no parameters. The input parameter array is already passed through the `InitParams()` method and stored in the member variable `m_pInParamVector`. Each entry in the array matches the corresponding IN and IN-OUT ports of the External Procedure transformation, in the same order. The Integration Service fills this vector before calling the dispatch function.

Use the member variable `m_pOutParamVector` to pass the output row before returning the `Dispatch()` function.

For the MyExternal Procedure transformation, the external procedure function is the following, where the input parameters are in the member variable `m_pInParamVector` and the output values are in the member variable `m_pOutParamVector`:

```
INF_RESULT Tx<ModuleName>::MyFunc()
```

Property Access Functions

The property access functions provide information about the initialization properties associated with the External Procedure transformation. The initialization property names and values appear on the Initialization Properties tab when you edit the External Procedure transformation.

Informatica provides property access functions in both the base class and the `TINFConfigEntriesList` class. Use the `GetConfigEntryName()` and `GetConfigEntryValue()` functions in the `TINFConfigEntriesList` class to access the initialization property name and value, respectively.

Signature

Informatica provides the following functions in the base class:

```
TINFConfigEntriesList*
TINFBaseExternalModule60::accessConfigEntriesList();

const char* GetConfigEntry(const char* LHS);
```

Informatica provides the following functions in the `TINFConfigEntriesList` class:

```
const char* TINFConfigEntriesList::GetConfigEntryValue(const char* LHS);

const char* TINFConfigEntriesList::GetConfigEntryValue(int i);

const char* TINFConfigEntriesList::GetConfigEntryName(int i);

const char* TINFConfigEntriesList::GetConfigEntry(const char* LHS)
```

Note: In the `TINFConfigEntriesList` class, use the `GetConfigEntryName()` and `GetConfigEntryValue()` property access functions to access the initialization property names and values.

You can call these functions from a TX program. The TX program then converts this string value into a number, for example by using `atoi` or `sscanf`. In the following example, "addFactor" is an Initialization Property. `accessConfigEntriesList()` is a member variable of the TX base class and does not need to be defined.

```
const char* addFactorStr = accessConfigEntriesList()->
GetConfigEntryValue("addFactor");
```

Parameter Access Functions

Parameter access functions are datatype specific. Use the parameter access function GetDataType to return the datatype of a parameter. Then use a parameter access function corresponding to this datatype to return information about the parameter.

A parameter passed to an external procedure belongs to the datatype TINFParam*. The header file infparam.h defines the related access functions. The Designer generates stub code that includes comments indicating the parameter datatype. You can also determine the datatype of a parameter in the corresponding External Procedure transformation in the Designer.

Signature

A parameter passed to an external procedure is a pointer to an object of the TINFParam class. This fixed-signature function is a method of that class and returns the parameter datatype as an enum value.

The valid datatypes are:

INF_DATATYPE_LONG
INF_DATATYPE_STRING
INF_DATATYPE_DOUBLE
INF_DATATYPE_RAW
INF_DATATYPE_TIME

Table 6-5 lists a brief description of some parameter access functions:

Table 6-5. Descriptions of Parameter Access Functions

Parameter Access Function	Description
INF_DATATYPE GetDataType(void);	Gets the datatype of a parameter. Use the parameter datatype to determine which datatype-specific function to use when accessing parameter values.
INF_Boolean IsValid(void);	Verifies that input data is valid. Returns FALSE if the parameter contains truncated data and is a string.
INF_Boolean IsNULL(void);	Verifies that input data is NULL.
INF_Boolean IsInputMapped (void);	Verifies that input port passing data to this parameter is connected to a transformation.
INF_Boolean IsOutput Mapped (void);	Verifies that output port receiving data from this parameter is connected to a transformation.
INF_Boolean IsInput(void);	Verifies that parameter corresponds to an input port.
INF_Boolean IsOutput(void);	Verifies that parameter corresponds to an output port.
INF_Boolean GetName(void);	Gets the name of the parameter.

Table 6-5. Descriptions of Parameter Access Functions

Parameter Access Function	Description
SQLIndicator GetIndicator(void);	Gets the value of a parameter indicator. The IsValid and ISNULL functions are special cases of this function. This function can also return INF_SQL_DATA_TRUNCATED.
void SetIndicator(SQLIndicator Indicator);	Sets an output parameter indicator, such as invalid or truncated.
long GetLong(void);	Gets the value of a parameter having a Long or Integer datatype. Call this function only if you know the parameter datatype is Integer or Long. This function does not convert data to Long from another datatype.
double GetDouble(void);	Gets the value of a parameter having a Float or Double datatype. Call this function only if you know the parameter datatype is Float or Double. This function does not convert data to Double from another datatype.
char* GetString(void);	Gets the value of a parameter as a null-terminated string. Call this function only if you know the parameter datatype is String. This function does not convert data to String from another datatype. The value in the pointer changes when the next row of data is read. If you want to store the value from a row for later use, explicitly copy this string into its own allocated buffer.
char* GetRaw(void);	Gets the value of a parameter as a non-null terminated byte array. Call this function only if you know the parameter datatype is Raw. This function does not convert data to Raw from another datatype.
unsigned long GetActualDataLen(void);	Gets the current length of the array returned by GetRaw.
TINFTIME GetTime(void);	Gets the value of a parameter having a Date/Time datatype. Call this function only if you know the parameter datatype is Date/Time. This function does not convert data to Date/Time from another datatype.
void SetLong(long lVal);	Sets the value of an output parameter having a Long datatype.
void SetDouble(double dblVal);	Sets the value of an output parameter having a Double datatype.
void SetString(char* sVal);	Sets the value of an output parameter having a String datatype.
void SetRaw(char* rVal, size_t ActualDataLen);	Sets a non-null terminated byte array.
void SetTime(TINFTIME timeVal);	Sets the value of an output parameter having a Date/Time datatype.

Only use the SetInt32 or GetInt32 function when you run the external procedure on a 64-bit Integration Service. Do not use any of the following functions:

- ◆ GetLong
- ◆ SetLong
- ◆ GetpLong
- ◆ GetpDouble
- ◆ GetpTime

Pass the parameters using two parameter lists.

Table 6-6 lists the member variables of the external procedure base class.

Table 6-6. Member Variable of the External Procedure Base Class

Variable	Description
m_nInParamCount	Number of input parameters.
m_pInParamVector	Actual input parameter array.
m_nOutParamCount	Number of output parameters.
m_pOutParamVector	Actual output parameter array.

Note: Ports defined as input/output show up in both parameter lists.

Code Page Access Functions

Informatica provides two code page access functions that return the code page of the Integration Service and two that return the code page of the data the external procedure processes. When the Integration Service runs in Unicode mode, the string data passing to the external procedure program can contain multibyte characters. The code page determines how the external procedure interprets a multibyte character string. When the Integration Service runs in Unicode mode, data processed by the external procedure program must be two-way compatible with the Integration Service code page.

Signature

Use the following functions to obtain the Integration Service code page through the external procedure program. Both functions return equivalent information.

```
int GetServerCodePageID() const;  
const char* GetServerCodePageName() const;
```

Use the following functions to obtain the code page of the data the external procedure processes through the external procedure program. Both functions return equivalent information.

```
int GetDataCodePageID(); // returns 0 in case of error  
const char* GetDataCodePageName() const; // returns NULL in case of error
```

Transformation Name Access Functions

Informatica provides two transformation name access functions that return the name of the External Procedure transformation. The GetWidgetName() function returns the name of the transformation, and the GetWidgetInstanceName() function returns the name of the transformation instance in the mapplet or mapping.

Signature

The char* returned by the transformation name access functions is an MBCS string in the code page of the Integration Service. It is not in the data code page.

```
const char* GetWidgetInstanceName() const;  
const char* GetWidgetName() const;
```

Procedure Access Functions

Informatica provides two procedure access functions that provide information about the external procedure associated with the External Procedure transformation. The GetProcedureName() function returns the name of the external procedure specified in the Procedure Name field of the External Procedure transformation. The GetProcedureIndex() function returns the index of the external procedure.

Signature

Use the following function to get the name of the external procedure associated with the External Procedure transformation:

```
const char* GetProcedureName() const;
```

Use the following function to get the index of the external procedure associated with the External Procedure transformation:

```
inline unsigned long GetProcedureIndex() const;
```

Partition Related Functions

Use partition related functions for external procedures in sessions with multiple partitions. When you partition a session that contains External Procedure transformations, the Integration Service creates instances of these transformations for each partition. For example, if you define five partitions for a session, the Integration Service creates five instances of each external procedure at session runtime.

Signature

Use the following function to obtain the number of partitions in a session:

```
unsigned long GetNumberOfPartitions();
```

Use the following function to obtain the index of the partition that called this external procedure:

```
unsigned long GetPartitionIndex();
```

Tracing Level Function

The tracing level function returns the session trace level, for example:

```
typedef enum
{
    TRACE_UNSET = 0,
    TRACE_TERSE = 1,
    TRACE_NORMAL = 2,
    TRACE_VERBOSE_INIT = 3,
    TRACE_VERBOSE_DATA = 4
} TracingLevelType;
```

Signature

Use the following function to return the session trace level:

```
TracingLevelType GetSessionTraceLevel();
```


Chapter 7

Filter Transformation

This chapter includes the following topics:

- ◆ Overview, 190
- ◆ Filter Condition, 192
- ◆ Creating a Filter Transformation, 193
- ◆ Tips, 195
- ◆ Troubleshooting, 196

Overview

Transformation type:

Active

Connected

You can filter rows in a mapping with the Filter transformation. You pass all the rows from a source transformation through the Filter transformation, and then enter a filter condition for the transformation. All ports in a Filter transformation are input/output, and only rows that meet the condition pass through the Filter transformation.

In some cases, you need to filter data based on one or more conditions before writing it to targets. For example, if you have a human resources target containing information about current employees, you might want to filter out employees who are part-time and hourly.

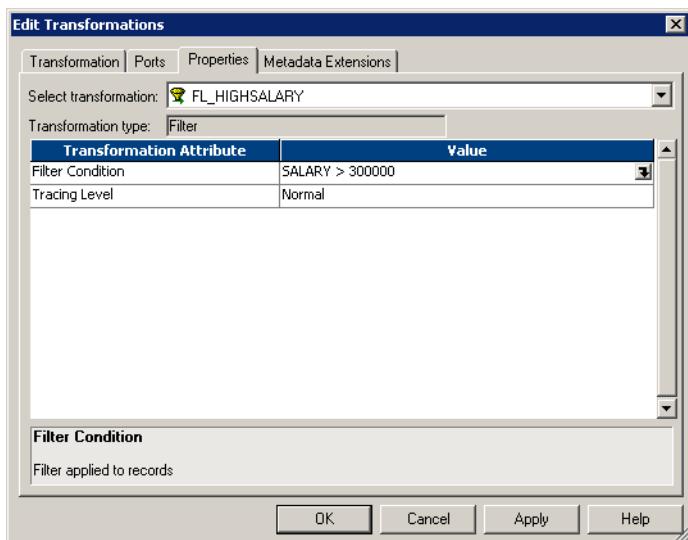
The mapping in Figure 7-1 passes the rows from a human resources table that contains employee data through a Filter transformation. The filter only allows rows through for employees that make salaries of \$30,000 or higher.

Figure 7-1. Sample Mapping with a Filter Transformation



Figure 7-2 shows the filter condition used in the mapping in Figure 7-1 on page 190:

Figure 7-2. Specifying a Filter Condition in a Filter Transformation



With the filter of SALARY > 30000, only rows of data where employees that make salaries greater than \$30,000 pass through to the target.

As an active transformation, the Filter transformation may change the number of rows passed through it. A filter condition returns TRUE or FALSE for each row that passes through the transformation, depending on whether a row meets the specified condition. Only rows that return TRUE pass through this transformation. Discarded rows do not appear in the session log or reject files.

To maximize session performance, include the Filter transformation as close to the sources in the mapping as possible. Rather than passing rows you plan to discard through the mapping, you then filter out unwanted data early in the flow of data from sources to targets.

You cannot concatenate ports from more than one transformation into the Filter transformation. The input ports for the filter must come from a single transformation. The Filter transformation does not allow setting output default values.

Filter Condition

You use the transformation language to enter the filter condition. The condition is an expression that returns TRUE or FALSE. For example, if you want to filter *out* rows for employees whose salary is less than \$30,000, you enter the following condition:

```
SALARY > 30000
```

You can specify multiple components of the condition, using the AND and OR logical operators. If you want to filter out employees who make less than \$30,000 and more than \$100,000, you enter the following condition:

```
SALARY > 30000 AND SALARY < 100000
```

You do not need to specify TRUE or FALSE as values in the expression. TRUE and FALSE are implicit return values from any condition you set. If the filter condition evaluates to NULL, the row is assumed to be FALSE.

Enter conditions using the Expression Editor, available from the Properties tab of the Filter transformation. The filter condition is case sensitive. Any expression that returns a single value can be used as a filter. You can also enter a constant for the filter condition. The numeric equivalent of FALSE is zero (0). Any non-zero value is the equivalent of TRUE. For example, if you have a port called NUMBER_OF_UNITS with a numeric datatype, a filter condition of NUMBER_OF_UNITS returns FALSE if the value of NUMBER_OF_UNITS equals zero. Otherwise, the condition returns TRUE.

After entering the expression, you can validate it by clicking the Validate button in the Expression Editor. When you enter an expression, validate it before continuing to avoid saving an invalid mapping to the repository. If a mapping contains syntax errors in an expression, you cannot run any session that uses the mapping until you correct the error.

Creating a Filter Transformation

Creating a Filter transformation requires inserting the new transformation into the mapping, adding the appropriate input/output ports, and writing the condition.

To create a Filter transformation:

1. In the Designer, switch to the Mapping Designer and open a mapping.

2. Click Transformation > Create.

Select Filter transformation, and enter the name of the new transformation. The naming convention for the Filter transformation is FIL_*TransformationName*. Click Create, and then click Done.

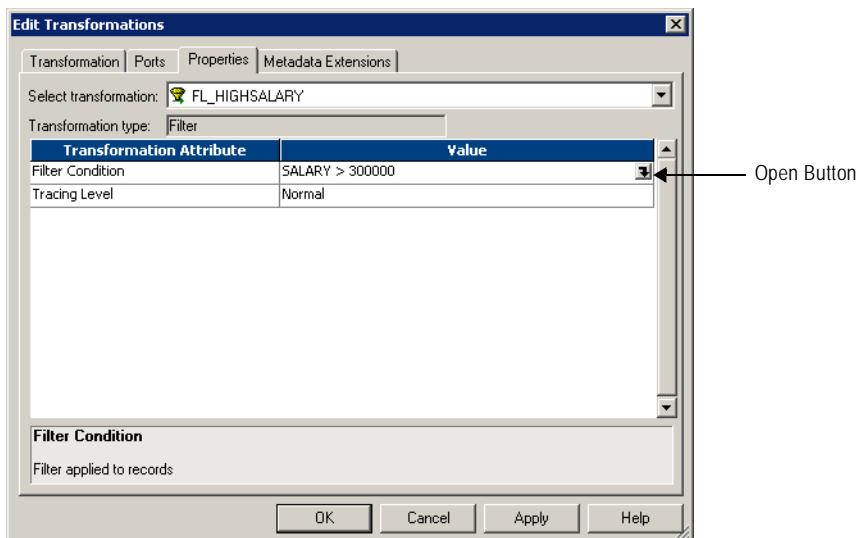
3. Select and drag all the ports from a source qualifier or other transformation to add them to the Filter transformation.

After you select and drag ports, copies of these ports appear in the Filter transformation. Each column has both an input and an output port.

4. Double-click the title bar of the new transformation.

5. Click the Properties tab.

A default condition appears in the list of conditions. The default condition is TRUE (a constant with a numeric value of 1).



6. Click the Value section of the condition, and then click the Open button.

The Expression Editor appears.

7. Enter the filter condition you want to apply.

Use values from one of the input ports in the transformation as part of this condition. However, you can also use values from output ports in other transformations.

8. Click Validate to check the syntax of the conditions you entered.

You may have to fix syntax errors before continuing.

9. Click OK.

10. Select the Tracing Level, and click OK to return to the Mapping Designer.

11. Click Repository > Save to save the mapping.

Tips

Use the Filter transformation early in the mapping.

To maximize session performance, keep the Filter transformation as close as possible to the sources in the mapping. Rather than passing rows that you plan to discard through the mapping, you can filter out unwanted data early in the flow of data from sources to targets.

Use the Source Qualifier transformation to filter.

The Source Qualifier transformation provides an alternate way to filter rows. Rather than filtering rows from within a mapping, the Source Qualifier transformation filters rows when read from a source. The main difference is that the source qualifier limits *the row set extracted from a source*, while the Filter transformation limits *the row set sent to a target*. Since a source qualifier reduces the number of rows used throughout the mapping, it provides better performance.

However, the Source Qualifier transformation only lets you filter rows from relational sources, while the Filter transformation filters rows from any type of source. Also, note that since it runs in the database, you must make sure that the filter condition in the Source Qualifier transformation only uses standard SQL. The Filter transformation can define a condition using any statement or transformation function that returns either a TRUE or FALSE value.

For more information about setting a filter for a Source Qualifier transformation, see “Source Qualifier Transformation” on page 425.

Troubleshooting

I imported a flat file into another database (Microsoft Access) and used SQL filter queries to determine the number of rows to import into the Designer. But when I import the flat file into the Designer and pass data through a Filter transformation using equivalent SQL statements, I do not import as many rows. Why is there a difference?

You might want to check two possible solutions:

- ◆ **Case sensitivity.** The filter condition is case sensitive, and queries in some databases do not take this into account.
- ◆ **Appended spaces.** If a field contains additional spaces, the filter condition needs to check for additional spaces for the length of the field. Use the RTRIM function to remove additional spaces.

How do I filter out rows with null values?

To filter out rows containing null values or spaces, use the ISNULL and IS_SPACES functions to test the value of the port. For example, if you want to filter out rows that contain NULLs in the FIRST_NAME port, use the following condition:

```
IIF(ISNULL(FIRST_NAME), FALSE, TRUE)
```

This condition states that if the FIRST_NAME port is NULL, the return value is FALSE and the row should be discarded. Otherwise, the row passes through to the next transformation.

For more information about the ISNULL and IS_SPACES functions, see “Functions” in the *Transformation Language Reference*.

Chapter 8

HTTP Transformation

This chapter includes the following topics:

- ◆ Overview, 198
- ◆ Creating an HTTP Transformation, 200
- ◆ Configuring the Properties Tab, 202
- ◆ Configuring the HTTP Tab, 204
- ◆ Examples, 209

Overview

Transformation type:

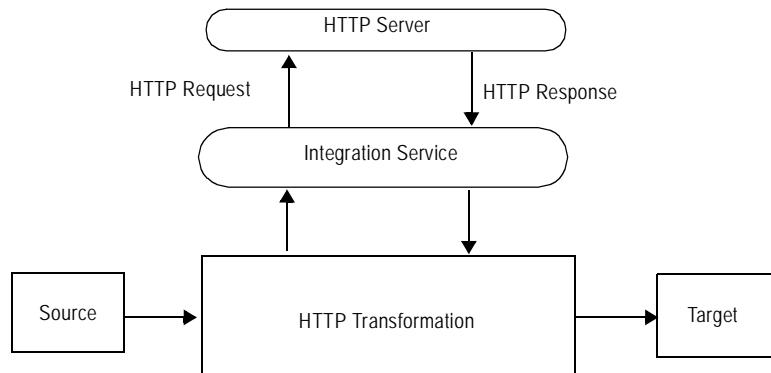
Passive
Connected

The HTTP transformation enables you to connect to an HTTP server to use its services and applications. When you run a session with an HTTP transformation, the Integration Service connects to the HTTP server and issues a request to retrieve data from or update data on the HTTP server, depending on how you configure the transformation:

- ◆ **Read data from an HTTP server.** When the Integration Service reads data from an HTTP server, it retrieves the data from the HTTP server and passes the data to the target or a downstream transformation in the mapping. For example, you can connect to an HTTP server to read current inventory data, perform calculations on the data during the PowerCenter session, and pass the data to the target.
- ◆ **Update data on the HTTP server.** When the Integration Service writes to an HTTP server, it posts data to the HTTP server and passes HTTP server responses to the target or a downstream transformation in the mapping. For example, you can post data providing scheduling information from upstream transformations to the HTTP server during a session.

Figure 8-1 shows how the Integration Service processes an HTTP transformation:

Figure 8-1. HTTP Transformation Processing



The Integration Service passes data from upstream transformations or the source to the HTTP transformation, reads a URL configured in the HTTP transformation or application connection, and sends an HTTP request to the HTTP server to either read or update data.

Requests contain header information and may contain body information. The header contains information such as authentication parameters, commands to activate programs or web services residing on the HTTP server, and other information that applies to the entire HTTP request. The body contains the data the Integration Service sends to the HTTP server.

When the Integration Service sends a request to read data, the HTTP server sends back an HTTP response with the requested data. The Integration Service sends the requested data to downstream transformations or the target.

When the Integration Service sends a request to update data, the HTTP server writes the data it receives and sends back an HTTP response that the update succeeded. The Integration Service then sends the HTTP response to downstream transformations or the target.

You can configure the HTTP transformation for the headers of HTTP responses. HTTP response body data passes through the **HTTPOUT** output port.

Authentication

The HTTP transformation uses the following forms of authentication:

- ◆ **Basic.** Based on a non-encrypted user name and password.
- ◆ **Digest.** Based on an encrypted user name and password.
- ◆ **NTLM.** Based on encrypted user name, password, and domain.

Connecting to the HTTP Server

When you configure an HTTP transformation, you can configure the URL for the connection. You can also create an HTTP connection object in the Workflow Manager. Configure an HTTP application connection in the following circumstances:

- ◆ The HTTP server requires authentication.
- ◆ You want to configure the connection timeout.
- ◆ You want to override the base URL in the HTTP transformation.

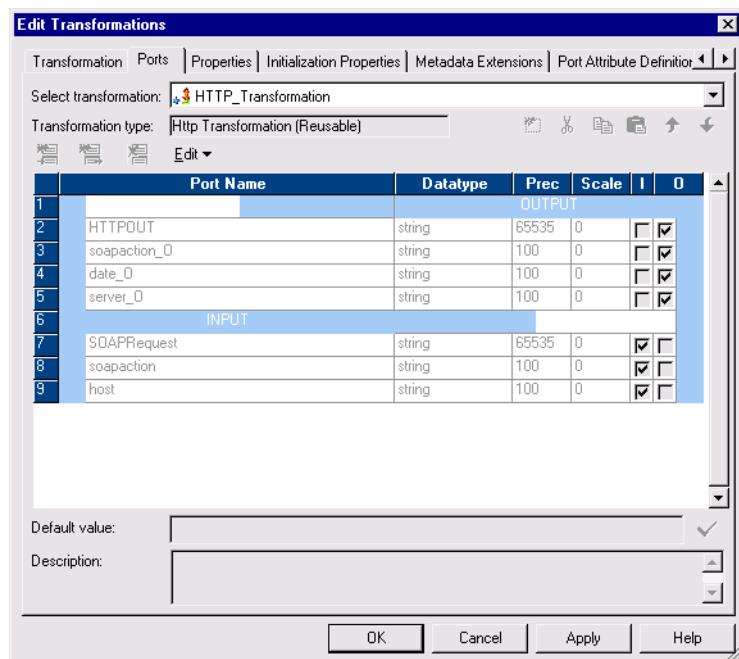
Creating an HTTP Transformation

You create HTTP transformations in the Transformation Developer or in the Mapping Designer. An HTTP transformation has the following tabs:

- ◆ **Transformation.** Configure the name and description for the transformation.
- ◆ **Ports.** View input and output ports for the transformation. You cannot add or edit ports on the Ports tab. The Designer creates ports on the Ports tab when you add ports to the header group on the HTTP tab. For more information, see “Configuring Groups and Ports” on page 205.
- ◆ **Properties.** Configure properties for the HTTP transformation on the Properties tab. For more information, see “Configuring the Properties Tab” on page 202.
- ◆ **Initialization Properties.** You can define properties that the external procedure uses at run time, such as during initialization. For more information about creating initialization properties, see “Working with Procedure Properties” on page 72.
- ◆ **Metadata Extensions.** You can specify the property name, datatype, precision, and value. Use metadata extensions for passing information to the procedure. For more information about creating metadata extensions, see “Metadata Extensions” in the *Repository Guide*.
- ◆ **Port Attribute Definitions.** You can view port attributes for HTTP transformation ports. You cannot edit port attribute definitions.
- ◆ **HTTP.** Configure the method, ports, and URL on the HTTP tab. For more information, see “Configuring the HTTP Tab” on page 204.

Figure 8-2 shows an HTTP transformation:

Figure 8-2. HTTP Transformation



To create an HTTP transformation:

1. In the Transformation Developer or Mapping Designer, click Transformation > Create.
 2. Select HTTP transformation.
 3. Enter a name for the transformation.
 4. Click Create.
- The HTTP transformation displays in the workspace.
5. Click Done.
 6. Configure the tabs in the transformation.

Configuring the Properties Tab

The HTTP transformation is built using the Custom transformation. Some Custom transformation properties do not apply to the HTTP transformation or are not configurable.

Figure 8-3 shows the Properties tab of an HTTP transformation:

Figure 8-3. HTTP Transformation Properties Tab

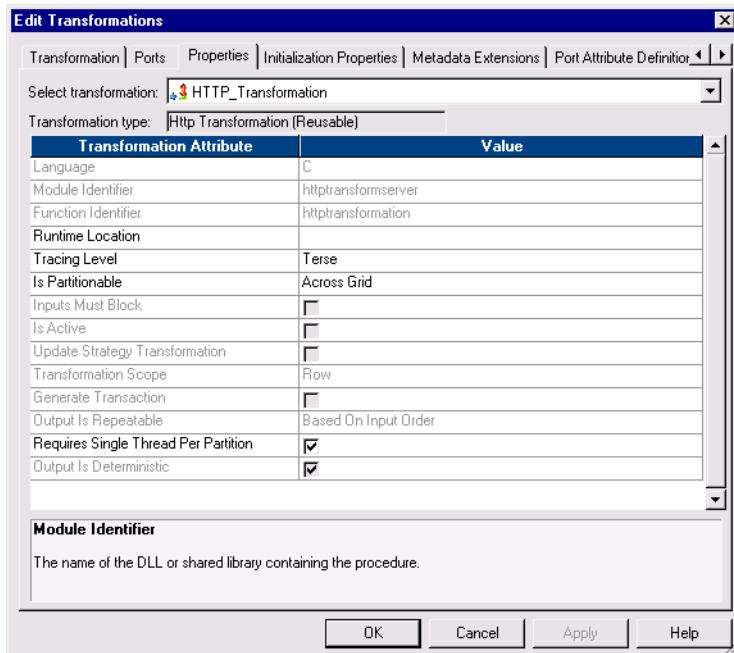


Table 8-1 describes the HTTP transformation properties that you can configure:

Table 8-1. HTTP Transformation Properties

Option	Description
Runtime Location	Location that contains the DLL or shared library. Default is \$PMExtProcDir. Enter a path relative to the Integration Service machine that runs the session using the HTTP transformation. If you make this property blank, the Integration Service uses the environment variable defined on the Integration Service machine to locate the DLL or shared library. You must copy all DLLs or shared libraries to the runtime location or to the environment variable defined on the Integration Service machine. The Integration Service fails to load the procedure when it cannot locate the DLL, shared library, or a referenced file.
Tracing Level	Amount of detail displayed in the session log for this transformation. Default is Normal.

Table 8-1. HTTP Transformation Properties

Option	Description
Is Partitionable	<p>Indicates if you can create multiple partitions in a pipeline that uses this transformation:</p> <ul style="list-style-type: none">- No. The transformation cannot be partitioned. The transformation and other transformations in the same pipeline are limited to one partition.- Locally. The transformation can be partitioned, but the Integration Service must run all partitions in the pipeline on the same node. Choose Local when different partitions of the Custom transformation must share objects in memory.- Across Grid. The transformation can be partitioned, and the Integration Service can distribute each partition to different nodes. <p>Default is No.</p> <p>For more information about using partitioning, see the <i>Workflow Administration Guide</i>.</p>
Requires Single Thread Per Partition	<p>Indicates if the Integration Service processes each partition at the procedure with one thread. When you enable this option, the procedure code can use thread-specific operations.</p> <p>Default is enabled.</p> <p>For more information about writing thread-specific operations, see "Working with Thread-Specific Procedure Code" on page 66.</p>

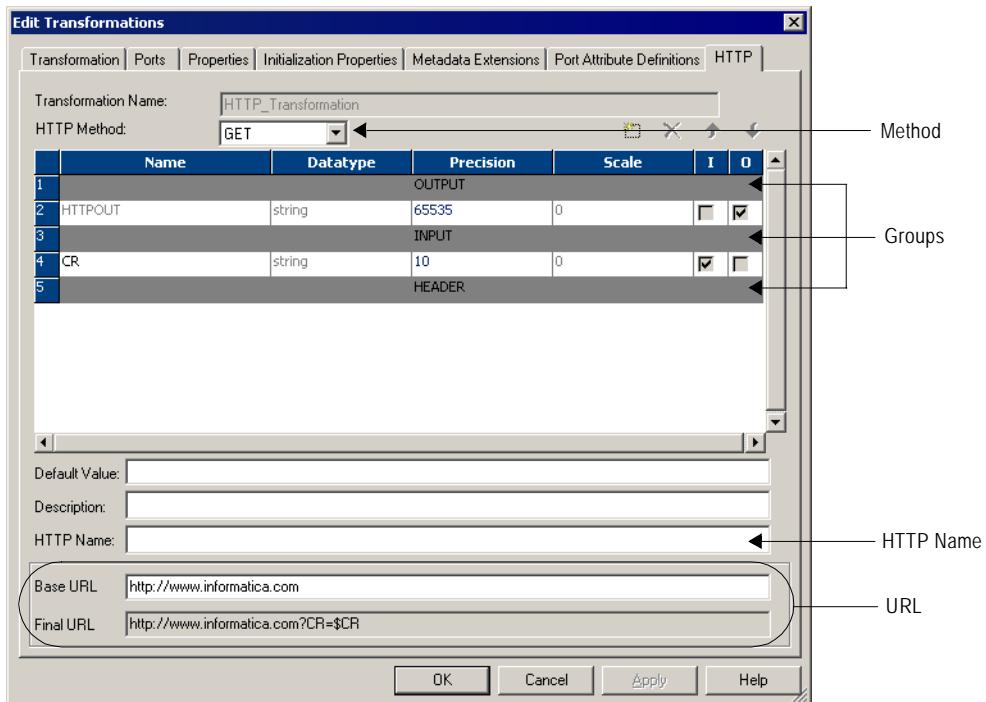
Configuring the HTTP Tab

On the HTTP tab, you can configure the transformation to read data from the HTTP server or write data to the HTTP server. Configure the following information on the HTTP tab:

- ♦ **Select the method.** Select GET, POST, or SIMPLE POST method based on whether you want to read data from or write data to an HTTP server. For more information, see “Selecting a Method” on page 204.
- ♦ **Configure groups and ports.** Manage HTTP request/response body and header details by configuring input and output ports. You can also configure port names with special characters. For more information, see “Configuring Groups and Ports” on page 205.
- ♦ **Configure a base URL.** Configure the base URL for the HTTP server you want to connect to. For more information, see “Configuring a URL” on page 207.

Figure 8-4 shows the HTTP tab of an HTTP transformation:

Figure 8-4. HTTP Transformation HTTP Tab



Selecting a Method

The groups and ports you define in a transformation depend on the method you select. To read data from an HTTP server, select the GET method. To write data to an HTTP server, select the POST or SIMPLE POST method.

Table 8-2 explains the different methods:

Table 8-2. HTTP Transformation Methods

Method	Description
GET	Reads data from an HTTP server.
POST	Writes data from multiple input ports to the HTTP server.
SIMPLE POST	A simplified version of the POST method. Writes data from one input port as a single block of data to the HTTP server.

To define the metadata for the HTTP request, you must configure input and output ports based on the method you select:

- ◆ **GET method.** Use the input group to add input ports that the Designer uses to construct the final URL for the HTTP server.
- ◆ **POST or SIMPLE POST method.** Use the input group for the data that defines the body of the HTTP request.

For all methods, use the header group for the HTTP request header information.

Configuring Groups and Ports

The ports you add to an HTTP transformation depend on the method you choose and the group. An HTTP transformation uses the following groups:

- ◆ **Output.** Contains body data for the HTTP response. Passes responses from the HTTP server to downstream transformations or the target. By default, contains one output port, **HTTPOUT**. You cannot add ports to the output group. You can modify the precision for the **HTTPOUT** output port.
- ◆ **Input.** Contains body data for the HTTP request. Also contains metadata the Designer uses to construct the final URL to connect to the HTTP server. To write data to an HTTP server, the input group passes body information to the HTTP server. By default, contains one input port.
- ◆ **Header.** Contains header data for the request and response. Passes header information to the HTTP server when the Integration Service sends an HTTP request. Ports you add to the header group pass data for HTTP headers. When you add ports to the header group the Designer adds ports to the input and output groups on the Ports tab. By default, contains no ports.

Note: The data that passes through an HTTP transformation must be of the String datatype. String data includes any markup language common in HTTP communication, such as HTML and XML.

Table 8-3 describes the groups and ports for the GET method:

Table 8-3. GET Method Groups and Ports

Request/ Response	Group	Description
REQUEST	Input	The Designer uses the names and values of the input ports to construct the final URL.
	Header	You can configure input and input/output ports for HTTP requests. The Designer adds ports to the input and output groups based on the ports you add to the header group: <ul style="list-style-type: none">- Input group. Creates input ports based on input and input/output ports from the header group.- Output group. Creates output ports based on input/output ports from the header group.
RESPONSE	Header	You can configure output and input/output ports for HTTP responses. The Designer adds ports to the input and output groups based on the ports you add to the header group: <ul style="list-style-type: none">- Input group. Creates input ports based on input/output ports from the header group.- Output group. Creates output ports based on output and input/output ports from the header group.
	Output	All body data for an HTTP response passes through the HTTPOUT output port.

Table 8-4 describes the ports for the POST method:

Table 8-4. POST Method Groups and Ports

Request/ Response	Group	Description
REQUEST	Input	You can add multiple ports to the input group. Body data for an HTTP request can pass through one or more input ports based on what you add to the header group.
	Header	You can configure input and input/output ports for HTTP requests. The Designer adds ports to the input and output groups based on the ports you add to the header group: <ul style="list-style-type: none">- Input group. Creates input ports based on input and input/output ports from the header group.- Output group. Creates output ports based on input/output ports from the header group.
RESPONSE	Header	You can configure output and input/output ports for HTTP responses. The Designer adds ports to the input and output groups based on the ports you add to the header group: <ul style="list-style-type: none">- Input group. Creates input ports based on input/output ports from the header group.- Output group. Creates output ports based on output and input/output ports from the header group.
	Output	All body data for an HTTP response passes through the HTTPOUT output port.

Table 8-5 describes the ports for the SIMPLE POST method:

Table 8-5. SIMPLE POST Method Groups and Ports

Request/ Response	Group	Description
REQUEST	Input	You can add one input port. Body data for an HTTP request can pass through one input port.
	Header	You can configure input and input/output ports for HTTP requests. The Designer adds ports to the input and output groups based on the ports you add to the header group: <ul style="list-style-type: none">- Input group. Creates input ports based on input and input/output ports from the header group.- Output group. Creates output ports based on input/output ports from the header group.
RESPONSE	Header	You can configure output and input/output ports for HTTP responses. The Designer adds ports to the input and output groups based on the ports you add to the header group: <ul style="list-style-type: none">- Input group. Creates input ports based on input/output ports from the header group.- Output group. Creates output ports based on output and input/output ports from the header group.
	Output	All body data for an HTTP response passes through the HTTPOUT output port.

Adding an HTTP Name

The Designer does not allow special characters, such as a dash (-), in port names. If you need to use special characters in a port name, you can configure an HTTP name to override the name of a port. For example, if you want an input port named Content-type, you can name the port ContentType and enter Content-Type as the HTTP name.

Configuring a URL

After you select a method and configure input and output ports, you must configure a URL. Enter a base URL, and the Designer constructs the final URL. If you select the GET method, the final URL contains the base URL and parameters based on the port names in the input group. If you select the POST or SIMPLE POST methods, the final URL is the same as the base URL.

You can also specify a URL when you configure an HTTP application connection. The base URL specified in the HTTP application connection overrides the base URL specified in the HTTP transformation.

Note: An HTTP server can redirect an HTTP request to another HTTP server. When this occurs, the HTTP server sends a URL back to the Integration Service, which then establishes a connection to the other HTTP server. The Integration Service can establish a maximum of five additional connections.

Final URL Construction for GET Method

The Designer constructs the final URL for the GET method based on the base URL and port names in the input group. It appends HTTP arguments to the base URL to construct the final URL in the form of an HTTP query string. A query string consists of a question mark

(?), followed by name/value pairs. The Designer appends the question mark and the name/value pairs that correspond to the names and values of the input ports you add to the input group.

When you select the GET method and add input ports to the input group, the Designer appends the following group and port information to the base URL to construct the final URL:

```
?<input group input port 1 name> = $<input group input port 1 value>
```

For each input port following the first input group input port, the Designer appends the following group and port information:

```
& <input group input port n name> = $<input group input port n value>
```

where *n* represents the input port.

For example, if you enter `www.company.com` for the base URL and add the input ports ID, EmpName, and Department to the input group, the Designer constructs the following final URL:

```
www.company.com?ID=$ID&EmpName=$EmpName&Department=$Department
```

You can edit the final URL to modify or add operators, variables, or other arguments. For more information about HTTP requests and query string, see <http://www.w3c.org>.

Examples

This section contains examples for each type of method:

- ◆ GET
- ◆ POST
- ◆ SIMPLE POST

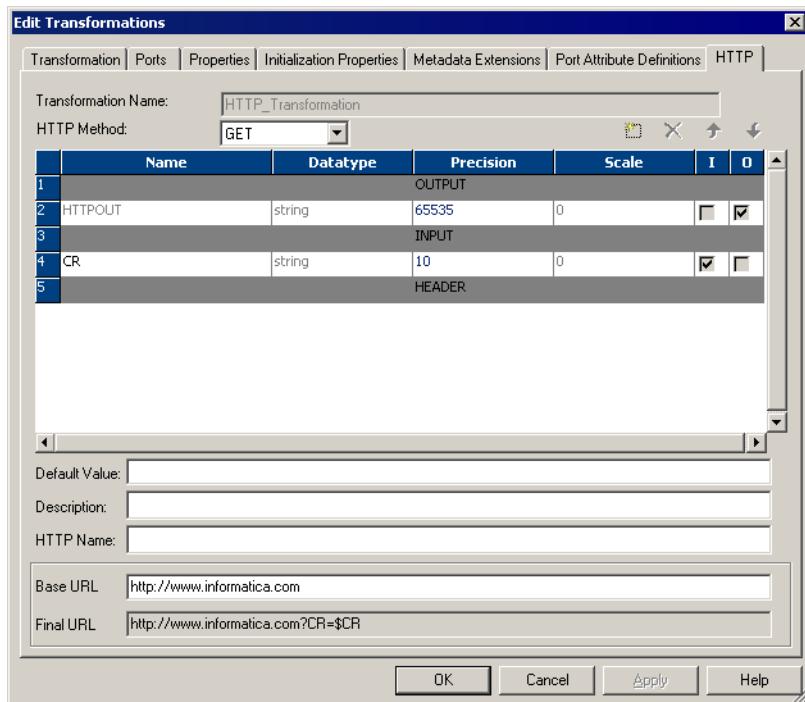
GET Example

The source file used with this example contains the following data:

```
78576  
78577  
78578
```

Figure 8-5 shows the HTTP tab of the HTTP transformation for the GET example:

Figure 8-5. HTTP Tab for a GET Example



The Designer appends a question mark (?), the input group input port name, a dollar sign (\$), and the input group input port name again to the base URL to construct the final URL:

```
http://www.informatica.com?CR=$CR
```

The Integration Service sends the source file values to the CR input port of the HTTP transformation and sends the following HTTP requests to the HTTP server:

```
http://www.informatica.com?CR=78576  
http://www.informatica.com?CR=78577  
http://www.informatica.com?CR=78578
```

The HTTP server sends an HTTP response back to the Integration Service, which sends the data through the output port of the HTTP transformation to the target.

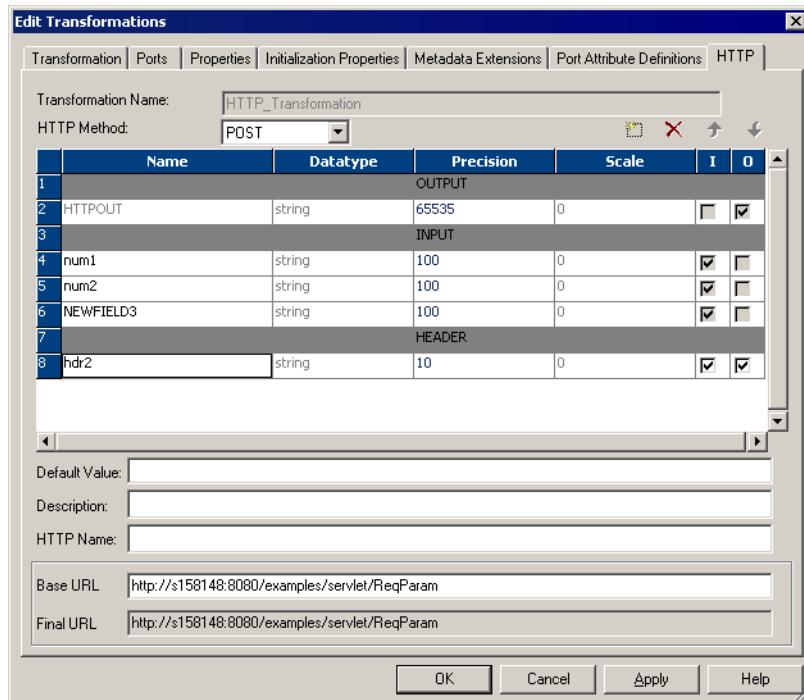
POST Example

The source file used with this example contains the following data:

```
33,44,1  
44,55,2  
100,66,0
```

Figure 8-6 shows that each field in the source file has a corresponding input port:

Figure 8-6. HTTP Tab for a POST Example



The Integration Service sends the values of the three fields for each row through the input ports of the HTTP transformation and sends the HTTP request to the HTTP server specified in the final URL.

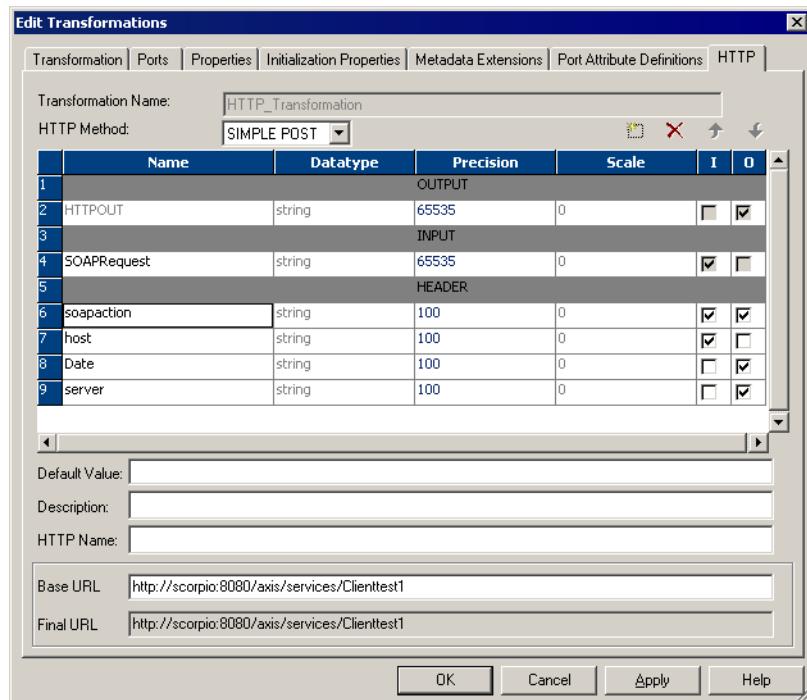
SIMPLE POST Example

The following text shows the XML file used with this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<n4:Envelope xmlns:cli="http://localhost:8080/axis/Clienttest1.jws"
  xmlns:n4="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://
  schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <n4:Header>
  </n4:Header>
  <n4:Body n4:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/
  "><cli:smplssource>
    <Metadatainfo xsi:type="xsd:string">smplssourceRequest.Metadatainfo106</
    Metadatainfo></cli:smplssource>
  </n4:Body>
</n4:Envelope>, capeconnect:Clienttest1services:Clienttest1#smplssource
```

Figure 8-7 shows the HTTP tab of the HTTP transformation for the SIMPLE POST example:

Figure 8-7. HTTP Tab for a SIMPLE POST Example



The Integration Service sends the body of the source file through the input port and sends the HTTP request to the HTTP server specified in the final URL.

Chapter 9

Java Transformation

This chapter includes the following topics:

- ◆ Overview, 214
- ◆ Using the Java Code Tab, 217
- ◆ Configuring Ports, 219
- ◆ Configuring Java Transformation Properties, 221
- ◆ Developing Java Code, 225
- ◆ Configuring Java Transformation Settings, 229
- ◆ Compiling a Java Transformation, 231
- ◆ Fixing Compilation Errors, 232

Overview

Transformation type:

Active/Passive
Connected

You can extend PowerCenter functionality with the Java transformation. The Java transformation provides a simple native programming interface to define transformation functionality with the Java programming language. You can use the Java transformation to quickly define simple or moderately complex transformation functionality without advanced knowledge of the Java programming language or an external Java development environment.

For example, you can define transformation logic to loop through input rows and generate multiple output rows based on a specific condition. You can also use expressions, user-defined functions, unconnected transformations, and mapping variables in the Java code.

You create Java transformations by writing Java code snippets that define transformation logic. You can use Java transformation API methods and standard Java language constructs. For example, you can use static code and variables, instance variables, and Java methods. You can use third-party Java APIs, built-in Java packages, or custom Java packages. You can also define and use Java expressions to call expressions from within a Java transformation.

For more information about the Java transformation API methods, see “Java Transformation API Reference” on page 237. For more information about using Java expressions, see “Java Expressions” on page 263.

The PowerCenter Client uses the Java Development Kit (JDK) to compile the Java code and generate byte code for the transformation. The Integration Service uses the Java Runtime Environment (JRE) to execute generated byte code at run time. When you run a session with a Java transformation, the Integration Service uses the JRE to execute the byte code and process input rows and generate output rows.

You can define transformation behavior for a Java transformation based on the following events:

- ◆ The transformation receives an input row
- ◆ The transformation has processed all input rows
- ◆ The transformation receives a transaction notification such as commit or rollback

Steps to Define a Java Transformation

Complete the following steps to write and compile Java code and fix compilation errors in a Java transformation:

1. Create the transformation in the Transformation Developer or Mapping Designer.
2. Configure input and output ports and groups for the transformation. Use port names as variables in Java code snippets. For more information, see “Configuring Ports” on page 219.

- Configure the transformation properties. For more information, see “Configuring Java Transformation Properties” on page 221.
- Use the code entry tabs in the transformation to write and compile the Java code for the transformation. For more information, see “Developing Java Code” on page 225 and “Compiling a Java Transformation” on page 231.
- Locate and fix compilation errors in the Java code for the transformation. For more information, see “Fixing Compilation Errors” on page 232.

Active and Passive Java Transformations

You can create active and passive Java transformations. You select the type of Java transformation when you create the transformation. After you set the transformation type, you cannot change it. Active and passive Java transformations run the Java code in the On Input Row tab for the transformation one time for each row of input data.

Use an active transformation when you want to generate more than one output row for each input row in the transformation. You must use Java transformation generateRow API to generate an output row. For example, a Java transformation contains two input ports that represent a start date and an end date. You can generate an output row for each date between the start date and end date.

Use a passive transformation when you need one output row for each input row in the transformation. Passive transformations generate an output row after processing each input row.

Datatype Mapping

The Java transformation maps PowerCenter datatypes to Java primitives, based on the Java transformation port type. The Java transformation maps input port datatypes to Java primitives when it reads input rows, and it maps Java primitives to output port datatypes when it writes output rows.

For example, if an input port in a Java transformation has an Integer datatype, the Java transformation maps it to an integer primitive. The transformation treats the value of the input port as Integer in the transformation, and maps the Integer primitive to an integer datatype when the transformation generates the output row.

Table 9-1 shows the mapping between PowerCenter datatypes and Java primitives by a Java transformation:

Table 9-1. Mapping from PowerCenter Datatypes to Java Datatypes

PowerCenter Datatype	Java Datatype
CHAR	String
BINARY	byte[]
LONG (INT32)	int

Table 9-1. Mapping from PowerCenter Datatypes to Java Datatypes

PowerCenter Datatype	Java Datatype
DOUBLE	double
DECIMAL	double * BigDecimal
Date/Time	long (number of milliseconds since January 1, 1970 00:00:00.000 GMT)

* For more information about configuring the Java datatype for PowerCenter Decimal datatypes, see “Enabling High Precision” on page 230.

String and byte[] are object datatypes in Java. Int, double, and long are primitive datatypes.

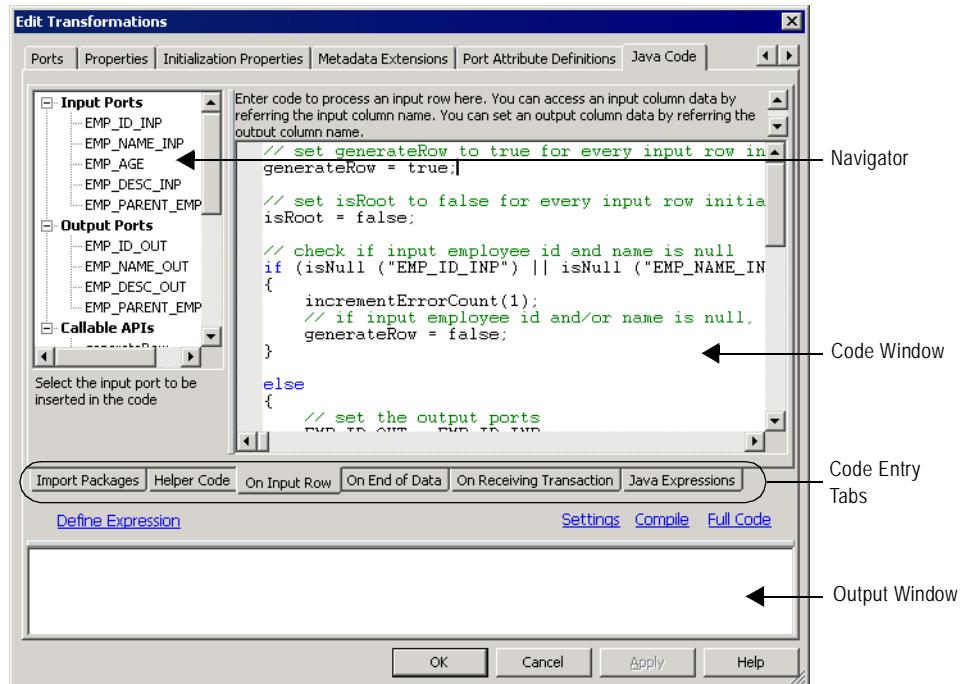
Using the Java Code Tab

Use the Java Code tab to define, compile, and fix compilation errors in Java code. You can use code snippets that Java packages, define static code or a static block, instance variables, and user-defined methods, define and call Java expressions, and define transformation logic. Create code snippets in the code entry tabs.

After you develop code snippets, you can compile the Java code and view the results of the compilation in the Output window or view the full Java code.

Figure 9-1 shows the components of the Java Code tab:

Figure 9-1. Java Code Tab Components



The Java Code tab contains the following components:

- ◆ **Navigator.** Add input or output ports or APIs to a code snippet. The Navigator lists the input and output ports for the transformation, the available Java transformation APIs, and a description of the port or API function. For input and output ports, the description includes the port name, type, datatype, precision, and scale. For API functions, the description includes the syntax and use of the API function.

The Navigator disables any port or API function that is unavailable for the code entry tab. For example, you cannot add ports or call API functions from the Import Packages code entry tab.

For more information about using the Navigator when you develop Java code, see “Developing Java Code” on page 225.

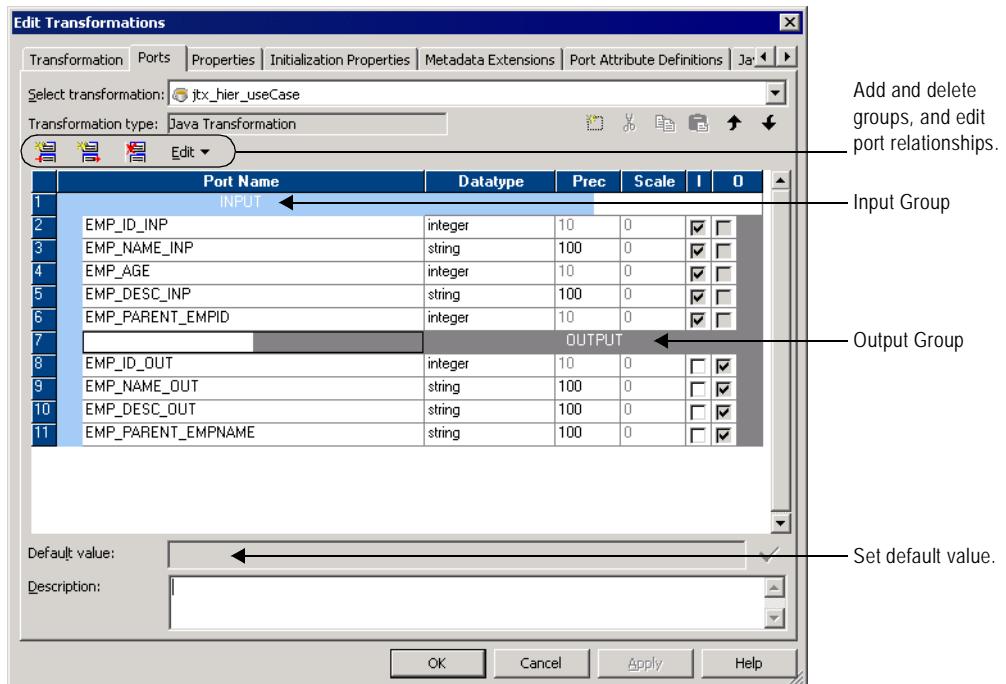
- ◆ **Code window.** Develop Java code for the transformation. The code window uses basic Java syntax highlighting. For more information, see “Developing Java Code” on page 225.
- ◆ **Code entry tabs.** Define transformation behavior. Each code entry tab has an associated Code window. To enter Java code for a code entry tab, click the tab and write Java code in the Code window. For more information about the code entry tabs, see “Developing Java Code” on page 225.
- ◆ **Define Expression link.** Launches the Define Expression dialog box that you use to create Java expressions. For more information about creating and using Java expressions, see “Java Expressions” on page 263.
- ◆ **Settings link.** Launches the Settings dialog box that you use to set the classpath for third-party and custom Java packages and to enable high precision for Decimal datatypes. For more information, see “Configuring Java Transformation Settings” on page 229.
- ◆ **Compile link.** Compiles the Java code for the transformation. Output from the Java compiler, including error and informational messages, appears in the Output window. For more information about compiling Java transformations, see “Compiling a Java Transformation” on page 231.
- ◆ **Full Code link.** Opens the Full Code window to display the complete class code for the Java transformation. The complete code for the transformation includes the Java code from the code entry tabs added to the Java transformation class template. For more information about using the Full Code window, see “Fixing Compilation Errors” on page 232.
- ◆ **Output window.** Displays the compilation results for the Java transformation class. You can right-click an error message in the Output window to locate the error in the snippet code or the full code for the Java transformation class in the Full Code window. You can also double-click an error in the Output window to locate the source of the error. For more information about using the Output window to troubleshoot compilation errors, see “Fixing Compilation Errors” on page 232.

Configuring Ports

A Java transformation can have input ports, output ports, and input/output ports. You create and edit groups and ports on the Ports tab. You can specify default values for ports. After you add ports to a transformation, use the port names as variables in Java code snippets.

Figure 9-2 shows the Ports tab for a Java transformation with one input group and one output group:

Figure 9-2. Java Transformation Ports Tab



Creating Groups and Ports

When you create a Java transformation, it includes one input group and one output group. A Java transformation always has one input group and one output group. You can change the existing group names by typing the group header. If you delete a group, you can add a new group by clicking the Create Input Group or Create Output Group icon. The transformation is not valid if it has multiple input or output groups.

To create a port, click the Add button. When you create a port, the Designer adds it below the currently selected row or group.

For guidelines about creating and editing groups and ports, see “Working with Groups and Ports” on page 59.

Setting Default Port Values

You can define default values for ports in a Java transformation. The Java transformation initializes port variables with the default port value, depending on the datatype of the port. For more information about port datatypes, see “Datatype Mapping” on page 215.

Input and Output Ports

The Java transformation initializes the value of unconnected input ports or output ports that are not assigned a value in the Java code snippets. The Java transformation initializes the ports depending on the port datatype:

- ◆ **Simple datatypes.** If you define a default value for the port, the transformation initializes the value of the port variable to the default value. Otherwise, it initializes the value of the port variable to 0.
- ◆ **Complex datatypes.** If you provide a default value for the port, the transformation creates a new String or byte[] object, and initializes the object to the default value. Otherwise, the transformation initializes the port variable to NULL.

Input ports with a NULL value generate a NullPointerException if you access the value of the port variable in the Java code.

Input/Output Ports

The Java transformation treats input/output ports as pass-through ports. If you do not set a value for the port in the Java code for the transformation, the output value is the same as the input value. The Java transformation initializes the value of an input/output port in the same way as an input port.

If you set the value of a port variable for an input/output port in the Java code, the Java transformation uses this value when it generates an output row. If you do not set the value of an input/output port, the Java transformation sets the value of the port variable to 0 for simple datatypes and NULL for complex datatypes when it generates an output row.

Configuring Java Transformation Properties

The Java transformation includes properties for both the transformation code and the transformation. If you create a Java transformation in the Transformation Developer, you can override the transformation properties when you use it in a mapping.

Figure 9-3 shows the Java transformation Properties tab:

Figure 9-3. Java Transformation Properties

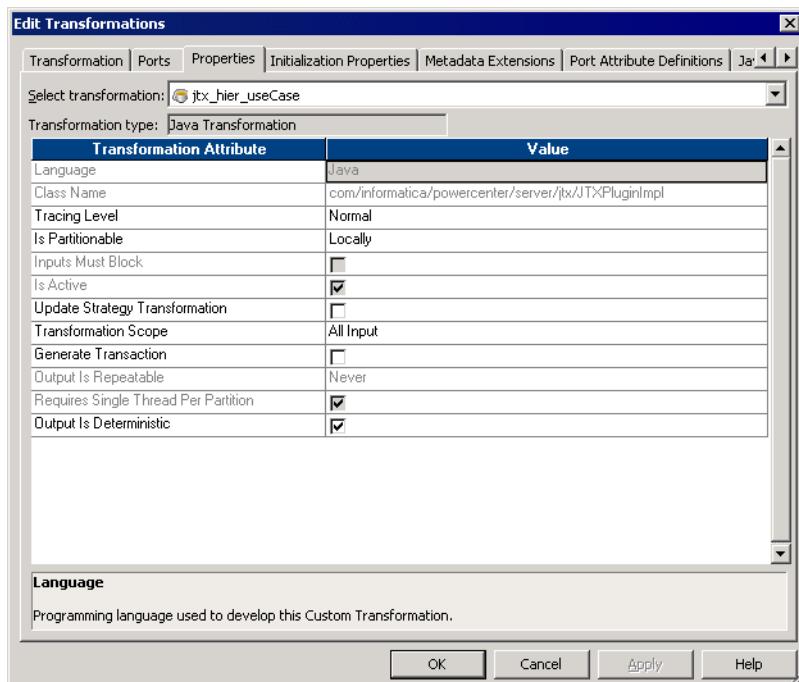


Table 9-2 describes the Java transformation properties:

Table 9-2. Java Transformation Properties

Property	Required / Optional	Description
Language	Required	Language used for the transformation code. You cannot change this value.
Class Name	Required	Name of the Java class for the transformation. You cannot change this value.

Table 9-2. Java Transformation Properties

Property	Required / Optional	Description
Tracing Level	Required	<p>Amount of detail displayed in the session log for this transformation. Use the following tracing levels:</p> <ul style="list-style-type: none">- Terse- Normal- Verbose Initialization- Verbose Data <p>Default is Normal. For more information about tracing levels, see "Session and Workflow Logs" in the <i>Workflow Administration Guide</i>.</p>
Is Partitionable	Required	<p>Multiple partitions in a pipeline can use this transformation. Use the following options:</p> <ul style="list-style-type: none">- No. The transformation cannot be partitioned. The transformation and other transformations in the same pipeline are limited to one partition. You might choose No if the transformation processes all the input data together, such as data cleansing.- Locally. The transformation can be partitioned, but the Integration Service must run all partitions in the pipeline on the same node. Choose Locally when different partitions of the transformation must share objects in memory.- Across Grid. The transformation can be partitioned, and the Integration Service can distribute each partition to different nodes. <p>Default is No.</p> <p>For more information about using partitioning with Java and Custom transformations, see "Working with Partition Points" in the <i>Workflow Administration Guide</i>.</p>
Inputs Must Block	Optional	<p>The procedure associated with the transformation must be able to block incoming data. Default is enabled.</p>
Is Active	Required	<p>The transformation can generate more than one output row for each input row.</p> <p>You cannot change this property after you create the Java transformation. If you need to change this property, create a new Java transformation.</p>
Update Strategy Transformation	Optional	<p>The transformation defines the update strategy for output rows. You can enable this property for active Java transformations.</p> <p>Default is disabled.</p> <p>For more information about setting the update strategy in Java transformations, see "Setting the Update Strategy" on page 224.</p>
Transformation Scope	Required	<p>The method in which the Integration Service applies the transformation logic to incoming data. Use the following options:</p> <ul style="list-style-type: none">- Row- Transaction- All Input <p>This property is always Row for passive transformations. Default is All Input for active transformations.</p> <p>For more information about working with transaction control, see "Working with Transaction Control" on page 223.</p>

Table 9-2. Java Transformation Properties

Property	Required / Optional	Description
Generate Transaction	Optional	The transformation generates transaction rows. You can enable this property for active Java transformations. Default is disabled. For more information about working with transaction control, see "Working with Transaction Control" on page 223.
Output Is Ordered	Required	The order of the output data is consistent between session runs. - Never. The order of the output data is inconsistent between session runs. - Based On Input Order. The output order is consistent between session runs when the input data order is consistent between session runs. - Always. The order of the output data is consistent between session runs even if the order of the input data is inconsistent between session runs. Default is Never for active transformations. Default is Based On Input Order for passive transformations.
Requires Single Thread Per Partition	Optional	A single thread processes the data for each partition. You cannot change this value.
Output Is Deterministic	Optional	The transformation generates consistent output data between session runs. You must enable this property to perform recovery on sessions that use this transformation. For more information about session recovery, see "Recovering Workflows" in the <i>Workflow Administration Guide</i> .

Working with Transaction Control

You can define transaction control for a Java transformation using the following properties:

- ◆ **Transformation Scope.** Determines how the Integration Service applies the transformation logic to incoming data.
- ◆ **Generate Transaction.** Indicates that the Java code for the transformation generates transaction rows and outputs them to the output group.

Transformation Scope

You can configure how the Integration Service applies the transformation logic to incoming data. You can choose one of the following values:

- ◆ **Row.** Applies the transformation logic to one row of data at a time. Choose Row when the results of the transformation depend on a single row of data. You must choose Row for passive transformations.
- ◆ **Transaction.** Applies the transformation logic to all rows in a transaction. Choose Transaction when the results of the transformation depend on all rows in the same transaction, but not on rows in other transactions. For example, you might choose Transaction when the Java code performs aggregate calculations on the data in a single transaction.
- ◆ **All Input.** Applies the transformation logic to all incoming data. When you choose All Input, the Integration Service drops transaction boundaries. Choose All Input when the

results of the transformation depend on all rows of data in the source. For example, you might choose All Input when the Java code for the transformation sorts all incoming data. For more information about transformation scope, see “Understanding Commit Points” in the *Workflow Administration Guide*.

Generate Transaction

You can define Java code in an active Java transformation to generate transaction rows, such as commit and rollback rows. If the transformation generates commit and rollback rows, configure the Java transformation to generate transactions with the Generate Transaction transformation property. For more information about Java transformation API methods to generate transaction rows, see “commit” on page 239 and “rollBack” on page 247.

When you configure the transformation to generate transaction rows, the Integration Service treats the Java transformation like a Transaction Control transformation. Most rules that apply to a Transaction Control transformation in a mapping also apply to the Java transformation. For example, when you configure a Java transformation to generate transaction rows, you cannot concatenate pipelines or pipeline branches containing the transformation. For more information about working with Transaction Control transformations, see “Transaction Control Transformation” on page 519.

When you edit or create a session using a Java transformation configured to generate transaction rows, configure it for user-defined commit.

Setting the Update Strategy

Use an active Java transformation to set the update strategy for a mapping. You can set the update strategy at the following levels:

- ◆ **Within the Java code.** You can write the Java code to set the update strategy for output rows. The Java code can flag rows for insert, update, delete, or reject. For more about setting the update strategy, see “setOutRowType” on page 249.
- ◆ **Within the mapping.** Use the Java transformation in a mapping to flag rows for insert, update, delete, or reject. Select the Update Strategy Transformation property for the Java transformation.
- ◆ **Within the session.** Configure the session to treat the source rows as data driven.

If you do not configure the Java transformation to define the update strategy, or you do not configure the session as data driven, the Integration Service does not use the Java code to flag the output rows. Instead, the Integration Service flags the output rows as insert.

Developing Java Code

Use the code entry tabs to enter Java code snippets that define Java transformation functionality. You can write Java code using the code entry tabs to import Java packages, write helper code, define Java expressions, and write Java code that defines transformation behavior for specific transformation events. You can develop snippets in the code entry tabs in any order.

You can enter Java code in the following code entry tabs:

- ◆ **Import Packages.** Import third-party Java packages, built-in Java packages, or custom Java packages. For more information, see “Importing Java Packages” on page 226.
- ◆ **Helper Code.** Define variables and methods available to all tabs except Import Packages. For more information, see “Defining Helper Code” on page 226.
- ◆ **On Input Row.** Define transformation behavior when it receives an input row. For more information, see “On Input Row Tab” on page 227.
- ◆ **On End of Data.** Define transformation behavior when it has processed all input data. For more information, see “On End of Data Tab” on page 228.
- ◆ **On Receiving Transaction.** Define transformation behavior when it receives a transaction notification. Use with active Java transformations. For more information, see “On Receiving Transaction Tab” on page 228.
- ◆ **Java Expressions.** Define Java expressions to call PowerCenter expressions. You can use Java expressions in the Helper Code, On Input Row, On End of Data, and On Transaction code entry tabs. For more information about Java expressions, see “Java Expressions” on page 263.

You can access input data and set output data on the On Input Row tab. For active transformations, you can also set output data on the On End of Data and On Receiving Transaction tabs.

Creating Java Code Snippets

Use the Code window in the Java Code tab to create Java code snippets to define transformation behavior.

To create a Java code snippet:

1. Click the appropriate code entry tab.
2. To access input or output column variables in the snippet, double-click the name of the port in the Navigator.
3. To call a Java transformation API in the snippet, double-click the name of the API in the navigator. If necessary, configure the appropriate API input values.
4. Write appropriate Java code, depending on the code snippet.

The Full Code windows displays the full class code for the Java transformation.

Importing Java Packages

Use the Import Package tab to import third-party Java packages, built-in Java packages, or custom Java packages for active or passive Java transformations. After you import Java packages, use the imported packages in any code entry tab. You cannot declare or use static variables, instance variables, or user methods in this tab.

For example, to import the Java I/O package, enter the following code in the Import Packages tab:

```
import java.io.*;
```

When you import non-standard Java packages, you must add the package or class to the classpath. For more information about setting the classpath, see “Configuring Java Transformation Settings” on page 229.

When you export or import metadata that contains a Java transformation in the PowerCenter Client, the JAR files or classes that contain the third-party or custom packages required by the Java transformation are not included. If you import metadata that contains a Java transformation, you must also copy the JAR files or classes that contain the required third-party or custom packages to the PowerCenter Client machine.

Defining Helper Code

Use the Helper Code tab in active or passive Java transformations to declare user-defined variables and methods for the Java transformation class. Use variables and methods declared in the Helper Code tab in any code entry tab except the Import Packages tab.

You can declare the following user-defined variables and user-defined methods:

- ♦ **Static code and static variables.** You can declare static variables and static code within a static block. All instances of a reusable Java transformation in a mapping and all partitions in a session share static code and variables. Static code executes before any other code in a Java transformation.

For example, the following code declares a static variable to store the error threshold for all instances of a Java transformation in a mapping:

```
static int errorThreshold;
```

You can then use this variable to store the error threshold for the transformation and access it from all instances of the Java transformation in a mapping and from any partition in a session.

Note: You must synchronize static variables in a multiple partition session or in a reusable Java transformation.

- ♦ **Instance variables.** You can declare partition-level instance variables. Multiple instances of a reusable Java transformation in a mapping or multiple partitions in a session do not share instance variables. Declare instance variables with a prefix to avoid conflicts and initialize non-primitive instance variables.

For example, the following code uses a boolean variable to decide whether to generate an output row:

```
// boolean to decide whether to generate an output row  
// based on validity of input  
private boolean generateRow;
```

- ◆ **User-defined methods.** Create user-defined static or instance methods to extend the functionality of the Java transformation. Java methods declared in the Helper Code tab can use or modify output variables or locally declared instance variables. You cannot access input variables from Java methods in the Helper Code tab.

For example, use the following code in the Helper Code tab to declare a function that adds two integers:

```
private int myTXAdd (int num1,int num2)  
{  
    return num1+num2;  
}
```

On Input Row Tab

Use the On Input Row tab to define the behavior of the Java transformation when it receives an input row. The Java code in this tab executes one time for each input row. You can access input row data in the On Input Row tab only.

You can access and use the following input and output port data, variables, and methods from the On Input Row tab:

- ◆ **Input port and output port variables.** You can access input and output port data as a variable by using the name of the port as the name of the variable. For example, if “in_int” is an Integer input port, you can access the data for this port by referring as a variable “in_int” with the Java primitive datatype int. You do not need to declare input and output ports as variables.

Do not assign a value to an input port variable. If you assign a value to an input variable in the On Input Row tab, you cannot get the input data for the corresponding port in the current row.

- ◆ **Instance variables and user-defined methods.** Use any instance or static variable or user-defined method you declared in the Helper Code tab.

For example, an active Java transformation has two input ports, BASE_SALARY and BONUSES, with an integer datatype, and a single output port, TOTAL_COMP, with an integer datatype. You create a user-defined method in the Helper Code tab, myTXAdd, that adds two integers and returns the result. Use the following Java code in the On Input Row tab to assign the total values for the input ports to the output port and generate an output row:

```
TOTAL_COMP = myTXAdd (BASE_SALARY,BONUSES);  
generateRow();
```

When the Java transformation receives an input row, it adds the values of the BASE_SALARY and BONUSES input ports, assigns the value to the TOTAL_COMP output port, and generates an output row.

- ◆ **Java transformation API methods.** You can call API methods provided by the Java transformation. For more information about Java transformation API methods, see “Java Transformation API Reference” on page 237.

On End of Data Tab

Use the On End of Data tab in active or passive Java transformations to define the behavior of the Java transformation when it has processed all input data. If you want to generate output rows in the On End of Data tab, you must set the transformation scope for the transformation to Transaction or All Input. You cannot access or set the value of input port variables in this tab.

You can access and use the following variables and methods from the On End of Data tab:

- ◆ **Output port variables.** Use the names of output ports as variables to access or set output data for active Java transformations.
- ◆ **Instance variables and user-defined methods.** Use any instance variables or user-defined methods you declared in the Helper Code tab.
- ◆ **Java transformation API methods.** You can call API methods provided by the Java transformation. Use the commit and rollBack API methods to generate a transaction. For more information about API methods, see “Java Transformation API Reference” on page 237.

For example, use the following Java code to write information to the session log when the end of data is reached:

```
logInfo("Number of null rows for partition is: " + partCountNullRows);
```

On Receiving Transaction Tab

Use the On Receiving Transaction tab in active Java transformations to define the behavior of an active Java transformation when it receives a transaction notification. The code snippet for the On Receiving Transaction tab is only executed if the Transaction Scope for the transformation is set to Transaction. You cannot access or set the value of input port variables in this tab.

You can access and use the following output data, variables, and methods from the On Receiving Transaction tab:

- ◆ **Output port variables.** Use the names of output ports as variables to access or set output data.
- ◆ **Instance variables and user-defined methods.** Use any instance variables or user-defined methods you declared in the Helper Code tab.
- ◆ **Java transformation API methods.** You can call API methods provided by the Java transformation. Use the commit and rollBack API methods to generate a transaction. For more information about API methods, see “Java Transformation API Reference” on page 237. For example, use the following Java code to generate a transaction after the transformation receives a transaction:

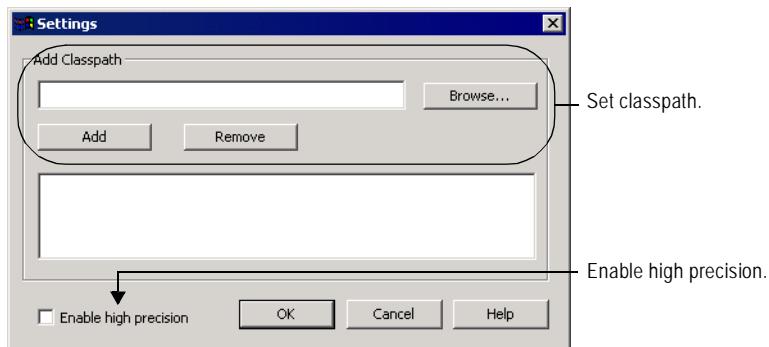
```
commit();
```

Configuring Java Transformation Settings

You can configure Java transformation settings to set the classpath for third-party and custom Java packages and to enable high precision for Decimal datatypes.

Figure 9-4 shows the Settings dialog box for a Java transformation where you can set the classpath and enable high precision:

Figure 9-4. Java Transformation Settings Dialog Box



Configuring the Classpath

When you import non-standard Java packages in the Import package tab, you must set the classpath to the location of the JAR files or class files for the Java package. You can set the CLASSPATH environment variable on the PowerCenter Client machine or configure the Java transformation settings to set the classpath. The PowerCenter Client adds the Java packages or class files you add in the Settings dialog box to the system classpath when you compile the Java code for the transformation.

For example, you import the Java package converter in the Import Packages tab and define the package in converter.jar. You must add converter.jar to the classpath before you compile the Java code for the Java transformation.

You do not need to set the classpath for built-in Java packages. For example, java.io is a built-in Java package. If you import java.io, you do not need to set the classpath for java.io.

Note: You can also add Java packages to the system classpath for a session, using the Java Classpath session property. For more information, see “Session Properties Reference” in the *Workflow Administration Guide*.

To set the classpath for a Java transformation:

1. On the Java Code tab, click the Settings link.
The Settings dialog box appears.
2. Click Browse under Add Classpath to select the JAR file or class file for the imported package. Click OK.

3. Click Add.

The JAR or class file appears in the list of JAR and class files for the transformation.

4. To remove a JAR file or class file, select the JAR or class file and click Remove.

Enabling High Precision

By default, the Java transformation maps ports of type Decimal to double datatypes (with a precision of 15). If you want to process a Decimal datatype with a precision greater than 15, enable high precision to process decimal ports with the Java class BigDecimal.

When you enable high precision, you can process Decimal ports with precision less than 28 as BigDecimal. The Java transformation maps Decimal ports with a precision greater than 28 to double datatypes.

For example, a Java transformation has an input port of type Decimal that receives a value of 40012030304957666903. If you enable high precision, the value of the port is treated as it appears. If you do not enable high precision, the value of the port is 4.00120303049577 x 10¹⁹.

Compiling a Java Transformation

The PowerCenter Client uses the Java compiler to compile the Java code and generate the byte code for the transformation. The Java compiler compiles the Java code and displays the results of the compilation in the Output window. The Java compiler installs with the PowerCenter Client in the java/bin directory.

To compile the full code for the Java transformation, click Compile in the Java Code tab.

When you create a Java transformation, it contains a Java class that defines the base functionality for a Java transformation. The full code for the Java class contains the template class code for the transformation, plus the Java code you define in the code entry tabs.

When you compile a Java transformation, the PowerCenter Client adds the code from the code entry tabs to the template class for the transformation to generate the full class code for the transformation. The PowerCenter Client then calls the Java compiler to compile the full class code. The Java compiler compiles the transformation and generates the byte code for the transformation.

The results of the compilation display in the Output window. Use the results of the compilation to identify and locate Java code errors.

Note: The Java transformation is also compiled when you click OK in the transformation.

Fixing Compilation Errors

You can identify Java code errors and locate the source of Java code errors for a Java transformation in the Output window. Java transformation errors may occur as a result of an error in a code entry tab or may occur as a result of an error in the full code for the Java transformation class.

To troubleshoot a Java transformation:

- ◆ **Locate the source of the error.** You can locate the source of the error in the Java snippet code or in the full class code for the transformation.
- ◆ **Identify the type of error.** Use the results of the compilation in the output window and the location of the error to identify the type of error.

After you identify the source and type of error, fix the Java code in the code entry tab and compile the transformation again.

Locating the Source of Compilation Errors

When you compile a Java transformation, the Output window displays the results of the compilation. Use the results of the compilation to identify compilation errors. When you use the Output window to locate the source of an error, the PowerCenter Client highlights the source of the error in a code entry tab or in the Full Code window.

You can locate errors in the Full Code window, but you cannot edit Java code in the Full Code window. To fix errors that you locate in the Full Code window, you need to modify the code in the appropriate code entry tab. You might need to use the Full Code window to view errors caused by adding user code to the full class code for the transformation.

Use the results of the compilation in the Output window to identify errors in the following locations:

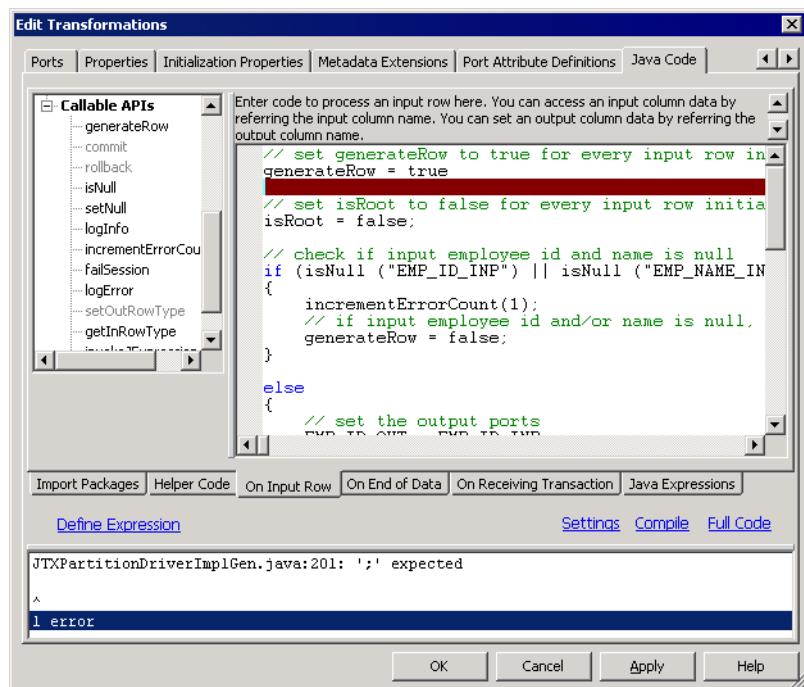
- ◆ Code entry tabs
- ◆ Full Code window

Locating Errors in the Code Entry Tabs

To locate the source of an error in the code entry tabs, right-click on the error in the Output window and choose View error in snippet or double-click on the error in the Output window. The PowerCenter Client highlights the source of the error in the appropriate code entry tab.

Figure 9-5 shows a highlighted error in a code entry tab:

Figure 9-5. Highlighted Error in Code Entry Tab

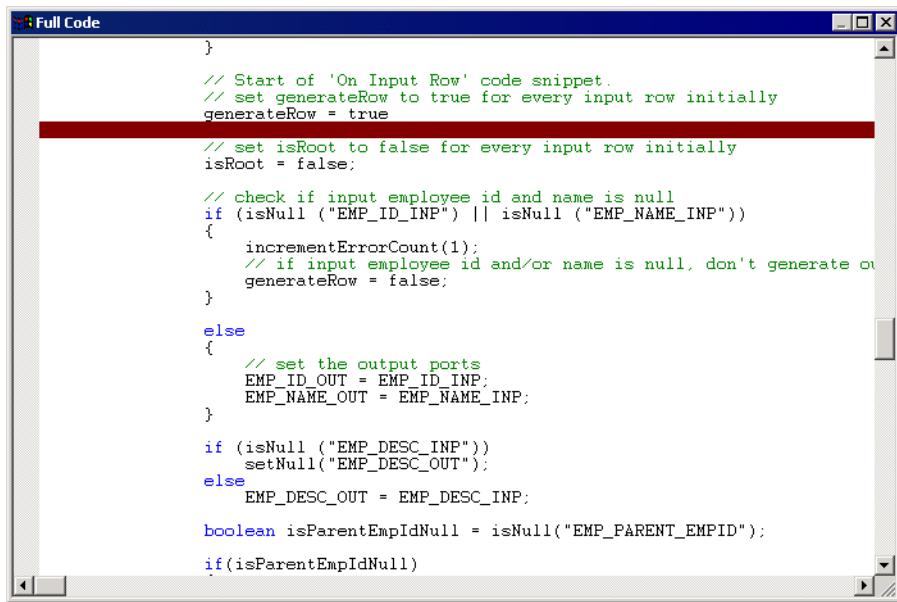


Locating Errors in the Full Code Window

To locate the source of errors in the Full Code window, right-click on the error in the Output window and choose View error in full code or double-click the error in the Output window. The PowerCenter Client highlights the source of the error in the Full Code window.

Figure 9-6 shows a highlighted error in the Full Code window:

Figure 9-6. Highlighted Error in Full Code Window



```
    }
    // Start of 'On Input Row' code snippet.
    // set generateRow to true for every input row initially
    generateRow = true
    // set isRoot to false for every input row initially
    isRoot = false;
    // check if input employee id and name is null
    if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))
    {
        incrementErrorCount(1);
        // if input employee id and/or name is null, don't generate output
        generateRow = false;
    }
    else
    {
        // set the output ports
        EMP_ID_OUT = EMP_ID_INP;
        EMP_NAME_OUT = EMP_NAME_INP;
    }
    if (isNull ("EMP_DESC_INP"))
        setNull("EMP_DESC_OUT");
    else
        EMP_DESC_OUT = EMP_DESC_INP;
    boolean isParentEmpIdNull = isNull("EMP_PARENT_EMPID");
    if(isParentEmpIdNull)
```

Identifying Compilation Errors

Compilation errors may appear as a result of errors in the user code. Errors in the user code may also generate an error in the non-user code for the class. Compilation errors occur in the following code for the Java transformation:

- ◆ User code
- ◆ Non-user code

User Code Errors

Errors may occur in the user code in the code entry tabs. User code errors may include standard Java syntax and language errors. User code errors may also occur when the PowerCenter Client adds the user code from the code entry tabs to the full class code.

For example, a Java transformation has an input port with a name of int1 and an integer datatype. The full code for the class declares the input port variable with the following code:

```
int int1;
```

However, if you use the same variable name in the On Input Row tab, the Java compiler issues an error for a redeclaration of a variable. You must rename the variable in the On Input Row code entry tab to fix the error.

Non-user Code Errors

User code in the code entry tabs may cause errors in non-user code.

For example, a Java transformation has an input port and an output port, int1 and out1, with integer datatypes. You write the following code in the On Input Row code entry tab to calculate interest for input port int1 and assign it to the output port out1:

```
int interest;  
interest = CallInterest(int1); // calculate interest  
out1 = int1 + interest;  
}
```

When you compile the transformation, the PowerCenter Client adds the code from the On Input Row code entry tab to the full class code for the transformation. When the Java compiler compiles the Java code, the unmatched brace causes a method in the full class code to terminate prematurely, and the Java compiler issues an error.

Chapter 10

Java Transformation API Reference

This chapter includes the following topic:

- ◆ Java Transformation API Methods, 238

Java Transformation API Methods

You can call Java transformation API methods in the Java Code tab of a Java transformation to define transformation behavior.

The Java transformation provides the following API methods:

- ◆ **commit.** Generates a transaction. For more information, see “commit” on page 239.
- ◆ **failSession.** Throws an exception with an error message and fails the session. For more information, see “failSession” on page 240.
- ◆ **generateRow.** Generates an output row for active Java transformations. For more information, see “generateRow” on page 241.
- ◆ **getInRowType.** Returns the input type of the current row in the transformation. For more information, see “getInRowType” on page 242.
- ◆ **incrementErrorCount.** Increases the error count for the session. For more information, see “incrementErrorCount” on page 243.
- ◆ **isNull.** Checks the value of an input column for a null value. For more information, see “isNull” on page 244.
- ◆ **logError.** Writes an error message to the session log. For more information, see “logError” on page 246.
- ◆ **logInfo.** Writes an informational message to the session log. For more information, see “logInfo” on page 245.
- ◆ **rollback.** Generates a rollback transaction. For more information, see “rollBack” on page 247.
- ◆ **setNull.** Sets the value of an output column in an active or passive Java transformation to NULL. For more information, see “setNull” on page 248.
- ◆ **setOutRowType.** Sets the update strategy for output rows. For more information, see “setOutRowType” on page 249.

You can add any API method to a code entry tab by double-clicking the name of the API method in the Navigator, dragging the method from the Navigator into the Java code snippet, or manually typing the API method in the Java code snippet.

You can also use the defineJExpression and invokeJExpression API methods to create and invoke Java expressions. For more information about using the API methods with Java expressions, see “Java Expressions” on page 263.

commit

Generates a transaction.

Use commit in any tab except the Import Packages or Java Expressions code entry tabs. You can only use commit in active transformations configured to generate transactions. If you use commit in an active transformation not configured to generate transactions, the Integration Service throws an error and fails the session.

Syntax

Use the following syntax:

```
commit();
```

Example

Use the following Java code to generate a transaction for every 100 rows processed by a Java transformation and then set the rowsProcessed counter to 0:

```
if (rowsProcessed==100) {  
    commit();  
    rowsProcessed=0;  
}
```

failSession

Throws an exception with an error message and fails the session. Use failSession to terminate the session. Do not use failSession in a try/catch block in a code entry tab.

Use failSession in any tab except the Import Packages or Java Expressions code entry tabs.

Syntax

Use the following syntax:

```
failSession(String errorMessage);
```

Argument	Datatype	Input/ Output	Description
errorMessage	String	Input	Error message string.

Example

Use the following Java code to test the input port input1 for a null value and fail the session if input1 is NULL:

```
if(isNull("input1")) {  
    failSession("Cannot process a null value for port input1.");  
}
```

generateRow

Generates an output row for active Java transformations. When you call generateRow, the Java transformation generates an output row using the current value of the output port variables. If you want to generate multiple rows corresponding to an input row, you can call generateRow more than once for each input row. If you do not use generateRow in an active Java transformation, the transformation does not generate output rows.

Use generateRow in any code entry tab except the Import Packages or Java Expressions code entry tabs. You can use generateRow with active transformations only. If you use generateRow in a passive transformation, the session generates an error.

Syntax

Use the following syntax:

```
generateRow();
```

Example

Use the following Java code to generate one output row, modify the values of the output ports, and generate another output row:

```
// Generate multiple rows.  
if(!isNull("input1") && !isNull("input2"))  
{  
    output1 = input1 + input2;  
    output2 = input1 - input2;  
}  
generateRow();  
  
// Generate another row with modified values.  
output1 = output1 * 2;  
output2 = output2 * 2;  
generateRow();
```

getInRowType

Returns the input type of the current row in the transformation. The method returns a value of insert, update, delete, or reject.

You can only use getInRowType in the On Input Row code entry tab. You can only use the getInRowType method in active transformations configured to set the update strategy. If you use this method in an active transformation not configured to set the update strategy, the session generates an error.

Syntax

Use the following syntax:

```
rowType getInRowType();
```

Argument	Datatype	Input/ Output	Description
rowType	String	Output	Update strategy type. Value can be INSERT, UPDATE, DELETE, or REJECT.

Example

Use the following Java code to propagate the input type of the current row if the row type is UPDATE or INSERT and the value of the input port input1 is less than 100 or set the output type as DELETE if the value of input1 is greater than 100:

```
// Set the value of the output port.  
output1 = input1;  
// Get and set the row type.  
String rowType = getInRowType();  
setOutRowType(rowType);  
// Set row type to DELETE if the output port value is > 100.  
if(input1 > 100)  
    setOutRowType(DELETE);
```

incrementErrorCount

Increases the error count for the session. If the error count reaches the error threshold for the session, the session fails. Use incrementErrorCount in any tab except the Import Packages or Java Expressions code entry tabs.

Syntax

Use the following syntax:

```
incrementErrorCount(int nErrors);
```

Argument	Datatype	Input/ Output	Description
nErrors	Integer	Input	Number of errors to increment the error count for the session.

Example

Use the following Java code to increment the error count if an input port for a transformation has a null value:

```
// Check if input employee id and name is null.  
if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))  
{  
    incrementErrorCount(1);  
    // if input employee id and/or name is null, don't generate a output  
    // row for this input row  
    generateRow = false;  
}
```

isNull

Checks the value of an input column for a null value. Use isNull to check if data of an input column is NULL before using the column as a value. You can use the isNull method in the On Input Row code entry tab only.

Syntax

Use the following syntax:

```
Boolean isNull(String strColName);
```

Argument	Datatype	Input/ Output	Description
strColName	String	Input	Name of an input column.

Example

Use the following Java code to check the value of the SALARY input column before adding it to the instance variable totalSalaries:

```
// if value of SALARY is not null  
  
if (!isNull("SALARY")) {  
  
    // add to totalSalaries  
  
    TOTAL_SALARIES += SALARY;  
  
}
```

or

```
// if value of SALARY is not null  
  
String strColName = "SALARY";  
  
if (!isNull(strColName)) {  
  
    // add to totalSalaries  
  
    TOTAL_SALARIES += SALARY;  
  
}
```

logInfo

Writes an informational message to the session log.

Use logInfo in any tab except the Import Packages or Java Expressions tabs.

Syntax

Use the following syntax:

```
logInfo(String logMessage);
```

Argument	Datatype	Input/ Output	Description
logMessage	String	Input	Information message string.

Example

Use the following Java code to write a message to the message log after the Java transformation processes a message threshold of 1,000 rows:

```
if (numRowsProcessed == messageThreshold) {  
    logInfo("Processed " + messageThreshold + " rows.");  
}
```

The following message appears in the session log:

```
[JTX_1012] [INFO] Processed 1000 rows.
```

logError

Write an error message to the session log.

Use logError in any tab except the Import Packages or Java Expressions code entry tabs.

Syntax

Use the following syntax:

```
logError(String errorMessage);
```

Argument	Datatype	Input/ Output	Description
errorMessage	String	Input	Error message string.

Example

Use the following Java code to log an error of the input port is null:

```
// check BASE_SALARY  
  
if (isNull("BASE_SALARY")) {  
  
    logError("Cannot process a null salary field.");  
  
}
```

The following message appears in the message log:

```
[JTX_1013] [ERROR] Cannot process a null salary field.
```

rollBack

Generates a rollback transaction.

Use rollBack in any tab except the Import Packages or Java Expressions code entry tabs. You can only use rollback in active transformations configured to generate transactions. If you use rollback in an active transformation not configured to generate transactions, the Integration Service generates an error and fails the session.

Syntax

Use the following syntax:

```
rollBack();
```

Example

Use the following code to generate a rollback transaction and fail the session if an input row has an illegal condition or generate a transaction if the number of rows processed is 100:

```
// If row is not legal, rollback and fail session.  
if (!isRowLegal()) {  
    rollback();  
    failSession("Cannot process illegal row.");  
}  
else if (rowsProcessed==100) {  
    commit();  
    rowsProcessed=0;  
}
```

setNull

Sets the value of an output column in an active or passive Java transformation to NULL. Once you set an output column to NULL, you cannot modify the value until you generate an output row.

Use setNull in any tab except the Import Packages or Java Expressions code entry tabs.

Syntax

Use the following syntax:

```
setNull(String strColName);
```

Argument	Datatype	Input/ Output	Description
strColName	String	Input	Name of an output column.

Example

Use the following Java code to check the value of an input column and set the corresponding value of an output column to null:

```
// check value of Q3RESULTS input column
if(isNull("Q3RESULTS")) {
    // set the value of output column to null
    setNull("RESULTS");
}
```

or

```
// check value of Q3RESULTS input column
String strColName = "Q3RESULTS";
if(isNull(strColName)) {
    // set the value of output column to null
    setNull(strColName);
}
```

setOutRowType

Sets the update strategy for output rows. The setOutRowType method can flag rows for insert, update, or delete.

You can only use setOutRowType in the On Input Row code entry tab. You can only use setOutRowType in active transformations configured to set the update strategy. If you use setOutRowType in an active transformation not configured to set the update strategy, the session generates an error and the session fails.

Syntax

Use the following syntax:

```
setOutRowType(String rowType);
```

Argument	Datatype	Input/ Output	Description
rowType	String	Input	Update strategy type. Value can be INSERT, UPDATE, or DELETE.

Example

Use the following Java code to propagate the input type of the current row if the row type is UPDATE or INSERT and the value of the input port input1 is less than 100 or set the output type as DELETE if the value of input1 is greater than 100:

```
// Set the value of the output port.  
output1 = input1;  
// Get and set the row type.  
String rowType = getInRowType();  
setOutRowType(rowType);  
// Set row type to DELETE if the output port value is > 100.  
if(input1 > 100)  
    setOutRowType(DELETE);
```


Chapter 11

Java Transformation Example

This chapter includes the following topics:

- ◆ Overview, 252
- ◆ Step 1. Import the Mapping, 253
- ◆ Step 2. Create Transformation and Configure Ports, 254
- ◆ Step 3. Enter Java Code, 256
- ◆ Step 4. Compile the Java Code, 261
- ◆ Step 5. Create a Session and Workflow, 262

Overview

You can use the Java code in this example to create and compile an active Java transformation. You import a sample mapping and create and compile the Java transformation. You can then create and run a session and workflow that contains the mapping.

The Java transformation processes employee data for a fictional company. It reads input rows from a flat file source and writes output rows to a flat file target. The source file contains employee data, including the employee identification number, name, job title, and the manager identification number.

The transformation finds the manager name for a given employee based on the manager identification number and generates output rows that contain employee data. The output data includes the employee identification number, name, job title, and the name of the employee's manager. If the employee has no manager in the source data, the transformation assumes the employee is at the top of the hierarchy in the company organizational chart.

Note: The transformation logic assumes the employee job titles are arranged in descending order in the source file.

Complete the following steps to import the sample mapping, create and compile a Java transformation, and create a session and workflow that contains the mapping:

1. **Import the sample mapping.** For more information, see “Step 1. Import the Mapping” on page 253.
2. **Create the Java transformation and configure the Java transformation ports.** For more information, see “Step 2. Create Transformation and Configure Ports” on page 254.
3. **Enter the Java code for the transformation in the appropriate code entry tabs.** For more information, see “Step 3. Enter Java Code” on page 256.
4. **Compile the Java code.** For more information, see “Step 4. Compile the Java Code” on page 261.
5. **Create and run a session and workflow.** For more information, see “Step 5. Create a Session and Workflow” on page 262.

For a sample source and target file for the session, see “Sample Data” on page 262.

The PowerCenter Client installation contains a mapping, m_jtx_hier_useCase.xml, and flat file source, hier_input, that you can use with this example.

For more information about creating transformations, mappings, sessions, and workflows, see *Getting Started*.

Step 1. Import the Mapping

Import the metadata for the sample mapping in the Designer. The sample mapping contains the following components:

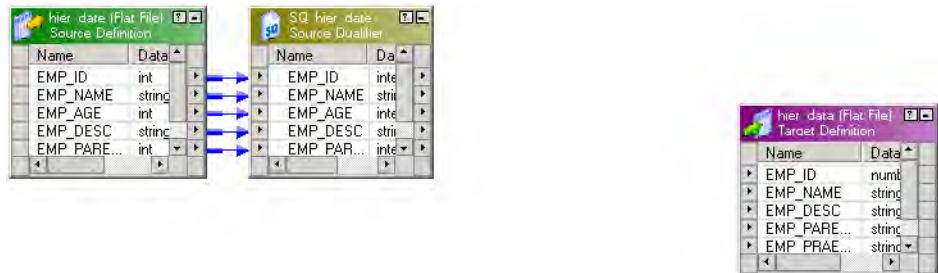
- ◆ **Source definition and Source Qualifier transformation.** Flat file source definition, hier_input, that defines the source data for the transformation.
- ◆ **Target definition.** Flat file target definition, hier_data, that receives the output data from the transformation.

You can import the metadata for the mapping from the following location:

```
<PowerCenter Client installation directory>\client\bin\m_jtx_hier_useCase.xml
```

Figure 11-1 shows the sample mapping:

Figure 11-1. Java Transformation Example - Sample Mapping



Step 2. Create Transformation and Configure Ports

You create the Java transformation and configure the ports in the Mapping Designer. You can use the input and output port names as variables in the Java code. In a Java transformation, you create input and output ports in an input or output group. A Java transformation may contain only one input group and one output group. For more information about configuring ports in a Java transformation, see “Configuring Ports” on page 219.

In the Mapping Designer, create an active Java transformation and configure the ports. In this example, the transformation is named `jtx_hier_useCase`.

Note: To use the Java code in this example, you must use the exact names for the input and output ports.

Table 11-1 shows the input and output ports for the transformation:

Table 11-1. Input and Output Ports

Port Name	Port Type	Datatype	Precision	Scale
EMP_ID_INP	Input	Integer	10	0
EMP_NAME_INP	Input	String	100	0
EMP_AGE	Input	Integer	10	0
EMP_DESC_INP	Input	String	100	0
EMP_PARENT_EMPID	Input	Integer	10	0
EMP_ID_OUT	Output	Integer	10	0
EMP_NAME_OUT	Output	String	100	0
EMP_DESC_OUT	Output	String	100	0
EMP_PARENT_EMPNAME	Output	String	100	0

Figure 11-2 shows the Ports tab in the Transformation Developer after you create the ports:

Figure 11-2. Java Transformation Example - Ports Tab

	Port Name	Datatype	Prec	Scale	I	O
INPUT						
1	EMP_ID_INP	integer	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	EMP_NAME_INP	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	EMP_AGE	integer	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	EMP_DESC_INP	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	EMP_PARENT_EMPID	integer	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
OUTPUT						
6	EMP_ID_OUT	integer	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
7	EMP_NAME_OUT	string	100	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
8	EMP_DESC_OUT	string	100	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
9	EMP_PARENT_EMPNAME	string	100	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Default value: ✓

Description:

OK Cancel Apply Help

Step 3. Enter Java Code

Enter Java code for the transformation in the following code entry tabs:

- ◆ **Import Packages.** Imports the `java.util.Map` and `java.util.HashMap` packages. For more information, see “Import Packages Tab” on page 256.
- ◆ **Helper Code.** Contains a Map object, lock object, and boolean variables used to track the state of data in the Java transformation. For more information, see “Helper Code Tab” on page 257.
- ◆ **On Input Row.** Contains the Java code that processes each input row in the transformation. For more information, see “On Input Row Tab” on page 258.

For more information about using the code entry tabs to develop Java code, see “Developing Java Code” on page 225.

Import Packages Tab

Import third-party Java packages, built-in Java packages, or custom Java packages in the Import Packages tab. The example transformation uses the Map and HashMap packages.

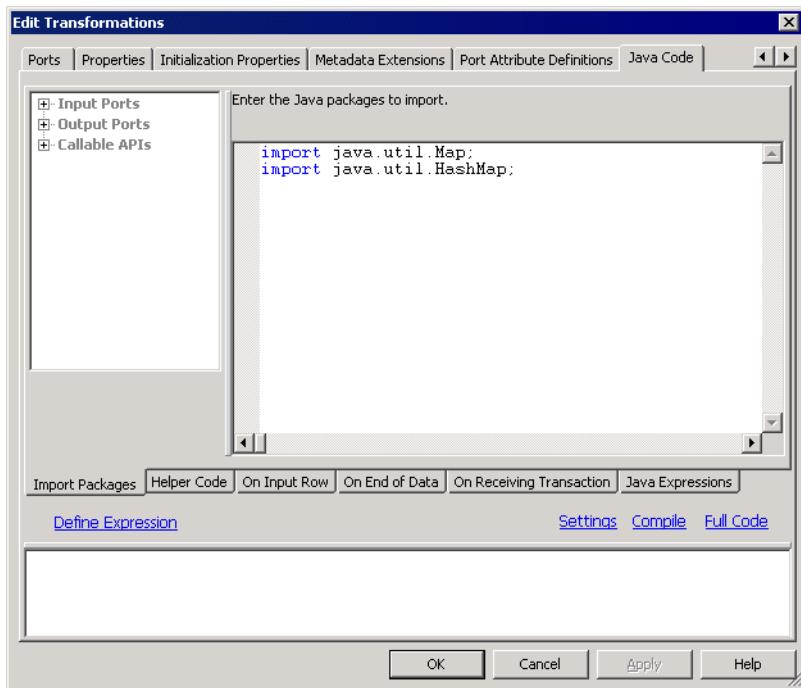
Enter the following code in the Import Packages tab:

```
import java.util.Map;  
import java.util.HashMap;
```

The Designer adds the import statements to the Java code for the transformation.

Figure 11-3 shows the Import Packages code entry tab:

Figure 11-3. Java Transformation Example - Import Packages Tab



Helper Code Tab

Declare user-defined variables and methods for the Java transformation on the Helper Code tab. The Helper Code tab defines the following variables that are used by the Java code in the On Input Row tab:

- ◆ **empMap**. Map object that stores the identification number and employee name from the source.
- ◆ **lock**. Lock object used to synchronize the access to empMap across partitions.
- ◆ **generateRow**. Boolean variable used to determine if an output row should be generated for the current input row.
- ◆ **isRoot**. Boolean variable used to determine if an employee is at the top of the company organizational chart (root).

Enter the following code in the Helper Code tab:

```
// Static Map object to store the ID and name relationship of an
// employee. If a session uses multiple partitions, empMap is shared
// across all partitions.
private static Map empMap = new HashMap();
// Static lock object to synchronize the access to empMap across
// partitions.
private static Object lock = new Object();
```

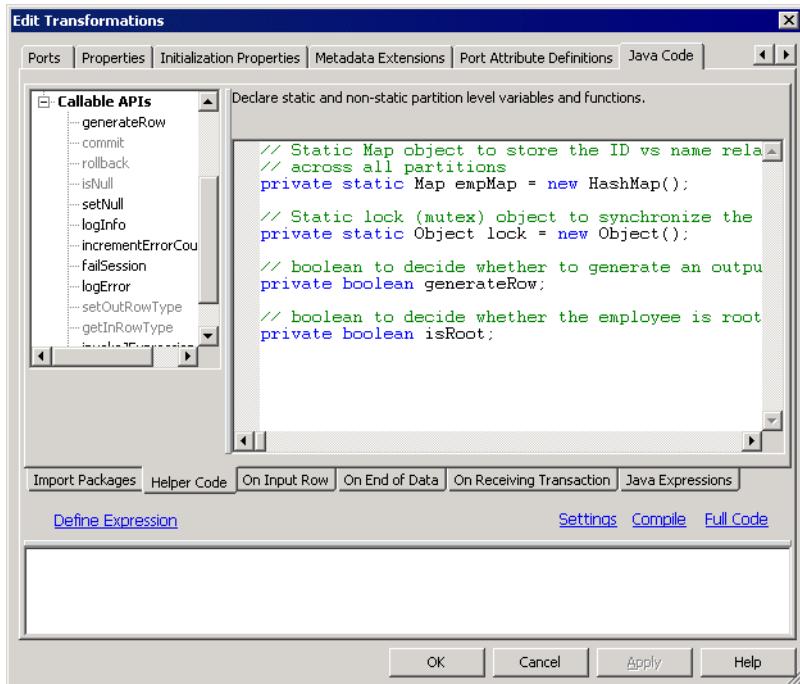
```

// Boolean to track whether to generate an output row based on validity
// of the input data.
private boolean generateRow;
// Boolean to track whether the employee is root.
private boolean isRoot;

```

Figure 11-4 shows the Helper Code tab:

Figure 11-4. Java Transformation Example - Helper Code Tab



On Input Row Tab

The Java transformation executes the Java code in the On Input Row tab when the transformation receives an input row. In this example, the transformation may or may not generate an output row, based on the values of the input row.

Enter the following code in the On Input Row tab:

```

// Initially set generateRow to true for each input row.
generateRow = true;
// Initially set isRoot to false for each input row.
isRoot = false;

// Check if input employee id and name is null.
if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))
{
    incrementErrorCount(1);
    // If input employee id and/or name is null, don't generate a output
}

```

```

        // row for this input row.
        generateRow = false;

    } else {
        // Set the output port values.
        EMP_ID_OUT = EMP_ID_INP;
        EMP_NAME_OUT = EMP_NAME_INP;
    }

    if (isNull ("EMP_DESC_INP"))
        setNull("EMP_DESC_OUT");
    else {
        EMP_DESC_OUT = EMP_DESC_INP;
    }
    boolean isParentEmpIdNull = isNull("EMP_PARENT_EMPID");

    if(isParentEmpIdNull)
    {
        // This employee is the root for the hierarchy.
        isRoot = true;
        logInfo("This is the root for this hierarchy.");
        setNull("EMP_PARENT_EMPNAME");
    }

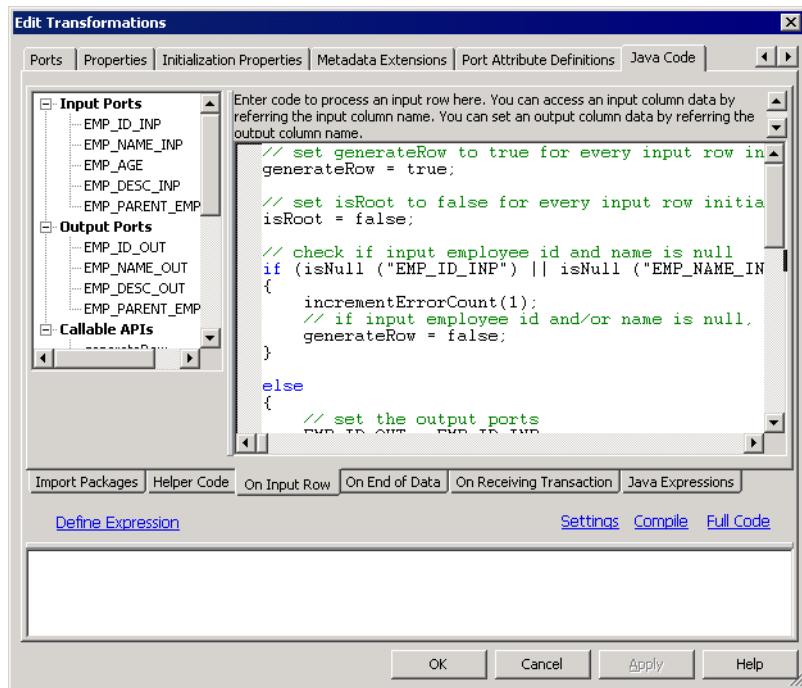
    synchronized(lock)
    {
        // If the employee is not the root for this hierarchy, get the
        // corresponding parent id.
        if(!isParentEmpIdNull)
            EMP_PARENT_EMPNAME = (String) (empMap.get(new Integer
(EMP_PARENT_EMPID)));
        // Add employee to the map for future reference.
        empMap.put (new Integer(EMP_ID_INP), EMP_NAME_INP);
    }

    // Generate row if generateRow is true.
    if(generateRow)
        generateRow();

```

Figure 11-5 shows the On Input Row tab:

Figure 11-5. Java Transformation Example - On Input Row Tab



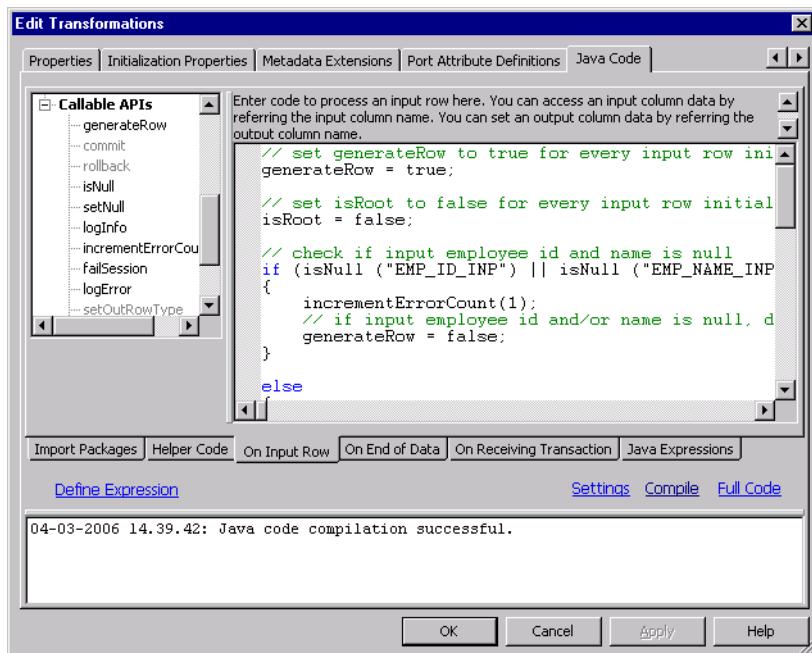
Step 4. Compile the Java Code

Click Compile in the Transformation Developer to compile the Java code for the transformation. The Output window displays the status of the compilation. If the Java code does not compile successfully, correct the errors in the code entry tabs and recompile the Java code. After you successfully compile the transformation, save the transformation to the repository.

For more information about compiling Java code, see “Compiling a Java Transformation” on page 231. For more information about troubleshooting compilation errors, see “Fixing Compilation Errors” on page 232.

Figure 11-6 shows the results of a successful compilation:

Figure 11-6. Java Transformation Example - Successful Compilation



Step 5. Create a Session and Workflow

Create a session and workflow for the mapping in the Workflow Manager, using the m_jtx_hier_useCase mapping.

When you configure the session, you can use the sample source file from the following location:

```
<PowerCenter Client installation directory>\client\bin\hier_data
```

Sample Data

The following data is an excerpt from the sample source file:

```
1,James Davis,50,CEO,  
4,Elaine Masters,40,Vice President - Sales,1  
5,Naresh Thiagarajan,40,Vice President - HR,1  
6,Jeanne Williams,40,Vice President - Software,1  
9,Geetha Manjunath,34,Senior HR Manager,5  
10,Dan Thomas,32,Senior Software Manager,6  
14,Shankar Rahul,34,Senior Software Manager,6  
20,Juan Cardenas,32,Technical Lead,10  
21,Pramodh Rahman,36,Lead Engineer,14  
22,Sandra Patterson,24,Software Engineer,10  
23,Tom Kelly,32,Lead Engineer,10  
35,Betty Johnson,27,Lead Engineer,14  
50,Dave Chu,26,Software Engineer,23  
70,Srihari Giran,23,Software Engineer,35  
71,Frank Smalls,24,Software Engineer,35
```

The following data is an excerpt from a sample target file:

```
1,James Davis,CEO,  
4,Elaine Masters,Vice President - Sales,James Davis  
5,Naresh Thiagarajan,Vice President - HR,James Davis  
6,Jeanne Williams,Vice President - Software,James Davis  
9,Geetha Manjunath,Senior HR Manager,Naresh Thiagarajan  
10,Dan Thomas,Senior Software Manager,Jeanne Williams  
14,Shankar Rahul,Senior Software Manager,Jeanne Williams  
20,Juan Cardenas,Technical Lead,Dan Thomas  
21,Pramodh Rahman,Lead Engineer,Shankar Rahul  
22,Sandra Patterson,Software Engineer,Dan Thomas  
23,Tom Kelly,Lead Engineer,Dan Thomas  
35,Betty Johnson,Lead Engineer,Shankar Rahul  
50,Dave Chu,Software Engineer,Tom Kelly  
70,Srihari Giran,Software Engineer,Betty Johnson  
71,Frank Smalls,Software Engineer,Betty Johnson
```

Chapter 12

Java Expressions

This chapter includes the following topics:

- ◆ Overview, 264
- ◆ Using the Define Expression Dialog Box, 266
- ◆ Working with the Simple Interface, 271
- ◆ Working with the Advanced Interface, 273
- ◆ JExpression API Reference, 279

Overview

You can invoke PowerCenter expressions in a Java transformation with the Java programming language. Use expressions to extend the functionality of a Java transformation. For example, you can invoke an expression in a Java transformation to look up the values of input or output ports or look up the values of Java transformation variables.

To invoke expressions in a Java transformation, you generate the Java code or use Java transformation APIs to invoke the expression. You invoke the expression and use the result of the expression in the appropriate code entry tab. You can generate the Java code that invokes an expression or use API methods to write the Java code that invokes the expression.

Use the following methods to create and invoke expressions in a Java transformation:

- ◆ **Use the Define Expression dialog box.** Create an expression and generate the code for an expression. For more information, see “Using the Define Expression Dialog Box” on page 266.
- ◆ **Use the simple interface.** Use a single method to invoke an expression and get the result of the expression. For more information, see “Working with the Simple Interface” on page 271.
- ◆ **Use the advanced interface.** Use the advanced interface to define the expression, invoke the expression, and use the result of the expression. For more information, see “Working with the Advanced Interface” on page 273.

You can invoke expressions in a Java transformation without advanced knowledge of the Java programming language. You can invoke expressions using the simple interface, which only requires a single method to invoke an expression. If you are familiar with object oriented programming and want more control over invoking the expression, you can use the advanced interface.

Expression Function Types

You can create expressions for a Java transformation using the Expression Editor, by writing the expression in the Define Expression dialog box, or by using the simple or advanced interface. You can enter expressions that use input or output port variables or variables in the Java code as input parameters. If you use the Define Expression dialog box, you can use the Expression Editor to validate the expression before you use it in a Java transformation.

You can invoke the following types of expression functions in a Java transformation:

- ◆ **Transformation language functions.** SQL-like functions designed to handle common expressions.
- ◆ **User-defined functions.** Functions you create in PowerCenter based on transformation language functions.
- ◆ **Custom functions.** Functions you create with the Custom Function API.
- ◆ **Unconnected transformations.** You can use unconnected transformations in expressions. For example, you can use an unconnected lookup transformation in an expression.

You can also use system variables, user-defined mapping and workflow variables, and pre-defined workflow variables such as `$Session.status` in expressions.

For more information about the transformation language and custom functions, see the *Transformation Language Reference*. For more information about user-defined functions, see “Working with User-Defined Functions” in the *Designer Guide*.

Using the Define Expression Dialog Box

When you define a Java expression, you configure the function, create the expression, and generate the code that invokes the expression. You can define the function and create the expression in the Define Expression dialog box.

To create an expression function and use the expression in a Java transformation, complete the following tasks:

1. **Configure the function.** Configure the function that invokes the expression, including the function name, description, and parameters. You use the function parameters when you create the expression. For more information, see “Step 1. Configure the Function” on page 266.
2. **Create the expression.** Create the expression syntax and validate the expression. For more information, see “Step 2. Create and Validate the Expression” on page 267.
3. **Generate Java code.** Use the Define Expression dialog box to generate the Java code that invokes the expression. The Designer places the code in the Java Expressions code entry tab in the Transformation Developer. For more information, see “Step 3. Generate Java Code for the Expression” on page 267.

After you generate the Java code, call the generated function in the appropriate code entry tab to invoke an expression or get a JExpression object, depending on whether you use the simple or advanced interface.

Note: To validate an expression when you create the expression, you must use the Define Expression dialog box.

Step 1. Configure the Function

You configure the function name, description, and input parameters for the Java function that invokes the expression.

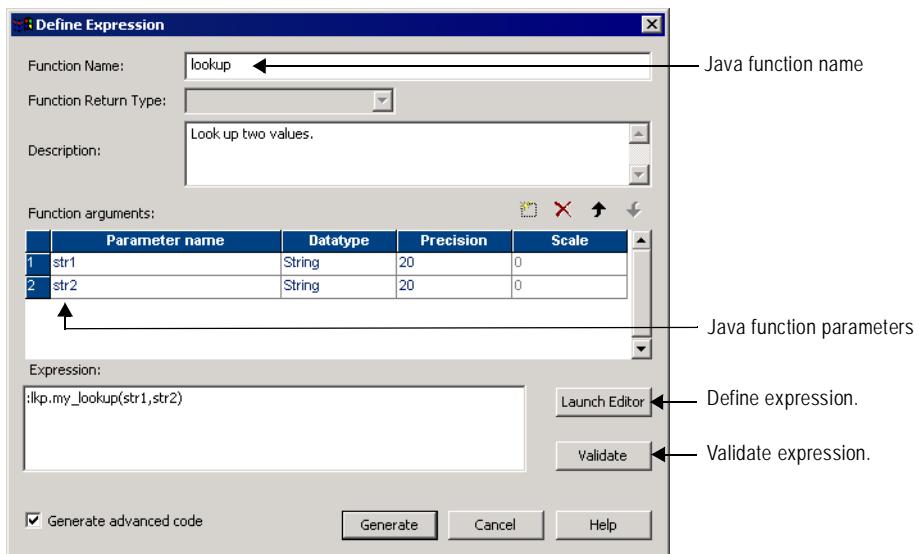
Use the following rules and guidelines when you configure the function:

- ◆ Use a unique function name that does not conflict with an existing Java function in the transformation or reserved Java keywords.
- ◆ You must configure the parameter name, Java datatype, precision, and scale. The input parameters are the values you pass when you call the function in the Java code for the transformation.
- ◆ To pass a Date datatype to an expression, use a String datatype for the input parameter. If an expression returns a Date datatype, you can use the return value as a String datatype in the simple interface and a String or long datatype in the advanced interface.

For more information about the mapping between PowerCenter datatypes and Java datatypes, see “Datatype Mapping” on page 215.

Figure 12-1 shows the Define Expression dialog box where you configure the function and the expression for a Java transformation:

Figure 12-1. Define Expression Dialog Box



Step 2. Create and Validate the Expression

When you create the expression, use the parameters you configured for the function. You can also use transformation language functions, custom functions, or other user-defined functions in the expression. You can create and validate the expression in the Define Expression dialog box or in the Expression Editor.

When you enter expression syntax, follow the transformation language rules and guidelines. For more information about expression syntax, see “The Transformation Language” in the *Transformation Language Reference*.

For more information about creating expressions, see “Working with Workflows” in the *Workflow Administration Guide*.

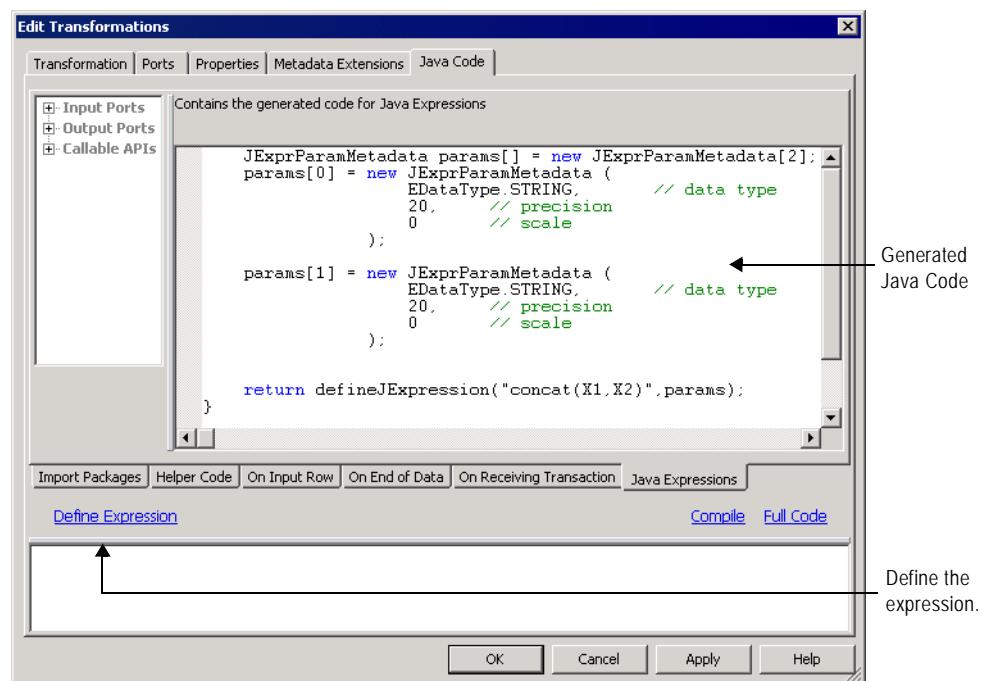
Step 3. Generate Java Code for the Expression

After you configure the function and function parameters and define and validate the expression, you can generate the Java code that invokes the expression. The Designer places the generated Java code in the Java Expressions code entry tab. Use the generated Java code to call the functions that invoke the expression in the code entry tabs in the Transformation Developer. You can generate the simple or advanced Java code.

After you generate the Java code that invokes an expression, you cannot edit the expression and revalidate it. To modify an expression after you generate the code, you must recreate the expression.

Figure 12-2 shows the Java Expressions code entry tab and generated Java code for an expression in the advanced interface:

Figure 12-2. Java Expressions Code Entry Tab



Steps to Create an Expression and Generate Java Code

Complete the following procedure to create an expression and generate the Java code to invoke the expression.

To generate Java code that calls an expression:

1. In the Transformation Developer, open a Java transformation or create a new Java transformation.
2. Click the Java Code tab.
3. Click the Define Expression link.
The Define Expression dialog box appears.
4. Enter a function name.
5. Optionally, enter a description for the expression.
You can enter up to 2,000 characters.
6. Create the parameters for the function.

When you create the parameters, configure the parameter name, datatype, precision, and scale.

7. Click Launch Editor to create an expression with the parameters you created in step 6.
8. Click Validate to validate the expression.
9. Optionally, you can enter the expression in the Expression field and click Validate to validate the expression.
10. If you want to generate Java code using the advanced interface, select Generate advanced code.
11. Click Generate.

The Designer generates the function to invoke the expression in the Java Expressions code entry tab.

Java Expression Templates

You can generate Java code for an expression using the simple or advanced Java code for an expression. The Java code for the expression is generated according to a template for the expression.

Simple Java Code

The following example shows the template for a Java expression generated for simple Java code:

```
Object function_name (Java datatype x1[,
                           Java datatype x2 ...] )
                           throws SDK Exception
{
    return (Object)invokeJExpression( String expression,
                                      new Object [] { x1[, x2, ... ] } );
}
```

The following example shows the template for a Java expression generated using the advanced interface:

```
JExpression function_name () throws SDKException
{
    JExprParamMetadata params[] = new JExprParamMetadata[number of parameters];
    params[0] = new JExprParamMetadata (
        EDataType.STRING, // data type
        20, // precision
        0 // scale
    );
    ...
    params[number of parameters - 1] = new JExprParamMetadata (
        EDataType.STRING, // data type
```

```
    20, // precision
    0   // scale
);
...
return defineJExpression(String expression,params);
}
```

Working with the Simple Interface

Use the invokeJExpression Java API method to invoke an expression in the simple interface.

invokeJExpression

Invokes an expression and returns the value for the expression. Input parameters for invokeJExpression are a string value that represents the expression and an array of objects that contain the expression input parameters.

Use the following rules and guidelines when you use invokeJExpression:

- ◆ **Return datatype.** The return type of invokeJExpression is an object. You must cast the return value of the function with the appropriate datatype. You can return values with Integer, Double, String, and byte[] datatypes.
- ◆ **Row type.** The row type for return values from invokeJExpression is INSERT. If you want to use a different row type for the return value, use the advanced interface. For more information, see “[invoke](#)” on page 279.
- ◆ **Null values.** If you pass a null value as a parameter or the return value for invokeJExpression is NULL, the value is treated as a null indicator. For example, if the return value of an expression is NULL and the return datatype is String, a string is returned with a value of null.
- ◆ **Date datatype.** You must convert input parameters with a Date datatype to String. To use the string in an expression as a Date datatype, use the to_date() function to convert the string to a Date datatype. Also, you must cast the return type of any expression that returns a Date datatype as a String.

Use the following syntax:

```
(datatype) invokeJExpression(  
    String expression,  
    Object[] paramMetadataArray);
```

Argument	Datatype	Input/ Output	Description
expression	String	Input	String that represents the expression.
paramMetadataArray	Object[]	Input	Array of objects that contain the input parameters for the expression.

The following example concatenates the two strings “John” and “Smith” and returns “John Smith”:

```
(String) invokeJExpression("concat(x1,x2)", new Object [] { "John ",  
    "Smith" });
```

Note: The parameters passed to the expression must be numbered consecutively and start with the letter x. For example, to pass three parameters to an expression, name the parameters x1, x2, and x3.

Simple Interface Example

You can define and call expressions that use the invokeJExpression API in the Helper Code or On Input Row code entry tabs. The following example shows how to perform a lookup on the NAME and ADDRESS input ports in a Java transformation and assign the return value to the COMPANY_NAME output port.

Use the following code in the On Input Row code entry tab:

```
COMPANY_NAME = (String)invokeJExpression(":lkp.my_lookup(X1,X2)",  
                                         new Object [] {str1 ,str2} );  
generateRow();
```

Working with the Advanced Interface

You can use the object oriented APIs in the advanced interface to define, invoke, and get the result of an expression.

The advanced interface contains the following classes and Java transformation APIs:

- ◆ **EDataType class.** Enumerates the datatypes for an expression. For more information, see “[EDataType Class](#)” on page 274.
- ◆ **JExprParamMetadata class.** Contains the metadata for each parameter in an expression. Parameter metadata includes datatype, precision, and scale. For more information, see “[JExprParamMetadata Class](#)” on page 274.
- ◆ **defineJExpression API.** Defines the expression. Includes PowerCenter expression string and parameters. For more information, see “[defineJExpression](#)” on page 275.
- ◆ **JExpression class.** Contains the methods to create, invoke, get the metadata and get the expression result, and check the return datatype. For more information, see “[JExpression API Reference](#)” on page 279.

Steps to Invoke an Expression with the Advanced Interface

Complete the following process to define, invoke, and get the result of an expression:

1. In the Helper Code or On Input Row code entry tab, create an instance of JExprParamMetadata for each parameter for the expression and set the value of the metadata. Optionally, you can instantiate the JExprParamMetadata object in defineJExpression.
2. Use defineJExpression to get the JExpression object for the expression.
3. In the appropriate code entry tab, invoke the expression with invoke.
4. Check the result of the return value with isResultNull.
5. You can get the datatype of the return value or the metadata of the return value with getResultDataType and getResultMetadata.
6. Get the result of the expression using the appropriate API. You can use getInt, getDouble, getStringBuffer, and getBytes.

Rules and Guidelines for Working with the Advanced Interface

Use the following rules and guidelines when you work with expressions in the advanced interface:

- ◆ **Null values.** If you pass a null value as a parameter or if the result of an expression is null, the value is treated as a null indicator. For example, if the result of an expression is null and the return datatype is String, a string is returned with a value of null. You can check the result of an expression using isResultNull. For more information, see “[isResultNull](#)” on page 280.

- ◆ **Date datatype.** You must convert input parameters with a Date datatype to a String before you can use them in an expression. To use the string in an expression as a Date datatype, use the `to_date()` function to convert the string to a Date datatype. You can get the result of an expression that returns a Data datatype as String or long datatype. For more information, see “`getStringBuffer`” on page 282 and “`getLong`” on page 281.

EDataType Class

Enumerates the Java datatypes used in expressions. You can use the EDataType class to get the return datatype of an expression or assign the datatype for a parameter in a JExprParamMetadata object. You do not need to instantiate the EDataType class.

Table 12-1 lists the enumerated values for Java datatypes in expressions:

Table 12-1. Enumerated Java Datatypes

Datatype	Enumerated Value
INT	1
DOUBLE	2
STRING	3
BYTE_ARRAY	4
DATE_AS_LONG	5

The following example shows how to use the EDataType class to assign a datatype of String to an JExprParamMetadata object:

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata (
    EDataType.STRING, // data type
    20, // precision
    0 // scale
);
...

```

JExprParamMetadata Class

Instantiates an object that represents the parameters for an expression and sets the metadata for the parameters. You use an array of JExprParamMetadata objects as input to the `defineJExpression` to set the metadata for the input parameters. You can create a instance of the JExprParamMetadata object in the Java Expressions code entry tab or in `defineJExpression`.

Use the following syntax:

```
JExprParamMetadata paramMetadataArray[] =  
    new JExprParamMetadata[numberOfParameters];  
  
paramMetadataArray[0] = new JExprParamMetadata(datatype, precision,  
scale);  
  
...  
  
paramMetadataArray[numberofParameters - 1] =  
    new JExprParamMetadata(datatype, precision, scale);;
```

Argument	Datatype	Input/ Output	Description
datatype	EDataType	Input	Datatype of the parameter.
precision	Integer	Input	Precision of the parameter.
scale	Integer	Input	Scale of the parameter.

For example, use the following Java code to instantiate an array of two JExprParamMetadata objects with String datatypes, precision of 20, and scale of 0:

```
JExprParamMetadata params[] = new JExprParamMetadata[2];  
params[0] = new JExprParamMetadata(EDataType.STRING, 20, 0);  
params[1] = new JExprParamMetadata(EDataType.STRING, 20, 0);  
return defineJExpression(":LKP.LKP_addresslookup(X1,X2)",params);
```

defineJExpression

Defines the expression, including the expression string and input parameters. Arguments for defineJExpression include a JExprParamMetadata object that contains the input parameters and a string value that defines the expression syntax.

To use defineJExpression, you must instantiate an array of JExprParamMetadata objects that represent the input parameters for the expression. You set the metadata values for the parameters and pass the array as an argument to defineJExpression.

Use the following syntax:

```
defineJExpression(  
    String expression,  
    Object[] paramMetadataArray  
) ;
```

Argument	Datatype	Input/ Output	Description
expression	String	Input	String that represents the expression.
paramMetadataArray	Object[]	Input	Array of JExprParaMetadata objects that contain the input parameters for the expression.

For example, use the following Java code to create an expression to perform a lookup on two strings:

```
JExprParaMetadata params[] = new JExprParamMetadata[2];  
params[0] = new JExprParamMetadata(EDataType.STRING, 20, 0);  
params[1] = new JExprParamMetadata(EDataType.STRING, 20, 0);  
defineJExpression(":1kp.mylookup(x1,x2)",params);
```

Note: The parameters passed to the expression must be numbered consecutively and start with the letter x. For example, to pass three parameters to an expression, name the parameters x1, x2, and x3.

JExpression Class

The JExpression class contains the methods to create and invoke an expression, return the value of an expression, and check the return datatype.

Table 12-2 lists the JExpression API methods:

Table 12-2. JExpression API Methods

Method Name	Description
invoke	Invokes an expression.
getResultDataType	Returns the datatype of the expression result.
getResultMetadata	Returns the metadata of the expression result.
isResultNull	Checks the result value of an expression result.
getInt	Returns the value of an expression result as an Integer datatype.
getDouble	Returns the value of an expression result as a Double datatype.
getStringBuffer	Returns the value of an expression result as a String datatype.
getBytes	Returns the value of an expression result as a byte[] datatype.

For more information about the JExpression class, including syntax, usage, and examples, see “JExpression API Reference” on page 279.

Advanced Interface Example

The following example shows how to use the advanced interface to create and invoke a lookup expression in a Java transformation. The Java code shows how to create a function that calls an expression and how to invoke the expression to get the return value. This example passes the values for two input ports with a String datatype, NAME and COMPANY, to the function myLookup. The myLookup function uses a lookup expression to look up the value for the ADDRESS output port.

Note: This example assumes you have an unconnected lookup transformation in the mapping called LKP_addresslookup.

Use the following Java code in the Helper Code tab of the Transformation Developer:

```
JExprParamMetadata addressLookup() throws SDKException
{
    JExprParamMetadata params[] = new JExprParamMetadata[2];
    params[0] = new JExprParamMetadata (
        EDataType.STRING,           // data type
        50,                         // precision
        0                           // scale
    );
    params[1] = new JExprParamMetadata (
        EDataType.STRING,           // data type
        50,                         // precision
        0                           // scale
    );
    return defineJExpression(":LKP.LKP_addresslookup(X1,X2)",params);
}
JExpression lookup = null;
boolean isJExprObjCreated = false;
```

Use the following Java code in the On Input Row tab to invoke the expression and return the value of the ADDRESS port:

```
...
if(!iisJExprObjCreated)
{
    lookup = addressLookup();
    isJExprObjCreated = true;
}
lookup = addressLookup();
lookup.invoke(new Object [] {NAME,COMPANY}, ERowType.INSERT);
EDataType addressDataType = lookup.getResultDataType();
if(addressDataType == EDataType.STRING)
{
    ADDRESS = (lookup.getStringBuffer()).toString();
} else {
    logError("Expression result datatype is incorrect.");
}
...
...
```

JExpression API Reference

The JExpression class contains the following API methods:

- ◆ invoke
- ◆ getResultDataType
- ◆ getResultMetadata
- ◆ isResultNull
- ◆ getInt
- ◆ getDouble
- ◆ getStringBuffer
- ◆ getBytes

invoke

Invokes an expression. Arguments for invoke include an object that defines the input parameters and the row type. You must instantiate an JExpression object before you use invoke.

You can use ERowType.INSERT, ERowType.DELETE, and ERowType.UPDATE for the row type.

Use the following syntax:

```
objectName.invoke(  
    new Object[] { param1[, ... paramN ]},  
    rowType  
);
```

Argument	Datatype	Input/ Output	Description
objectName	JExpression	Input	JExpression object name.
parameters	n/a	Input	Object array that contains the input values for the expression.

For example, you create a function in the Java Expressions code entry tab named address_lookup() that returns an JExpression object that represents the expression. Use the following code to invoke the expression that uses input ports NAME and COMPANY:

```
JExpression myObject = address_lookup();  
myObject.invoke(new Object[] { NAME, COMPANY }, ERowType INSERT);
```

getResultSetType

Returns the datatype of an expression result. getResultDataType returns a value of EDataType. For more information about the EDataType enumerated class, see “EDataType Class” on page 274.

Use the following syntax:

```
objectName.getResultDataType();
```

For example, use the following code to invoke an expression and assign the datatype of the result to the variable dataType:

```
myObject.invoke(new Object[] { NAME, COMPANY }, ERowType INSERT);  
EDataType dataType = myObject.getResultDataType();
```

getResultSetMetadata

Returns the metadata for an expression result. For example, you can use getResultMetadata to get the precision, scale, and datatype of an expression result.

You can assign the metadata of the return value from an expression to an JExprParamMetadata object. Use the getScale, getPrecision, and getDataType object methods to retrieve the result metadata.

Use the following syntax:

```
objectName.getResultMetadata();
```

For example, use the following Java code to assign the scale, precision, and datatype of the return value of myObject to variables:

```
JExprParamMetadata myMetadata = myObject.getResultMetadata();  
  
int scale = myMetadata.getScale();  
  
int prec = myMetadata.getPrecision();  
  
int datatype = myMetadata.getDataType();
```

Note: The getDataType object method returns the integer value of the datatype, as enumerated in EDataType. For more information about the EDataType class, see “EDataType Class” on page 274.

isResultNull

Check the value of an expression result.

Use the following syntax:

```
objectName.isResultNull();
```

For example, use the following Java code to invoke an expression and assign the return value of the expression to the variable address if the return value is not null:

```
JExpression myObject = address_lookup();
myObject.invoke(new Object[] { NAME, COMPANY }, ERowType INSERT);
if(!myObject.isResultNull()) {
    String address = myObject.getStringBuffer();
}
```

getInt

Returns the value of an expression result as an Integer datatype.

Use the following syntax:

```
objectName.getInt();
```

For example, use the following Java code to get the result of an expression that returns an employee ID number as an integer, where findEmpID is a JExpression object:

```
int empID = findEmpID.getInt();
```

getDouble

Returns the value of an expression result as a Double datatype.

Use the following syntax:

```
objectName.getDouble();
```

For example, use the following Java code to get the result of an expression that returns a salary value as a double, where JExprSalary is an JExpression object:

```
double salary = JExprSalary.getDouble();
```

getLong

Returns the value of an expression result as a Long datatype.

You can use getLong to get the result of an expression that uses a Date datatype.

Use the following syntax:

```
objectName.getLong();
```

For example, use the following Java code to get the result of an expression that returns a Date value as a Long datatype, where JExprCurrentDate is an JExpression object:

```
long currDate = JExprCurrentDate.getLong();
```

getStringBuffer

Returns the value of an expression result as a String datatype.

Use the following syntax:

```
objectName.getStringBuffer();
```

For example, use the following Java code to get the result of an expression that returns two concatenated strings, where JExprConcat is an JExpression object:

```
String result = JExprConcat.getStringBuffer();
```

getBytes

Returns the value of an expression result as an byte[] datatype. For example, you can use `getByte` to get the result of an expression that encrypts data with the AES_ENCRYPT function.

Use the following syntax:

```
objectName.getBytes();
```

For example, use the following Java code to get the result of an expression that encrypts the binary data using the AES_ENCRYPT function, where JExprEncryptData is an JExpression object:

```
byte[] newBytes = JExprEncryptData.getBytes();
```

Chapter 13

Joiner Transformation

This chapter includes the following topics:

- ◆ Overview, 284
- ◆ Joiner Transformation Properties, 286
- ◆ Defining a Join Condition, 288
- ◆ Defining the Join Type, 289
- ◆ Using Sorted Input, 292
- ◆ Joining Data from a Single Source, 296
- ◆ Blocking the Source Pipelines, 299
- ◆ Working with Transactions, 300
- ◆ Creating a Joiner Transformation, 303
- ◆ Tips, 306

Overview

Transformation type:

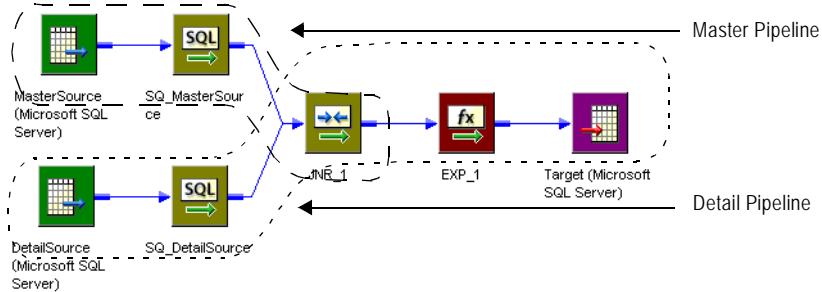
Active
Connected

Use the Joiner transformation to join source data from two related heterogeneous sources residing in different locations or file systems. You can also join data from the same source. The Joiner transformation joins sources with at least one matching column. The Joiner transformation uses a condition that matches one or more pairs of columns between the two sources.

The two input pipelines include a master pipeline and a detail pipeline or a master and a detail branch. The master pipeline ends at the Joiner transformation, while the detail pipeline continues to the target.

Figure 13-1 shows the master and detail pipelines in a mapping with a Joiner transformation:

Figure 13-1. Mapping with Master and Detail Pipelines



To join more than two sources in a mapping, join the output from the Joiner transformation with another source pipeline. Add Joiner transformations to the mapping until you have joined all the source pipelines.

The Joiner transformation accepts input from most transformations. However, consider the following limitations on the pipelines you connect to the Joiner transformation:

- ◆ You cannot use a Joiner transformation when either input pipeline contains an Update Strategy transformation.
- ◆ You cannot use a Joiner transformation if you connect a Sequence Generator transformation directly before the Joiner transformation.

Working with the Joiner Transformation

When you work with the Joiner transformation, you must configure the transformation properties, join type, and join condition. You can configure the Joiner transformation for sorted input to improve Integration Service performance. You can also configure the

transformation scope to control how the Integration Service applies transformation logic. To work with the Joiner transformation, complete the following tasks:

- ◆ **Configure the Joiner transformation properties.** Properties for the Joiner transformation identify the location of the cache directory, how the Integration Service processes the transformation, and how the Integration Service handles caching. For more information, see “Joiner Transformation Properties” on page 286.
- ◆ **Configure the join condition.** The join condition contains ports from both input sources that must match for the Integration Service to join two rows. Depending on the type of join selected, the Integration Service either adds the row to the result set or discards the row. For more information, see “Defining a Join Condition” on page 288.
- ◆ **Configure the join type.** A join is a relational operator that combines data from multiple tables in different databases or flat files into a single result set. You can configure the Joiner transformation to use a Normal, Master Outer, Detail Outer, or Full Outer join type. For more information, see “Defining the Join Type” on page 289.
- ◆ **Configure the session for sorted or unsorted input.** You can improve session performance by configuring the Joiner transformation to use sorted input. To configure a mapping to use sorted data, you establish and maintain a sort order in the mapping so that the Integration Service can use the sorted data when it processes the Joiner transformation. For more information about configuring the Joiner transformation for sorted input, see “Using Sorted Input” on page 292.
- ◆ **Configure the transaction scope.** When the Integration Service processes a Joiner transformation, it can apply transformation logic to all data in a transaction, all incoming data, or one row of data at a time. For more information about configuring how the Integration Service applies transformation logic, see “Working with Transactions” on page 300.

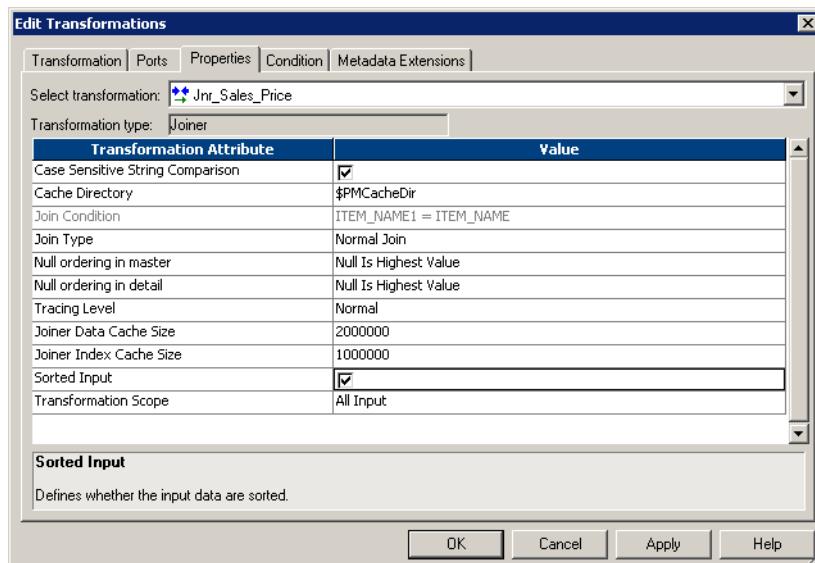
If you have the partitioning option in PowerCenter, you can increase the number of partitions in a pipeline to improve session performance. For information about partitioning with the Joiner transformation, see “Working with Partition Points” in the *Workflow Administration Guide*.

Joiner Transformation Properties

Properties for the Joiner transformation identify the location of the cache directory, how the Integration Service processes the transformation, and how the Integration Service handles caching. The properties also determine how the Integration Service joins tables and files.

Figure 13-2 shows the Joiner transformation properties:

Figure 13-2. Joiner Transformation Properties Tab



When you create a mapping, you specify the properties for each Joiner transformation. When you create a session, you can override some properties, such as the index and data cache size for each transformation.

Table 13-1 describes the Joiner transformation properties:

Table 13-1. Joiner Transformation Properties

Option	Description
Case-Sensitive String Comparison	If selected, the Integration Service uses case-sensitive string comparisons when performing joins on string columns.
Cache Directory	Specifies the directory used to cache master or detail rows and the index to these rows. By default, the cache files are created in a directory specified by the process variable \$PMCacheDir. If you override the directory, make sure the directory exists and contains enough disk space for the cache files. The directory can be a mapped or mounted drive.
Join Type	Specifies the type of join: Normal, Master Outer, Detail Outer, or Full Outer.
Null Ordering in Master	Not applicable for this transformation type.

Table 13-1. Joiner Transformation Properties

Option	Description
Null Ordering in Detail	Not applicable for this transformation type.
Tracing Level	Amount of detail displayed in the session log for this transformation. The options are Terse, Normal, Verbose Data, and Verbose Initialization.
Joiner Data Cache Size	Data cache size for the transformation. Default cache size is 2,000,000 bytes. If the total configured cache size is 2 GB or more, you must run the session on a 64-bit Integration Service. You can configure a numeric value, or you can configure the Integration Service to determine the cache size at runtime. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.
Joiner Index Cache Size	Index cache size for the transformation. Default cache size is 1,000,000 bytes. If the total configured cache size is 2 GB or more, you must run the session on a 64-bit Integration Service. You can configure a numeric value, or you can configure the Integration Service to determine the cache size at runtime. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.
Sorted Input	Specifies that data is sorted. Choose Sorted Input to join sorted data. Using sorted input can improve performance. For more information about working with sorted input, see "Using Sorted Input" on page 292.
Transformation Scope	Specifies how the Integration Service applies the transformation logic to incoming data. You can choose Transaction, All Input, or Row. For more information, see "Working with Transactions" on page 300.

Defining a Join Condition

The join condition contains ports from both input sources that must match for the Integration Service to join two rows. Depending on the type of join selected, the Integration Service either adds the row to the result set or discards the row. The Joiner transformation produces result sets based on the join type, condition, and input data sources.

Before you define a join condition, verify that the master and detail sources are configured for optimal performance. During a session, the Integration Service compares each row of the master source against the detail source. To improve performance for an unsorted Joiner transformation, use the source with fewer rows as the master source. To improve performance for a sorted Joiner transformation, use the source with fewer duplicate key values as the master.

By default, when you add ports to a Joiner transformation, the ports from the first source pipeline display as detail sources. Adding the ports from the second source pipeline sets them as master sources. To change these settings, click the M column on the Ports tab for the ports you want to set as the master source. This sets ports from this source as master ports and ports from the other source as detail ports.

You define one or more conditions based on equality between the specified master and detail sources. For example, if two sources with tables called EMPLOYEE_AGE and EMPLOYEE_POSITION both contain employee ID numbers, the following condition matches rows with employees listed in both sources:

```
EMP_ID1 = EMP_ID2
```

Use one or more ports from the input sources of a Joiner transformation in the join condition. Additional ports increase the time necessary to join two sources. The order of the ports in the condition can impact the performance of the Joiner transformation. If you use multiple ports in the join condition, the Integration Service compares the ports in the order you specify.

The Designer validates datatypes in a condition. Both ports in a condition must have the same datatype. If you need to use two ports in the condition with non-matching datatypes, convert the datatypes so they match.

If you join Char and Varchar datatypes, the Integration Service counts any spaces that pad Char values as part of the string:

```
Char(40) = "abcd"  
Varchar(40) = "abcd"
```

The Char value is “abcd” padded with 36 blank spaces, and the Integration Service does not join the two fields because the Char field contains trailing spaces.

Note: The Joiner transformation does not match null values. For example, if both EMP_ID1 and EMP_ID2 contain a row with a null value, the Integration Service does not consider them a match and does not join the two rows. To join rows with null values, replace null input with default values, and then join on the default values. For more information about default values, see “Using Default Values for Ports” on page 18.

Defining the Join Type

In SQL, a join is a relational operator that combines data from multiple tables into a single result set. The Joiner transformation is similar to an SQL join except that data can originate from different types of sources.

You define the join type on the Properties tab in the transformation. The Joiner transformation supports the following types of joins:

- ◆ Normal
- ◆ Master Outer
- ◆ Detail Outer
- ◆ Full Outer

Note: A normal or master outer join performs faster than a full outer or detail outer join.

If a result set includes fields that do not contain data in either of the sources, the Joiner transformation populates the empty fields with null values. If you know that a field will return a NULL and you do not want to insert NULLs in the target, you can set a default value on the Ports tab for the corresponding port.

Normal Join

With a normal join, the Integration Service discards all rows of data from the master and detail source that do not match, based on the condition.

For example, you might have two sources of data for auto parts called PARTS_SIZE and PARTS_COLOR with the following data:

PARTS_SIZE (master source)

PART_ID1	DESCRIPTION	SIZE
1	Seat Cover	Large
2	Ash Tray	Small
3	Floor Mat	Medium

PARTS_COLOR (detail source)

PART_ID2	DESCRIPTION	COLOR
1	Seat Cover	Blue
3	Floor Mat	Black
4	Fuzzy Dice	Yellow

To join the two tables by matching the PART_IDS in both sources, you set the condition as follows:

```
PART_ID1 = PART_ID2
```

When you join these tables with a normal join, the result set includes the following data:

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black

The following example shows the equivalent SQL statement:

```
SELECT * FROM PARTS_SIZE, PARTS_COLOR WHERE PARTS_SIZE.PART_ID1 =
PARTS_COLOR.PART_ID2
```

Master Outer Join

A master outer join keeps all rows of data from the detail source and the matching rows from the master source. It discards the unmatched rows from the master source.

When you join the sample tables with a master outer join and the same condition, the result set includes the following data:

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black
4	Fuzzy Dice	NULL	Yellow

Because no size is specified for the Fuzzy Dice, the Integration Service populates the field with a NULL.

The following example shows the equivalent SQL statement:

```
SELECT * FROM PARTS_SIZE RIGHT OUTER JOIN PARTS_COLOR ON
(PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

Detail Outer Join

A detail outer join keeps all rows of data from the master source and the matching rows from the detail source. It discards the unmatched rows from the detail source.

When you join the sample tables with a detail outer join and the same condition, the result set includes the following data:

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black

Because no color is specified for the Ash Tray, the Integration Service populates the field with a NULL.

The following example shows the equivalent SQL statement:

```
SELECT * FROM PARTS_SIZE LEFT OUTER JOIN PARTS_COLOR ON  
(PARTS_COLOR.PART_ID2 = PARTS_SIZE.PART_ID1)
```

Full Outer Join

A full outer join keeps all rows of data from both the master and detail sources.

When you join the sample tables with a full outer join and the same condition, the result set includes:

PART_ID	DESCRIPTION	SIZE	Color
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black
4	Fuzzy Dice	NULL	Yellow

Because no color is specified for the Ash Tray and no size is specified for the Fuzzy Dice, the Integration Service populates the fields with NULL.

The following example shows the equivalent SQL statement:

```
SELECT * FROM PARTS_SIZE FULL OUTER JOIN PARTS_COLOR ON  
(PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

Using Sorted Input

You can improve session performance by configuring the Joiner transformation to use sorted input. When you configure the Joiner transformation to use sorted data, the Integration Service improves performance by minimizing disk input and output. You see the greatest performance improvement when you work with large data sets.

To configure a mapping to use sorted data, you establish and maintain a sort order in the mapping so the Integration Service can use the sorted data when it processes the Joiner transformation. Complete the following tasks to configure the mapping:

- ♦ **Configure the sort order.** Configure the sort order of the data you want to join. You can join sorted flat files, or you can sort relational data using a Source Qualifier transformation. You can also use a Sorter transformation.
- ♦ **Add transformations.** Use transformations that maintain the order of the sorted data.
- ♦ **Configure the Joiner transformation.** Configure the Joiner transformation to use sorted data and configure the join condition to use the sort origin ports. The sort origin represents the source of the sorted data.

When you configure the sort order in a session, you can select a sort order associated with the Integration Service code page. When you run the Integration Service in Unicode mode, it uses the selected session sort order to sort character data. When you run the Integration Service in ASCII mode, it sorts all character data using a binary sort order. To ensure that data is sorted as the Integration Service requires, the database sort order must be the same as the user-defined session sort order.

When you join sorted data from partitioned pipelines, you must configure the partitions to maintain the order of sorted data. For more information about joining data from partitioned pipelines, see “Working with Partition Points” in the *Workflow Administration Guide*.

Configuring the Sort Order

You must configure the sort order to ensure that the Integration Service passes sorted data to the Joiner transformation.

Configure the sort order using one of the following methods:

- ♦ **Use sorted flat files.** When the flat files contain sorted data, verify that the order of the sort columns match in each source file.
- ♦ **Use sorted relational data.** Use sorted ports in the Source Qualifier transformation to sort columns from the source database. Configure the order of the sorted ports the same in each Source Qualifier transformation.

For more information about using sorted ports, see “Using Sorted Ports” on page 452.

- ♦ **Use Sorter transformations.** Use a Sorter transformation to sort relational or flat file data. Place a Sorter transformation in the master and detail pipelines. Configure each Sorter transformation to use the same order of the sort key ports and the sort order direction.

For more information about using the Sorter transformation, see “Creating a Sorter Transformation” on page 423.

If you pass unsorted or incorrectly sorted data to a Joiner transformation configured to use sorted data, the session fails and the Integration Service logs the error in the session log file.

Adding Transformations to the Mapping

When you add transformations between the sort origin and the Joiner transformation, use the following guidelines to maintain sorted data:

- ◆ Do not place any of the following transformations between the sort origin and the Joiner transformation:
 - Custom
 - Unsorted Aggregator
 - Normalizer
 - Rank
 - Union transformation
 - XML Parser transformation
 - XML Generator transformation
 - Mapplet, if it contains one of the above transformations
- ◆ You can place a sorted Aggregator transformation between the sort origin and the Joiner transformation if you use the following guidelines:
 - Configure the Aggregator transformation for sorted input using the guidelines in “Using Sorted Input” on page 45.
 - Use the same ports for the group by columns in the Aggregator transformation as the ports at the sort origin.
 - The group by ports must be in the same order as the ports at the sort origin.
- ◆ When you join the result set of a Joiner transformation with another pipeline, verify that the data output from the first Joiner transformation is sorted.

Tip: You can place the Joiner transformation directly after the sort origin to maintain sorted data.

Configuring the Joiner Transformation

To configure the Joiner transformation, complete the following tasks:

- ◆ Enable Sorted Input on the Properties tab.
- ◆ Define the join condition to receive sorted data in the same order as the sort origin.

Defining the Join Condition

Configure the join condition to maintain the sort order established at the sort origin: the sorted flat file, the Source Qualifier transformation, or the Sorter transformation. If you use a sorted Aggregator transformation between the sort origin and the Joiner transformation, treat the sorted Aggregator transformation as the sort origin when you define the join condition. Use the following guidelines when you define join conditions:

- ◆ The ports you use in the join condition must match the ports at the sort origin.
- ◆ When you configure multiple join conditions, the ports in the first join condition must match the first ports at the sort origin.
- ◆ When you configure multiple conditions, the order of the conditions must match the order of the ports at the sort origin, and you must not skip any ports.
- ◆ The number of sorted ports in the sort origin can be greater than or equal to the number of ports at the join condition.

Example of a Join Condition

For example, you configure Sorter transformations in the master and detail pipelines with the following sorted ports:

1. ITEM_NO
2. ITEM_NAME
3. PRICE

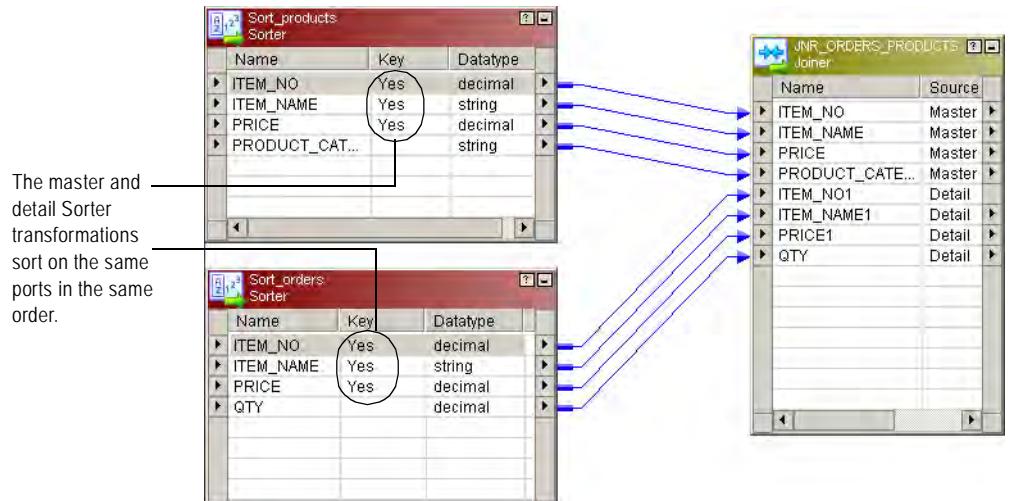
When you configure the join condition, use the following guidelines to maintain sort order:

- ◆ You must use ITEM_NO in the first join condition.
- ◆ If you add a second join condition, you must use ITEM_NAME.
- ◆ If you want to use PRICE in a join condition, you must also use ITEM_NAME in the second join condition.

If you skip ITEM_NAME and join on ITEM_NO and PRICE, you lose the sort order and the Integration Service fails the session.

Figure 13-3 shows a mapping configured to sort and join on the ports ITEM_NO, ITEM_NAME, and PRICE:

Figure 13-3. Mapping Configured to Join Data from Two Pipelines



When you use the Joiner transformation to join the master and detail pipelines, you can configure any one of the following join conditions:

ITEM_NO = ITEM_NO

or

ITEM_NO = ITEM_NO1

ITEM_NAME = ITEM_NAME1

or

ITEM_NO = ITEM_NO1

ITEM_NAME = ITEM_NAME1

PRICE = PRICE1

Joining Data from a Single Source

You may want to join data from the same source if you want to perform a calculation on part of the data and join the transformed data with the original data. When you join the data using this method, you can maintain the original data and transform parts of that data within one mapping. You can join data from the same source in the following ways:

- ◆ Join two branches of the same pipeline.
- ◆ Join two instances of the same source.

Joining Two Branches of the Same Pipeline

When you join data from the same source, you can create two branches of the pipeline. When you branch a pipeline, you must add a transformation between the source qualifier and the Joiner transformation in at least one branch of the pipeline. You must join sorted data and configure the Joiner transformation for sorted input.

For example, you have a source with the following ports:

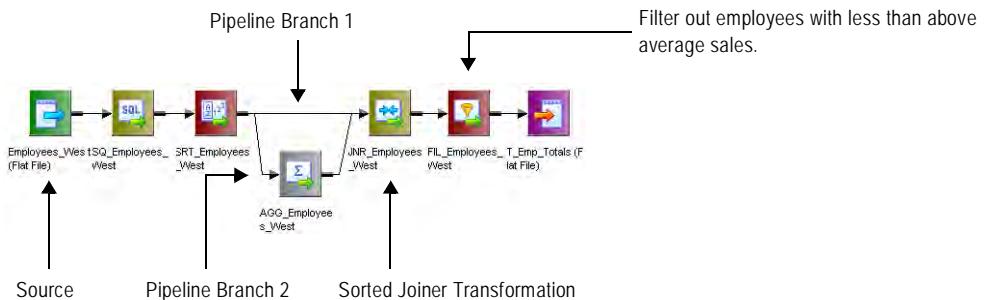
- ◆ Employee
- ◆ Department
- ◆ Total Sales

In the target, you want to view the employees who generated sales that were greater than the average sales for their departments. To do this, you create a mapping with the following transformations:

- ◆ **Sorter transformation.** Sorts the data.
- ◆ **Sorted Aggregator transformation.** Averages the sales data and group by department. When you perform this aggregation, you lose the data for individual employees. To maintain employee data, you must pass a branch of the pipeline to the Aggregator transformation and pass a branch with the same data to the Joiner transformation to maintain the original data. When you join both branches of the pipeline, you join the aggregated data with the original data.
- ◆ **Sorted Joiner transformation.** Uses a sorted Joiner transformation to join the sorted aggregated data with the original data.
- ◆ **Filter transformation.** Compares the average sales data against sales data for each employee and filter out employees with less than above average sales.

Figure 13-4 shows joining two branches of the same pipeline:

Figure 13-4. Mapping that Joins Two Branches of a Pipeline



Note: You can also join data from output groups of the same transformation, such as the Custom transformation or XML Source Qualifier transformation. Place a Sorter transformation between each output group and the Joiner transformation and configure the Joiner transformation to receive sorted input.

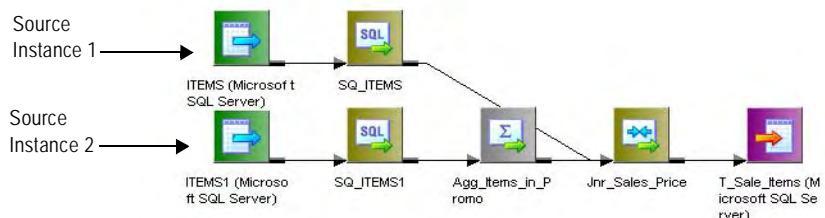
Joining two branches might impact performance if the Joiner transformation receives data from one branch much later than the other branch. The Joiner transformation caches all the data from the first branch, and writes the cache to disk if the cache fills. The Joiner transformation must then read the data from disk when it receives the data from the second branch. This can slow processing.

Joining Two Instances of the Same Source

You can also join same source data by creating a second instance of the source. After you create the second source instance, you can join the pipelines from the two source instances. If you want to join unsorted data, you must create two instances of the same source and join the pipelines.

Figure 13-5 shows two instances of the same source joined using a Joiner transformation:

Figure 13-5. Mapping that Joins Two Instances of the Same Source



Note: When you join data using this method, the Integration Service reads the source data for each source instance, so performance can be slower than joining two branches of a pipeline.

Guidelines

Use the following guidelines when deciding whether to join branches of a pipeline or join two instances of a source:

- ◆ Join two branches of a pipeline when you have a large source or if you can read the source data only once. For example, you can only read source data from a message queue once.
- ◆ Join two branches of a pipeline when you use sorted data. If the source data is unsorted and you use a Sorter transformation to sort the data, branch the pipeline after you sort the data.
- ◆ Join two instances of a source when you need to add a blocking transformation to the pipeline between the source and the Joiner transformation.
- ◆ Join two instances of a source if one pipeline may process slower than the other pipeline.
- ◆ Join two instances of a source if you need to join unsorted data.

Blocking the Source Pipelines

When you run a session with a Joiner transformation, the Integration Service blocks and unblocks the source data, based on the mapping configuration and whether you configure the Joiner transformation for sorted input.

For more information about blocking source data, see “Integration Service Architecture” in the *Administrator Guide*.

Unsorted Joiner Transformation

When the Integration Service processes an unsorted Joiner transformation, it reads all master rows before it reads the detail rows. To ensure it reads all master rows before the detail rows, the Integration Service blocks the detail source while it caches rows from the master source. Once the Integration Service reads and caches all master rows, it unblocks the detail source and reads the detail rows.

Some mappings with unsorted Joiner transformations violate data flow validation. For more information about mappings containing blocking transformations that violate data flow validation, see “Mappings” in the *Designer Guide*.

Sorted Joiner Transformation

When the Integration Service processes a sorted Joiner transformation, it blocks data based on the mapping configuration. Blocking logic is possible if master and detail input to the Joiner transformation originate from different sources.

The Integration Service uses blocking logic to process the Joiner transformation if it can do so without blocking all sources in a target load order group simultaneously. Otherwise, it does not use blocking logic. Instead, it stores more rows in the cache.

When the Integration Service can use blocking logic to process the Joiner transformation, it stores fewer rows in the cache, increasing performance.

Caching Master Rows

When the Integration Service processes a Joiner transformation, it reads rows from both sources concurrently and builds the index and data cache based on the master rows. The Integration Service then performs the join based on the detail source data and the cache data. The number of rows the Integration Service stores in the cache depends on the partitioning scheme, the source data, and whether you configure the Joiner transformation for sorted input. To improve performance for an unsorted Joiner transformation, use the source with fewer rows as the master source. To improve performance for a sorted Joiner transformation, use the source with fewer duplicate key values as the master. For more information about Joiner transformation caches, see “Session Caches” in the *Workflow Administration Guide*.

Working with Transactions

When the Integration Service processes a Joiner transformation, it can apply transformation logic to all data in a transaction, all incoming data, or one row of data at a time. The Integration Service can drop or preserve transaction boundaries depending on the mapping configuration and the transformation scope. You configure how the Integration Service applies transformation logic and handles transaction boundaries using the transformation scope property.

You configure transformation scope values based on the mapping configuration and whether you want to preserve or drop transaction boundaries.

You can preserve transaction boundaries when you join the following sources:

- ◆ **You join two branches of the same source pipeline.** Use the Transaction transformation scope to preserve transaction boundaries. For information about preserving transaction boundaries for a single source, see “Preserving Transaction Boundaries for a Single Pipeline” on page 301.
- ◆ **You join two sources, and you want to preserve transaction boundaries for the detail source.** Use the Row transformation scope to preserve transaction boundaries in the detail pipeline. For more information about preserving transaction boundaries for the detail source, see “Preserving Transaction Boundaries in the Detail Pipeline” on page 301.

You can drop transaction boundaries when you join the following sources:

- ◆ **You join two sources or two branches and you want to drop transaction boundaries.** Use the All Input transformation scope to apply the transformation logic to all incoming data and drop transaction boundaries for both pipelines. For more information about dropping transaction boundaries for two pipelines, see “Dropping Transaction Boundaries for Two Pipelines” on page 302.

Table 13-2 summarizes how to preserve transaction boundaries using transformation scopes with the Joiner transformation:

Table 13-2. Integration Service Behavior with Transformation Scopes for the Joiner Transformation

Transformation Scope	Input Type	Integration Service Behavior
Row	Unsorted	Preserves transaction boundaries in the detail pipeline.
	Sorted	Session fails.
*Transaction	Sorted	Preserves transaction boundaries when master and detail originate from the same transaction generator. Session fails when master and detail do not originate from the same transaction generator
	Unsorted	Session fails.
*All Input	Sorted, Unsorted	Drops transaction boundaries.

**Sessions fail if you use real-time data with All Input or Transaction transformation scopes.*

For more information about transformation scope and transaction boundaries, see “Understanding Commit Points” in the *Workflow Administration Guide*.

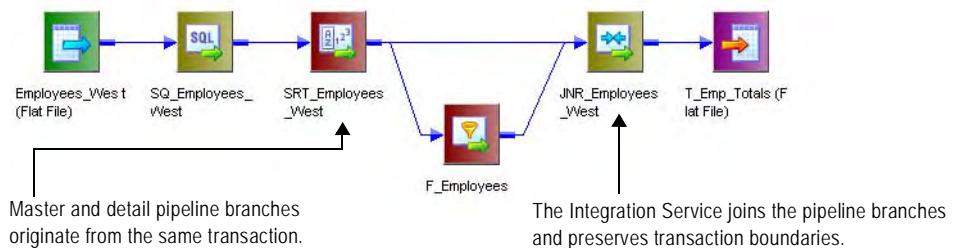
Preserving Transaction Boundaries for a Single Pipeline

When you join data from the same source, use the Transaction transformation scope to preserve incoming transaction boundaries for a single pipeline. Use the Transaction transformation scope when the Joiner transformation joins data from the same source, either two branches of the same pipeline or two output groups of one transaction generator. Use this transformation scope with sorted data and any join type.

When you use the Transaction transformation scope, verify that master and detail pipelines originate from the same transaction control point and that you use sorted input. For example, in Figure 13-6 the Sorter transformation is the transaction control point. You cannot place another transaction control point between the Sorter transformation and the Joiner transformation.

Figure 13-6 shows a mapping configured to join two branches of a pipeline and preserve transaction boundaries:

Figure 13-6. Preserving Transaction Boundaries when You Join Two Pipeline Branches



Preserving Transaction Boundaries in the Detail Pipeline

When you want to preserve the transaction boundaries in the detail pipeline, choose the Row transformation scope. The Row transformation scope allows the Integration Service to process data one row at a time. The Integration Service caches the master data and matches the detail data with the cached master data.

When the source data originates from a real-time source, such as IBM MQ Series, the Integration Service matches the cached master data with each message as it is read from the detail source.

Use the Row transformation scope with Normal and Master Outer join types that use unsorted data.

Dropping Transaction Boundaries for Two Pipelines

When you want to join data from two sources or two branches and you do not need to preserve transaction boundaries, use the All Input transformation scope. When you use All Input, the Integration Service drops incoming transaction boundaries for both pipelines and outputs all rows from the transformation as an open transaction. At the Joiner transformation, the data from the master pipeline can be cached or joined concurrently, depending on how you configure the sort order. Use this transformation scope with sorted and unsorted data and any join type.

For more information about configuring the sort order, see “Joiner Transformation Properties” on page 286.

Creating a Joiner Transformation

To use a Joiner transformation, add a Joiner transformation to the mapping, set up the input sources, and configure the transformation with a condition and join type and sort type.

To create a Joiner Transformation:

1. In the Mapping Designer, click Transformation > Create. Select the Joiner transformation. Enter a name, and click OK.

The naming convention for Joiner transformations is JNR_TransformationName. Enter a description for the transformation.

The Designer creates the Joiner transformation.

2. Drag all the input/output ports from the first source into the Joiner transformation.

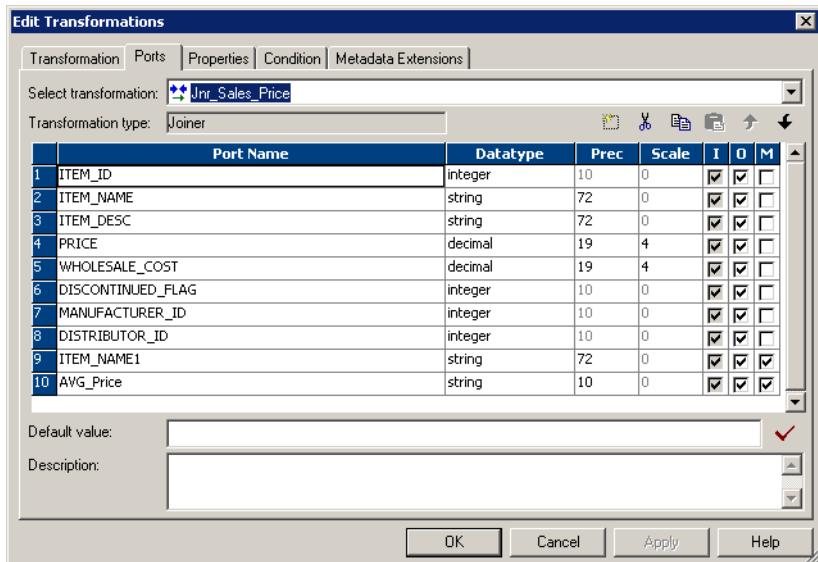
The Designer creates input/output ports for the source fields in the Joiner transformation as detail fields by default. You can edit this property later.

3. Select and drag all the input/output ports from the second source into the Joiner transformation.

The Designer configures the second set of source fields and master fields by default.

4. Double-click the title bar of the Joiner transformation to open the transformation.

5. Click the Ports tab.



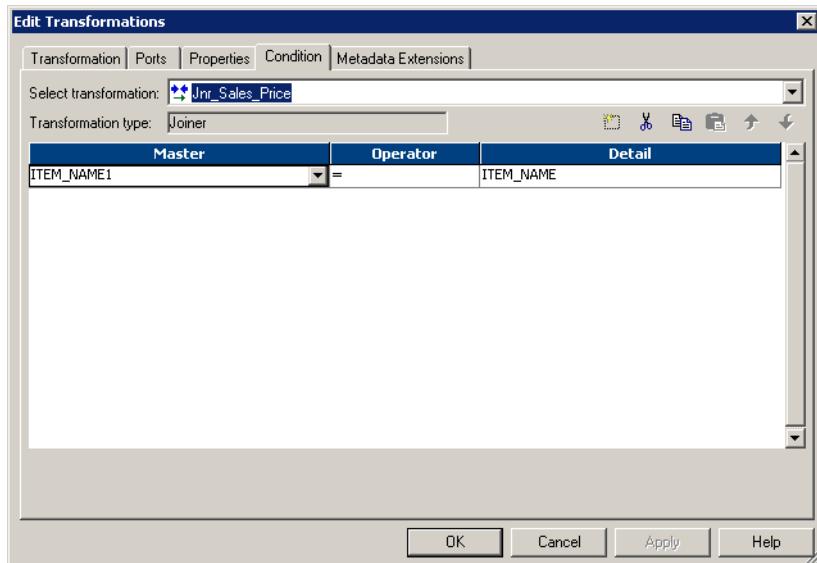
6. Click any box in the M column to switch the master/detail relationship for the sources.

Tip: To improve performance for an unsorted Joiner transformation, use the source with fewer rows as the master source. To improve performance for a sorted Joiner transformation, use the source with fewer duplicate key values as the master.

7. Add default values for specific ports.

Some ports are likely to contain null values, since the fields in one of the sources may be empty. You can specify a default value if the target database does not handle NULLs.

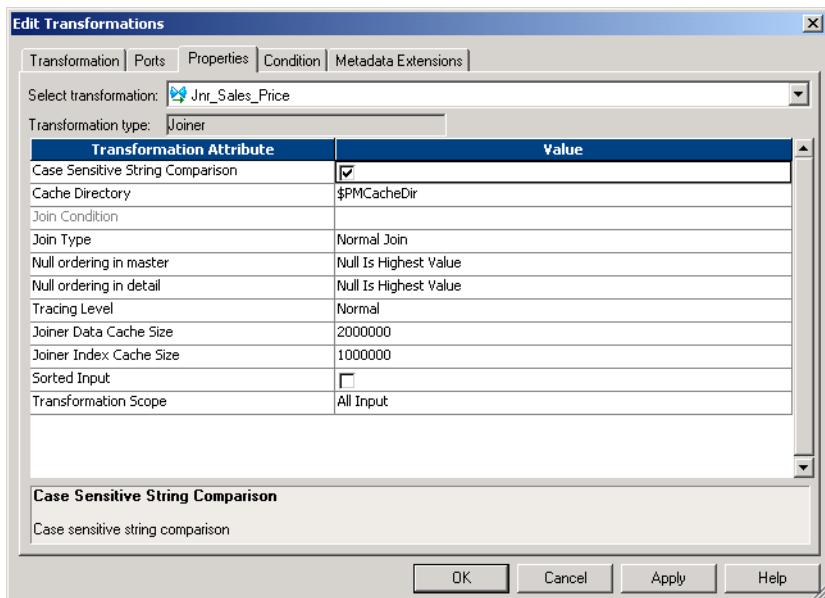
8. Click the Condition tab and set the join condition.



9. Click the Add button to add a condition. You can add multiple conditions. The master and detail ports must have matching datatypes. The Joiner transformation only supports equivalent (=) joins.

For more information about defining the join condition, see “Defining a Join Condition” on page 288.

- 10.** Click the Properties tab and configure properties for the transformation.



Note: You can edit the join condition from the Condition tab. The keyword AND separates multiple conditions.

For more information about defining the properties, see “Joiner Transformation Properties” on page 286.

- 11.** Click OK.

- 12.** Click the Metadata Extensions tab to configure metadata extensions.

For information about working with metadata extensions, see “Metadata Extensions” in the *Repository Guide*.

- 13.** Click Repository > Save to save changes to the mapping.

Tips

The following tips can help improve session performance.

Perform joins in a database when possible.

Performing a join in a database is faster than performing a join in the session. In some cases, this is not possible, such as joining tables from two different databases or flat file systems. If you want to perform a join in a database, use the following options:

- ◆ Create a pre-session stored procedure to join the tables in a database.
- ◆ Use the Source Qualifier transformation to perform the join. For more information, see “Joining Source Data” on page 434 for more information.

Join sorted data when possible.

You can improve session performance by configuring the Joiner transformation to use sorted input. When you configure the Joiner transformation to use sorted data, the Integration Service improves performance by minimizing disk input and output. You see the greatest performance improvement when you work with large data sets. For more information, see “Using Sorted Input” on page 292.

For an unsorted Joiner transformation, designate the source with fewer rows as the master source.

For optimal performance and disk storage, designate the source with the fewer rows as the master source. During a session, the Joiner transformation compares each row of the master source against the detail source. The fewer unique rows in the master, the fewer iterations of the join comparison occur, which speeds the join process.

For a sorted Joiner transformation, designate the source with fewer duplicate key values as the master source.

For optimal performance and disk storage, designate the source with fewer duplicate key values as the master source. When the Integration Service processes a sorted Joiner transformation, it caches rows for one hundred keys at a time. If the master source contains many rows with the same key value, the Integration Service must cache more rows, and performance can be slowed.

Chapter 14

Lookup Transformation

This chapter includes the following topics:

- ◆ Overview, 308
- ◆ Connected and Unconnected Lookups, 309
- ◆ Relational and Flat File Lookups, 311
- ◆ Lookup Components, 313
- ◆ Lookup Properties, 316
- ◆ Lookup Query, 324
- ◆ Lookup Condition, 328
- ◆ Lookup Caches, 330
- ◆ Configuring Unconnected Lookup Transformations, 331
- ◆ Creating a Lookup Transformation, 335
- ◆ Tips, 336

Overview

Transformation type:

Passive

Connected/Unconnected

Use a Lookup transformation in a mapping to look up data in a flat file or a relational table, view, or synonym. You can import a lookup definition from any flat file or relational database to which both the PowerCenter Client and Integration Service can connect. Use multiple Lookup transformations in a mapping.

The Integration Service queries the lookup source based on the lookup ports in the transformation. It compares Lookup transformation port values to lookup source column values based on the lookup condition. Pass the result of the lookup to other transformations and a target.

Use the Lookup transformation to perform many tasks, including:

- ◆ **Get a related value.** For example, the source includes employee ID, but you want to include the employee name in the target table to make the summary data easier to read.
- ◆ **Perform a calculation.** Many normalized tables include values used in a calculation, such as gross sales per invoice or sales tax, but not the calculated value (such as net sales).
- ◆ **Update slowly changing dimension tables.** Use a Lookup transformation to determine whether rows already exist in the target.

You can configure the Lookup transformation to complete the following types of lookups:

- ◆ **Connected or unconnected.** Connected and unconnected transformations receive input and send output in different ways.
- ◆ **Relational or flat file lookup.** When you create a Lookup transformation, you can choose to perform a lookup on a flat file or a relational table.

When you create a Lookup transformation using a relational table as the lookup source, you can connect to the lookup source using ODBC and import the table definition as the structure for the Lookup transformation.

When you create a Lookup transformation using a flat file as a lookup source, the Designer invokes the Flat File Wizard. For more information about using the Flat File Wizard, see “Working with Flat Files” in the *Designer Guide*.

- ◆ **Cached or uncached.** Sometimes you can improve session performance by caching the lookup table. If you cache the lookup, you can choose to use a dynamic or static cache. By default, the lookup cache remains static and does not change during the session. With a dynamic cache, the Integration Service inserts or updates rows in the cache during the session. When you cache the target table as the lookup, you can look up values in the target and insert them if they do not exist, or update them if they do.

Connected and Unconnected Lookups

You can configure a connected Lookup transformation to receive input directly from the mapping pipeline, or you can configure an unconnected Lookup transformation to receive input from the result of an expression in another transformation.

Table 14-1 lists the differences between connected and unconnected lookups:

Table 14-1. Differences Between Connected and Unconnected Lookups

Connected Lookup	Unconnected Lookup
Receives input values directly from the pipeline.	Receives input values from the result of a :LKP expression in another transformation.
Use a dynamic or static cache.	Use a static cache.
Cache includes all lookup columns used in the mapping (that is, lookup source columns included in the lookup condition and lookup source columns linked as output ports to other transformations).	Cache includes all lookup/output ports in the lookup condition and the lookup/return port.
Can return multiple columns from the same row or insert into the dynamic lookup cache.	Designate one return port (R). Returns one column from each row.
If there is no match for the lookup condition, the Integration Service returns the default value for all output ports. If you configure dynamic caching, the Integration Service inserts rows into the cache or leaves it unchanged.	If there is no match for the lookup condition, the Integration Service returns NULL.
If there is a match for the lookup condition, the Integration Service returns the result of the lookup condition for all lookup/output ports. If you configure dynamic caching, the Integration Service either updates the row in the cache or leaves the row unchanged.	If there is a match for the lookup condition, the Integration Service returns the result of the lookup condition into the return port.
Pass multiple output values to another transformation. Link lookup/output ports to another transformation.	Pass one output value to another transformation. The lookup/output/return port passes the value to the transformation calling :LKP expression.
Supports user-defined default values.	Does not support user-defined default values.

Connected Lookup Transformation

The following steps describe how the Integration Service processes a connected Lookup transformation:

1. A connected Lookup transformation receives input values directly from another transformation in the pipeline.
2. For each input row, the Integration Service queries the lookup source or cache based on the lookup ports and the condition in the transformation.
3. If the transformation is uncached or uses a static cache, the Integration Service returns values from the lookup query.

If the transformation uses a dynamic cache, the Integration Service inserts the row into the cache when it does not find the row in the cache. When the Integration Service finds the row in the cache, it updates the row in the cache or leaves it unchanged. It flags the row as insert, update, or no change.

4. The Integration Service passes return values from the query to the next transformation.

If the transformation uses a dynamic cache, you can pass rows to a Filter or Router transformation to filter new rows to the target.

Note: This chapter discusses connected Lookup transformations unless otherwise specified.

Unconnected Lookup Transformation

An unconnected Lookup transformation receives input values from the result of a :LKP expression in another transformation. You can call the Lookup transformation more than once in a mapping.

A common use for unconnected Lookup transformations is to update slowly changing dimension tables. For more information about slowly changing dimension tables, visit the Informatica Knowledge Base at <http://my.informatica.com>.

The following steps describe the way the Integration Service processes an unconnected Lookup transformation:

1. An unconnected Lookup transformation receives input values from the result of a :LKP expression in another transformation, such as an Update Strategy transformation.
2. The Integration Service queries the lookup source or cache based on the lookup ports and condition in the transformation.
3. The Integration Service returns one value into the return port of the Lookup transformation.
4. The Lookup transformation passes the return value into the :LKP expression.

For more information about unconnected Lookup transformations, see “Configuring Unconnected Lookup Transformations” on page 331.

Relational and Flat File Lookups

When you create a Lookup transformation, you can choose to use a relational table or a flat file for the lookup source.

Relational Lookups

When you create a Lookup transformation using a relational table as a lookup source, you can connect to the lookup source using ODBC and import the table definition as the structure for the Lookup transformation.

You can override the default SQL statement to add a WHERE clause or to query multiple tables.

Flat File Lookups

When you use a flat file for a lookup source, use any flat file definition in the repository, or you can import it. When you import a flat file lookup source, the Designer invokes the Flat File Wizard.

Use the following options with flat file lookups only:

- ◆ Use indirect files as lookup sources by specifying a file list as the lookup file name.
- ◆ Use sorted input for the lookup.
- ◆ You can sort null data high or low. With relational lookups, this is based on the database support.
- ◆ Use case-sensitive string comparison with flat file lookups. With relational lookups, the case-sensitive comparison is based on the database support.

Using Sorted Input

When you configure a flat file Lookup transformation for sorted input, the condition columns must be grouped. If the condition columns are not grouped, the Integration Service cannot cache the lookup and fails the session. For best caching performance, sort the condition columns.

For example, a Lookup transformation has the following condition:

```
OrderID = OrderID1
```

```
CustID = CustID1
```

In the following flat file lookup source, the keys are grouped, but not sorted. The Integration Service can cache the data, but performance may not be optimal.

OrderID	CustID	ItemNo.	ItemDesc	Comments
1001	CA502	F895S	Flashlight	<i>Key data is grouped, but not sorted.</i>
	CA501	C530S	Compass	<i>CustID is out of order within OrderID.</i>

OrderID	CustID	ItemNo.	ItemDesc	Comments
1001	CA501	T552T	Tent	
1005	OK503	S104E	Safety Knife	<i>Key data is grouped, but not sorted. OrderID is out of order.</i>
1003	CA500	F304T	First Aid Kit	
1003	TN601	R938M	Regulator System	

The keys are not grouped in the following flat file lookup source. The Integration Service cannot cache the data and fails the session.

OrderID	CustID	ItemNo.	ItemDesc	Comments
1001	CA501	T552T	Tent	
1001	CA501	C530S	Compass	
1005	OK503	S104E	Safety Knife	
1003	TN601	R938M	Regulator System	
1003	CA500	F304T	First Aid Kit	
1001	CA502	F895S	Flashlight	<i>Key data for CustID is not grouped.</i>

If you choose sorted input for indirect files, the range of data must not overlap in the files.

Lookup Components

Define the following components when you configure a Lookup transformation in a mapping:

- ◆ Lookup source
- ◆ Ports
- ◆ Properties
- ◆ Condition
- ◆ Metadata extensions

Lookup Source

Use a flat file or a relational table for a lookup source. When you create a Lookup transformation, you can import the lookup source from the following locations:

- ◆ Any relational source or target definition in the repository
- ◆ Any flat file source or target definition in the repository
- ◆ Any table or file that both the Integration Service and PowerCenter Client machine can connect to

The lookup table can be a single table, or you can join multiple tables in the same database using a lookup SQL override. The Integration Service queries the lookup table or an in-memory cache of the table for all incoming rows into the Lookup transformation.

The Integration Service can connect to a lookup table using a native database driver or an ODBC driver. However, the native database drivers improve session performance.

Indexes and a Lookup Table

If you have privileges to modify the database containing a lookup table, you can improve lookup initialization time by adding an index to the lookup table. This is important for very large lookup tables. Since the Integration Service needs to query, sort, and compare values in these columns, the index needs to include every column used in a lookup condition.

You can improve performance by indexing the following types of lookup:

- ◆ **Cached lookups.** You can improve performance by indexing the columns in the lookup ORDER BY. The session log contains the ORDER BY clause.
- ◆ **Uncached lookups.** Because the Integration Service issues a SELECT statement for each row passing into the Lookup transformation, you can improve performance by indexing the columns in the lookup condition.

Lookup Ports

The Ports tab contains options similar to other transformations, such as port name, datatype, and scale. In addition to input and output ports, the Lookup transformation includes a

lookup port type that represents columns of data in the lookup source. An unconnected Lookup transformation also includes a return port type that represents the return value.

Table 14-2 describes the port types in a Lookup transformation:

Table 14-2. Lookup Transformation Port Types

Ports	Type of Lookup	Number Required	Description
I	Connected Unconnected	Minimum of 1	Input port. Create an input port for each lookup port you want to use in the lookup condition. You must have at least one input or input/output port in each Lookup transformation.
O	Connected Unconnected	Minimum of 1	Output port. Create an output port for each lookup port you want to link to another transformation. You can designate both input and lookup ports as output ports. For connected lookups, you must have at least one output port. For unconnected lookups, use a lookup/output port as a return port (R) to designate a return value.
L	Connected Unconnected	Minimum of 1	Lookup port. The Designer designates each column in the lookup source as a lookup (L) and output port (O).
R	Unconnected	1 only	Return port. Use only in unconnected Lookup transformations. Designates the column of data you want to return based on the lookup condition. You can designate one lookup/output port as the return port.

The Lookup transformation also enables an associated ports property that you configure when you use a dynamic cache.

Use the following guidelines to configure lookup ports:

- ♦ If you delete lookup ports from a flat file session, the session fails.
- ♦ You can delete lookup ports from a relational lookup if you are certain the mapping does not use the lookup port. This reduces the amount of memory the Integration Service uses to run the session.
- ♦ To ensure datatypes match when you add an input port, copy the existing lookup ports.

Lookup Properties

On the Properties tab, you can configure properties, such as an SQL override for relational lookups, the lookup source name, and tracing level for the transformation. You can also configure caching properties on the Properties tab.

For more information about lookup properties, see “Lookup Properties” on page 316.

Lookup Condition

On the Condition tab, you can enter the condition or conditions you want the Integration Service to use to determine whether input data qualifies values in the lookup source or cache.

For more information about the lookup condition, see “Lookup Condition” on page 328.

Metadata Extensions

You can extend the metadata stored in the repository by associating information with repository objects, such as Lookup transformations. For example, when you create a Lookup transformation, you may want to store your name and the creation date with the Lookup transformation. You associate information with repository metadata using metadata extensions. For more information, see “Metadata Extensions” in the *Repository Guide*.

Lookup Properties

Properties for the Lookup transformation identify the database source, how the Integration Service processes the transformation, and how it handles caching and multiple matches.

When you create a mapping, you specify the properties for each Lookup transformation. When you create a session, you can override some properties, such as the index and data cache size, for each transformation in the session properties.

Table 14-3 describes the Lookup transformation properties:

Table 14-3. Lookup Transformation Properties

Option	Lookup Type	Description
Lookup SQL Override	Relational	Overrides the default SQL statement to query the lookup table. Specifies the SQL statement you want the Integration Service to use for querying lookup values. Use only with the lookup cache enabled. For more information, see "Lookup Query" on page 324.
Lookup Table Name	Relational	Specifies the name of the table from which the transformation looks up and caches values. You can import a table, view, or synonym from another database by selecting the Import button on the dialog box that appears when you first create a Lookup transformation. If you enter a lookup SQL override, you do not need to add an entry for this option.
Lookup Caching Enabled	Flat File, Relational	Indicates whether the Integration Service caches lookup values during the session. When you enable lookup caching, the Integration Service queries the lookup source once, caches the values, and looks up values in the cache during the session. This can improve session performance. When you disable caching, each time a row passes into the transformation, the Integration Service issues a select statement to the lookup source for lookup values. Note: The Integration Service always caches flat file lookups.
Lookup Policy on Multiple Match	Flat File, Relational	Determines what happens when the Lookup transformation finds multiple rows that match the lookup condition. You can select the first or last row returned from the cache or lookup source, or report an error. Or, you can allow the Lookup transformation to use any value. When you configure the Lookup transformation to return any matching value, the transformation returns the first value that matches the lookup condition. It creates an index based on the key ports rather than all Lookup transformation ports. If you do not enable the Output Old Value On Update option, the Lookup Policy On Multiple Match option is set to Report Error for dynamic lookups. For more information about lookup caches, see "Lookup Caches" on page 337.
Lookup Condition	Flat File, Relational	Displays the lookup condition you set in the Condition tab.

Table 14-3. Lookup Transformation Properties

Option	Lookup Type	Description
Connection Information	Relational	<p>Specifies the database containing the lookup table. You can select the database connection or use the \$Source or \$Target variable. If you use one of these variables, the lookup table must reside in the source or target database you specify when you configure the session.</p> <p>If you select the database connection, you can also specify what type of database connection it is. Type <code>Application:</code> before the connection name if it is an Application connection. Type <code>Relational:</code> before the connection name if it is a relational connection.</p> <p>If you do not specify the type of database connection, the Integration Service fails the session if it cannot determine the type of database connection.</p> <p>For more information about using \$Source and \$Target, see "Configuring Relational Lookups in a Session" on page 322.</p>
Source Type	Flat File, Relational	Indicates that the Lookup transformation reads values from a relational database or a flat file.
Tracing Level	Flat File, Relational	Sets the amount of detail included in the session log when you run a session containing this transformation.
Lookup Cache Directory Name	Flat File, Relational	<p>Specifies the directory used to build the lookup cache files when you configure the Lookup transformation to cache the lookup source. Also used to save the persistent lookup cache files when you select the Lookup Persistent option.</p> <p>By default, the Integration Service uses the \$PMCacheDir directory configured for the Integration Service.</p>
Lookup Cache Persistent	Flat File, Relational	Indicates whether the Integration Service uses a persistent lookup cache, which consists of at least two cache files. If a Lookup transformation is configured for a persistent lookup cache and persistent lookup cache files do not exist, the Integration Service creates the files during the session. Use only with the lookup cache enabled.
Lookup Data Cache Size	Flat File, Relational	<p>Indicates the maximum size the Integration Service allocates to the data cache in memory. You can configure a numeric value, or you can configure the Integration Service to determine the cache size at runtime. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.</p> <p>If the Integration Service cannot allocate the configured amount of memory when initializing the session, it fails the session. When the Integration Service cannot store all the data cache data in memory, it pages to disk.</p> <p>The Lookup Data Cache Size is 2,000,000 bytes by default. The minimum size is 1,024 bytes. If the total configured session cache size is 2 GB (2,147,483, 648 bytes) or greater, you must run the session on a 64-bit Integration Service.</p> <p>Use only with the lookup cache enabled.</p>

Table 14-3. Lookup Transformation Properties

Option	Lookup Type	Description
Lookup Index Cache Size	Flat File, Relational	<p>Indicates the maximum size the Integration Service allocates to the index cache in memory. You can configure a numeric value, or you can configure the Integration Service to determine the cache size at runtime. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.</p> <p>If the Integration Service cannot allocate the configured amount of memory when initializing the session, it fails the session. When the Integration Service cannot store all the index cache data in memory, it pages to disk.</p> <p>The Lookup Index Cache Size is 1,000,000 bytes by default. The minimum size is 1,024 bytes. If the total configured session cache size is 2 GB (2,147,483, 648 bytes) or greater, you must run the session on a 64-bit Integration Service.</p> <p>Use only with the lookup cache enabled.</p>
Dynamic Lookup Cache	Flat File, Relational	<p>Indicates to use a dynamic lookup cache. Inserts or updates rows in the lookup cache as it passes rows to the target table.</p> <p>Use only with the lookup cache enabled.</p>
Output Old Value On Update	Flat File, Relational	<p>Use only with dynamic caching enabled. When you enable this property, the Integration Service outputs old values out of the lookup/output ports. When the Integration Service updates a row in the cache, it outputs the value that existed in the lookup cache before it updated the row based on the input data. When the Integration Service inserts a new row in the cache, it outputs null values.</p> <p>When you disable this property, the Integration Service outputs the same values out of the lookup/output and input/output ports.</p> <p>This property is enabled by default.</p>
Cache File Name Prefix	Flat File, Relational	<p>Use only with persistent lookup cache. Specifies the file name prefix to use with persistent lookup cache files. The Integration Service uses the file name prefix as the file name for the persistent cache files it saves to disk. Only enter the prefix. Do not enter .idx or .dat.</p> <p>If the named persistent cache files exist, the Integration Service builds the memory cache from the files. If the named persistent cache files do not exist, the Integration Service rebuilds the persistent cache files.</p>
Recache From Lookup Source	Flat File, Relational	<p>Use only with the lookup cache enabled. When selected, the Integration Service rebuilds the lookup cache from the lookup source when it first calls the Lookup transformation instance.</p> <p>If you use a persistent lookup cache, it rebuilds the persistent cache files before using the cache. If you do not use a persistent lookup cache, it rebuilds the lookup cache in memory before using the cache.</p>

Table 14-3. Lookup Transformation Properties

Option	Lookup Type	Description
Insert Else Update	Flat File, Relational	Use only with dynamic caching enabled. Applies to rows entering the Lookup transformation with the row type of <i>insert</i> . When you select this property and the row type entering the Lookup transformation is insert, the Integration Service inserts the row into the cache if it is new, and updates the row if it exists. If you do not select this property, the Integration Service only inserts new rows into the cache when the row type entering the Lookup transformation is insert. For more information about defining the row type, see "Using Update Strategy Transformations with a Dynamic Cache" on page 354.
Update Else Insert	Flat File, Relational	Use only with dynamic caching enabled. Applies to rows entering the Lookup transformation with the row type of <i>update</i> . When you select this property and the row type entering the Lookup transformation is update, the Integration Service updates the row in the cache if it exists, and inserts the row if it is new. If you do not select this property, the Integration Service only updates existing rows in the cache when the row type entering the Lookup transformation is update. For more information about defining the row type, see "Using Update Strategy Transformations with a Dynamic Cache" on page 354.
Datetime Format	Flat File	If you do not define a datetime format for a particular field in the lookup definition or on the Ports tab, the Integration Service uses the properties defined here. You can enter any datetime format. Default is MM/DD/YYYY HH24:MI:SS.
Thousand Separator	Flat File	If you do not define a thousand separator for a particular field in the lookup definition or on the Ports tab, the Integration Service uses the properties defined here. You can choose no separator, a comma, or a period. Default is no separator.
Decimal Separator	Flat File	If you do not define a decimal separator for a particular field in the lookup definition or on the Ports tab, the Integration Service uses the properties defined here. You can choose a comma or a period decimal separator. Default is period.
Case-Sensitive String Comparison	Flat File	If selected, the Integration Service uses case-sensitive string comparisons when performing lookups on string columns. Note: For relational lookups, the case-sensitive comparison is based on the database support.

Table 14-3. Lookup Transformation Properties

Option	Lookup Type	Description
Null Ordering	Flat File	Determines how the Integration Service orders null values. You can choose to sort null values high or low. By default, the Integration Service sorts null values high. This overrides the Integration Service configuration to treat nulls in comparison operators as high, low, or null. Note: For relational lookups, null ordering is based on the database support.
Sorted Input	Flat File	Indicates whether or not the lookup file data is sorted. This increases lookup performance for file lookups. If you enable sorted input, and the condition columns are not grouped, the Integration Service fails the session. If the condition columns are grouped, but not sorted, the Integration Service processes the lookup as if you did not configure sorted input. For more information about sorted input, see "Flat File Lookups" on page 311.

Configuring Lookup Properties in a Session

When you configure a session, you can configure lookup properties that are unique to sessions:

- ◆ **Flat file lookups.** Configure location information, such as the file directory, file name, and the file type.
- ◆ **Relational lookups.** You can define \$Source and \$Target variables in the session properties. You can also override connection information to use the session parameter \$DBConnection.

Configuring Flat File Lookups in a Session

Figure 14-1 shows the session properties for a flat file lookup:

Figure 14-1. Session Properties for Flat File Lookups

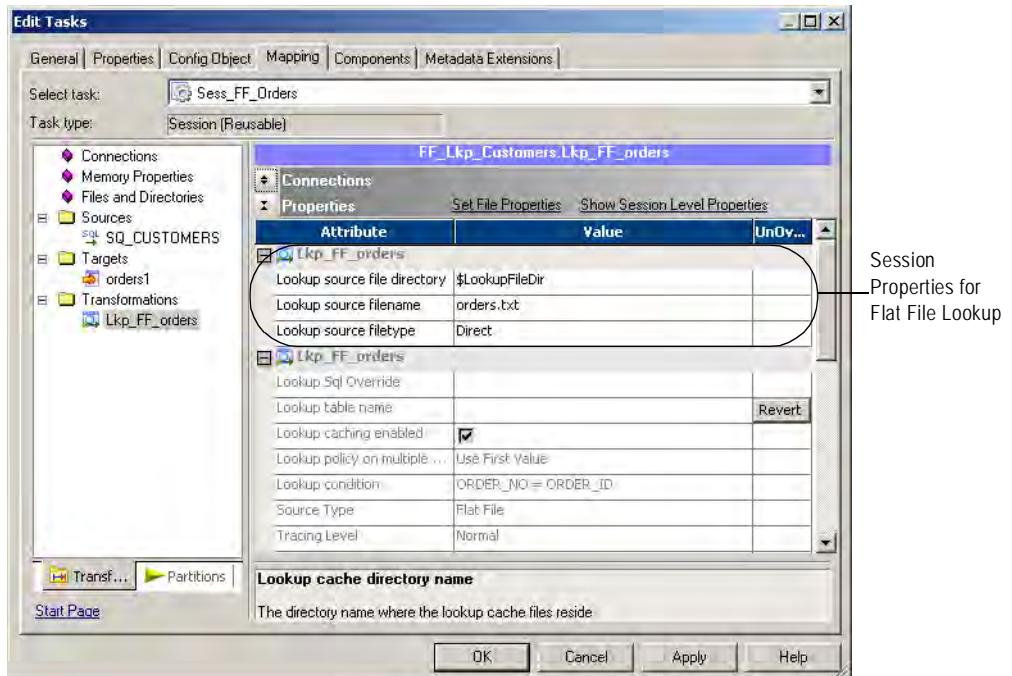


Table 14-4 describes the session properties you configure for flat file lookups:

Table 14-4. Session Properties for Flat File Lookups

Property	Description
Lookup Source File Directory	Enter the directory name. By default, the Integration Service looks in the process variable directory, \$PMLookupFileDir, for lookup files. You can enter the full path and file name. If you specify both the directory and file name in the Lookup Source Filename field, clear this field. The Integration Service concatenates this field with the Lookup Source Filename field when it runs the session. You can also use the \$InputFileName session parameter to specify the file name. For more information about session parameters, see “Working with Sessions” in the <i>Workflow Administration Guide</i> .
Lookup Source Filename	Name of the lookup file. If you use an indirect file, specify the name of the indirect file you want the Integration Service to read. You can also use the lookup file parameter, \$LookupFileName, to change the name of the lookup file a session uses. If you specify both the directory and file name in the Source File Directory field, clear this field. The Integration Service concatenates this field with the Lookup Source File Directory field when it runs the session. For example, if you have “C:\lookup_data” in the Lookup Source File Directory field, then enter “filename.txt” in the Lookup Source Filename field. When the Integration Service begins the session, it looks for “C:\lookup_data\filename.txt”. For more information, see “Working with Sessions” in the <i>Workflow Administration Guide</i> .
Lookup Source Filetype	Indicates whether the lookup source file contains the source data or a list of files with the same file properties. Choose Direct if the lookup source file contains the source data. Choose Indirect if the lookup source file contains a list of files. When you select Indirect, the Integration Service creates one cache for all files. If you use sorted input with indirect files, verify that the range of data in the files do not overlap. If the range of data overlaps, the Integration Service processes the lookup as if you did not configure for sorted input.

Configuring Relational Lookups in a Session

When you configure a session, you specify the connection for the lookup database in the Connection node on the Mapping tab (Transformation view). You have the following options to specify a connection:

- ♦ Choose any relational connection.
- ♦ Use the connection variable, \$DBConnection.
- ♦ Specify a database connection for \$Source or \$Target information.

If you use \$Source or \$Target for the lookup connection, configure the \$Source Connection Value and \$Target Connection Value in the session properties. This ensures that the Integration Service uses the correct database connection for the variable when it runs the session.

If you use \$Source or \$Target and you do not specify a Connection Value in the session properties, the Integration Service determines the database connection to use when it runs the session. It uses a source or target database connection for the source or target in the pipeline that contains the Lookup transformation. If it cannot determine which database connection to use, it fails the session.

The following list describes how the Integration Service determines the value of \$Source or \$Target when you do not specify \$Source Connection Value or \$Target Connection Value in the session properties:

- ◆ When you use \$Source and the pipeline contains one source, the Integration Service uses the database connection you specify for the source.
- ◆ When you use \$Source and the pipeline contains multiple sources joined by a Joiner transformation, the Integration Service uses different database connections, depending on the location of the Lookup transformation in the pipeline:
 - When the Lookup transformation is after the Joiner transformation, the Integration Service uses the database connection for the detail table.
 - When the Lookup transformation is before the Joiner transformation, the Integration Service uses the database connection for the source connected to the Lookup transformation.
- ◆ When you use \$Target and the pipeline contains one target, the Integration Service uses the database connection you specify for the target.
- ◆ When you use \$Target and the pipeline contains multiple relational targets, the session fails.
- ◆ When you use \$Source or \$Target in an unconnected Lookup transformation, the session fails.

Lookup Query

The Integration Service queries the lookup based on the ports and properties you configure in the Lookup transformation. The Integration Service runs a default SQL statement when the first row enters the Lookup transformation. If you use a relational lookup, you can customize the default query with the Lookup SQL Override property.

Default Lookup Query

The default lookup query contains the following statements:

- ◆ **SELECT.** The SELECT statement includes all the lookup ports in the mapping. You can view the SELECT statement by generating SQL using the Lookup SQL Override property. Do not add or delete any columns from the default SQL statement.
- ◆ **ORDER BY.** The ORDER BY clause orders the columns in the same order they appear in the Lookup transformation. The Integration Service generates the ORDER BY clause. You cannot view this when you generate the default SQL using the Lookup SQL Override property.

Overriding the Lookup Query

The lookup SQL override is similar to entering a custom query in a Source Qualifier transformation. You can override the lookup query for a relational lookup. You can enter the entire override, or you can generate and edit the default SQL statement. When the Designer generates the default SQL statement for the lookup SQL override, it includes the lookup/output ports in the lookup condition and the lookup/return port.

Override the lookup query in the following circumstances:

- ◆ **Override the ORDER BY clause.** Create the ORDER BY clause with fewer columns to increase performance. When you override the ORDER BY clause, you must suppress the generated ORDER BY clause with a comment notation. For more information, see “Overriding the ORDER BY Clause” on page 325.

Note: If you use pushdown optimization, you cannot override the ORDER BY clause or suppress the generated ORDER BY clause with a comment notation.

- ◆ **A lookup table name or column names contains a reserved word.** If the table name or any column name in the lookup query contains a reserved word, you must ensure that all reserved words are enclosed in quotes. For more information, see “Reserved Words” on page 326.
- ◆ **Use mapping parameters and variables.** Use mapping parameters and variables when you enter a lookup SQL override. However, the Designer cannot expand mapping parameters and variables in the query override and does not validate the lookup SQL override. When you run a session with a mapping parameter or variable in the lookup SQL override, the Integration Service expands mapping parameters and variables and connects to the lookup database to validate the query override. For more information about using mapping

parameters and variables in expressions, see “Mapping Parameters and Variables” in the *Designer Guide*.

- ◆ A **lookup column name contains a slash (/) character.** When generating the default lookup query, the Designer and Integration Service replace any slash character (/) in the lookup column name with an underscore character. To query lookup column names containing the slash character, override the default lookup query, replace the underscore characters with the slash character, and enclose the column name in double quotes.
 - ◆ Add a **WHERE statement.** Use a lookup SQL override to add a WHERE statement to the default SQL statement. You might want to use this to reduce the number of rows included in the cache. When you add a WHERE statement to a Lookup transformation using a dynamic cache, use a Filter transformation before the Lookup transformation. This ensures the Integration Service only inserts rows into the dynamic cache and target table that match the WHERE clause. For more information, see “Using the WHERE Clause with a Dynamic Cache” on page 358.
- Note:** The session fails if you include large object ports in a WHERE clause.
- ◆ **Other.** Use a lookup SQL override if you want to query lookup data from multiple lookups or if you want to modify the data queried from the lookup table before the Integration Service caches the lookup rows. For example, use TO_CHAR to convert dates to strings.

Overriding the ORDER BY Clause

By default, the Integration Service generates an ORDER BY clause for a cached lookup. The ORDER BY clause contains all lookup ports. To increase performance, you can suppress the default ORDER BY clause and enter an override ORDER BY with fewer columns.

Note: If you use pushdown optimization, you cannot override the ORDER BY clause or suppress the generated ORDER BY clause with a comment notation.

The Integration Service always generates an ORDER BY clause, even if you enter one in the override. Place two dashes ‘--’ after the ORDER BY override to suppress the generated ORDER BY clause. For example, a Lookup transformation uses the following lookup condition:

```
ITEM_ID = IN_ITEM_ID
PRICE <= IN_PRICE
```

The Lookup transformation includes three lookup ports used in the mapping, ITEM_ID, ITEM_NAME, and PRICE. When you enter the ORDER BY clause, enter the columns in the same order as the ports in the lookup condition. You must also enclose all database reserved words in quotes. Enter the following lookup query in the lookup SQL override:

```
SELECT ITEMS_DIM.ITEM_NAME, ITEMS_DIM.PRICE, ITEMS_DIM.ITEM_ID FROM
ITEMS_DIM ORDER BY ITEMS_DIM.ITEM_ID, ITEMS_DIM.PRICE --
```

To override the default ORDER BY clause for a relational lookup, complete the following steps:

1. Generate the lookup query in the Lookup transformation.
2. Enter an ORDER BY clause that contains the condition ports in the same order they appear in the Lookup condition.
3. Place two dashes '--' as a comment notation after the ORDER BY clause to suppress the ORDER BY clause that the Integration Service generates.

If you override the lookup query with an ORDER BY clause without adding comment notation, the lookup fails.

Note: Sybase has a 16 column ORDER BY limitation. If the Lookup transformation has more than 16 lookup/output ports (including the ports in the lookup condition), you might want to override the ORDER BY clause or use multiple Lookup transformations to query the lookup table.

Reserved Words

If any lookup name or column name contains a database reserved word, such as MONTH or YEAR, the session fails with database errors when the Integration Service executes SQL against the database. You can create and maintain a reserved words file, reswords.txt, in the Integration Service installation directory. When the Integration Service initializes a session, it searches for reswords.txt. If the file exists, the Integration Service places quotes around matching reserved words when it executes SQL against the database.

You may need to enable some databases, such as Microsoft SQL Server and Sybase, to use SQL-92 standards regarding quoted identifiers. Use connection environment SQL to issue the command. For example, with Microsoft SQL Server, use the following command:

```
SET QUOTED_IDENTIFIER ON
```

Note: The reserved words file, reswords.txt, is a file that you create and maintain in the Integration Service installation directory. The Integration Service searches this file and places quotes around reserved words when it executes SQL against source, target, and lookup databases. For more information about reswords.txt, see "Working with Targets" in the *Workflow Administration Guide*.

Guidelines to Overriding the Lookup Query

Use the following guidelines when you override the lookup SQL query:

- ◆ You can only override the lookup SQL query for relational lookups.
- ◆ Configure the Lookup transformation for caching. If you do not enable caching, the Integration Service does not recognize the override.
- ◆ Generate the default query, and then configure the override. This helps ensure that all the lookup/output ports are included in the query. If you add or subtract ports from the SELECT statement, the session fails.

- ◆ Use a Filter transformation before a Lookup transformation using a dynamic cache when you add a WHERE clause to the lookup SQL override. This ensures the Integration Service only inserts rows in the dynamic cache and target table that match the WHERE clause. For more information, see “Using the WHERE Clause with a Dynamic Cache” on page 358.
- ◆ If you want to share the cache, use the same lookup SQL override for each Lookup transformation.
- ◆ If you override the ORDER BY clause, the session fails if the ORDER BY clause does not contain the condition ports in the same order they appear in the Lookup condition or if you do not suppress the generated ORDER BY clause with the comment notation.
- ◆ If you use pushdown optimization, you cannot override the ORDER BY clause or suppress the generated ORDER BY clause with comment notation.
- ◆ If the table name or any column name in the lookup query contains a reserved word, you must enclose all reserved words in quotes.

Steps to Overriding the Lookup Query

Use the following steps to override the default lookup SQL query.

To override the default lookup query:

1. On the Properties tab, open the SQL Editor from within the Lookup SQL Override field.
2. Click Generate SQL to generate the default SELECT statement. Enter the lookup SQL override.
3. Connect to a database, and then click Validate to test the lookup SQL override.
4. Click OK to return to the Properties tab.

Lookup Condition

The Integration Service uses the lookup condition to test incoming values. It is similar to the WHERE clause in an SQL query. When you configure a lookup condition for the transformation, you compare transformation input values with values in the lookup source or cache, represented by lookup ports. When you run a workflow, the Integration Service queries the lookup source or cache for all incoming values based on the condition.

You must enter a lookup condition in all Lookup transformations. Some guidelines for the lookup condition apply for all Lookup transformations, and some guidelines vary depending on how you configure the transformation.

Use the following guidelines when you enter a condition for a Lookup transformation:

- ◆ The datatypes in a condition must match.
- ◆ Use one input port for each lookup port used in the condition. Use the same input port in more than one condition in a transformation.
- ◆ When you enter multiple conditions, the Integration Service evaluates each condition as an AND, not an OR. The Integration Service returns only rows that match all the conditions you specify.
- ◆ The Integration Service matches null values. For example, if an input lookup condition column is NULL, the Integration Service evaluates the NULL equal to a NULL in the lookup.
- ◆ If you configure a flat file lookup for sorted input, the Integration Service fails the session if the condition columns are not grouped. If the columns are grouped, but not sorted, the Integration Service processes the lookup as if you did not configure sorted input. For more information about sorted input, see “Flat File Lookups” on page 311.

The lookup condition guidelines and the way the Integration Service processes matches can vary, depending on whether you configure the transformation for a dynamic cache or an uncached or static cache. For more information about lookup caches, see “Lookup Caches” on page 337.

Uncached or Static Cache

Use the following guidelines when you configure a Lookup transformation without a cache or to use a static cache:

- ◆ Use the following operators when you create the lookup condition:

=, >, <, >=, <=, !=

Tip: If you include more than one lookup condition, place the conditions with an equal sign first to optimize lookup performance. For example, create the following lookup condition:

```
ITEM_ID = IN_ITEM_ID  
PRICE <= IN_PRICE
```

- ◆ The input value must meet all conditions for the lookup to return a value.

The condition can match equivalent values or supply a threshold condition. For example, you might look for customers who do not live in California, or employees whose salary is greater than \$30,000. Depending on the nature of the source and condition, the lookup might return multiple values.

Dynamic Cache

If you configure a Lookup transformation to use a dynamic cache, you can only use the equality operator (=) in the lookup condition.

Handling Multiple Matches

Lookups find a value based on the conditions you set in the Lookup transformation. If the lookup condition is not based on a unique key, or if the lookup source is denormalized, the Integration Service might find multiple matches in the lookup source or cache.

You can configure a Lookup transformation to handle multiple matches in the following ways:

- ◆ **Return the first matching value, or return the last matching value.** You can configure the transformation to return the first matching value or the last matching value. The first and last values are the first value and last value found in the lookup cache that match the lookup condition. When you cache the lookup source, the Integration Service generates an ORDER BY clause for each column in the lookup cache to determine the first and last row in the cache. The Integration Service then sorts each lookup source column in ascending order.

The Integration Service sorts numeric columns in ascending numeric order (such as 0 to 10), date/time columns from January to December and from the first of the month to the end of the month, and string columns based on the sort order configured for the session.

- ◆ **Return any matching value.** You can configure the Lookup transformation to return any value that matches the lookup condition. When you configure the Lookup transformation to return any matching value, the transformation returns the first value that matches the lookup condition. It creates an index based on the key ports rather than all Lookup transformation ports. When you use any matching value, performance can improve because the process of indexing rows is simplified.
- ◆ **Return an error.** When the Lookup transformation uses a static cache or no cache, the Integration Service marks the row as an error, writes the row to the session log by default, and increases the error count by one. When the Lookup transformation uses a dynamic cache, the Integration Service fails the session when it encounters multiple matches either while caching the lookup table or looking up values in the cache that contain duplicate keys. Also, if you configure the Lookup transformation to output old values on updates, the Lookup transformation returns an error when it encounters multiple matches.

Lookup Caches

You can configure a Lookup transformation to cache the lookup file or table. The Integration Service builds a cache in memory when it processes the first row of data in a cached Lookup transformation. It allocates memory for the cache based on the amount you configure in the transformation or session properties. The Integration Service stores condition values in the index cache and output values in the data cache. The Integration Service queries the cache for each row that enters the transformation.

The Integration Service also creates cache files by default in the \$PMCacheDir. If the data does not fit in the memory cache, the Integration Service stores the overflow values in the cache files. When the session completes, the Integration Service releases cache memory and deletes the cache files unless you configure the Lookup transformation to use a persistent cache.

When configuring a lookup cache, you can specify any of the following options:

- ◆ Persistent cache
- ◆ Recache from lookup source
- ◆ Static cache
- ◆ Dynamic cache
- ◆ Shared cache

Note: You can use a dynamic cache for relational or flat file lookups.

For more information about working with lookup caches, see “Lookup Caches” on page 337.

Configuring Unconnected Lookup Transformations

An unconnected Lookup transformation is separate from the pipeline in the mapping. You write an expression using the :LKP reference qualifier to call the lookup within another transformation. Some common uses for unconnected lookups include:

- ◆ Testing the results of a lookup in an expression
- ◆ Filtering rows based on the lookup results
- ◆ Marking rows for update based on the result of a lookup, such as updating slowly changing dimension tables
- ◆ Calling the same lookup multiple times in one mapping

Complete the following steps when you configure an unconnected Lookup transformation:

1. Add input ports.
2. Add the lookup condition.
3. Designate a return value.
4. Call the lookup from another transformation.

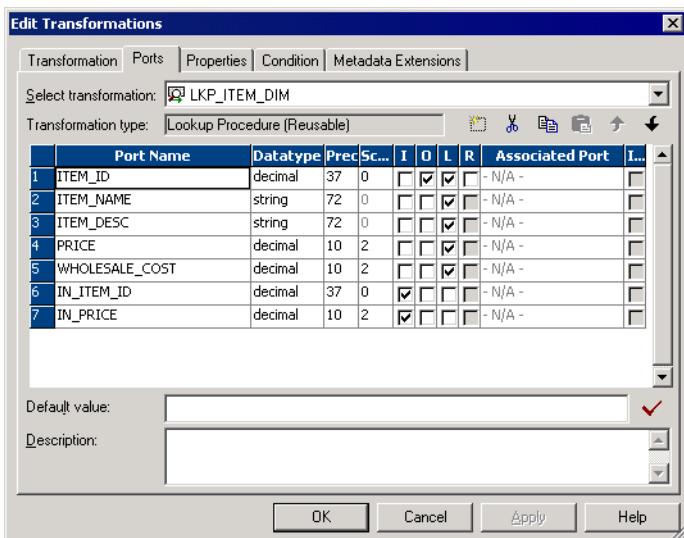
Step 1. Add Input Ports

Create an input port for each argument in the :LKP expression. For each lookup condition you plan to create, you need to add an input port to the Lookup transformation. You can create a different port for each condition, or use the same input port in more than one condition.

For example, a retail store increased prices across all departments during the last month. The accounting department only wants to load rows into the target for items with increased prices. To accomplish this, complete the following tasks:

- ◆ Create a lookup condition that compares the ITEM_ID in the source with the ITEM_ID in the target.
- ◆ Compare the PRICE for each item in the source with the price in the target table.
 - If the item exists in the target table and the item price in the source is less than or equal to the price in the target table, you want to delete the row.
 - If the price in the source is greater than the item price in the target table, you want to update the row.

- ♦ Create an input port (IN_ITEM_ID) with datatype Decimal (37,0) to match the ITEM_ID and an IN_PRICE input port with Decimal (10,2) to match the PRICE lookup port.



Step 2. Add the Lookup Condition

After you correctly configure the ports, define a lookup condition to compare transformation input values with values in the lookup source or cache. To increase performance, add conditions with an equal sign first.

In this case, add the following lookup condition:

```
ITEM_ID = IN_ITEM_ID
PRICE <= IN_PRICE
```

If the item exists in the mapping source and lookup source and the mapping source price is less than or equal to the lookup price, the condition is true and the lookup returns the values designated by the Return port. If the lookup condition is false, the lookup returns NULL. Therefore, when you write the update strategy expression, use ISNULL nested in an IIF to test for null values.

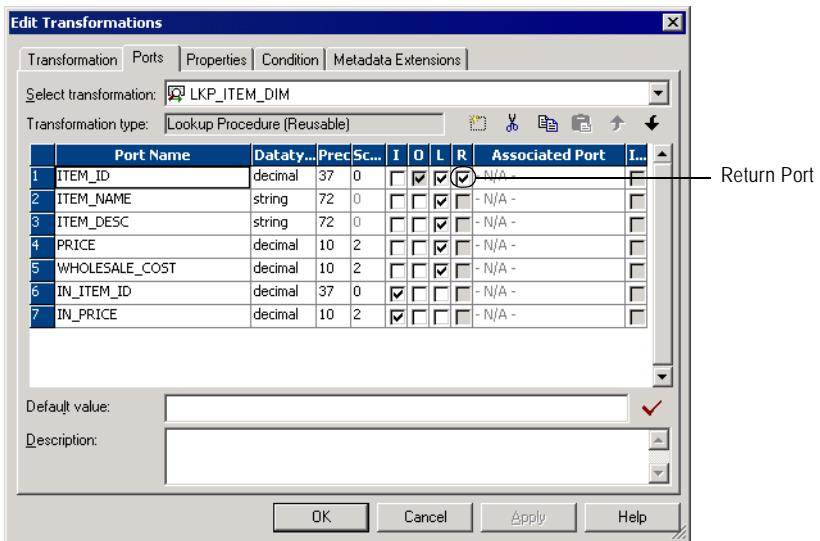
Step 3. Designate a Return Value

With unconnected Lookups, you can pass multiple input values into the transformation, but only one column of data out of the transformation. Designate one lookup/output port as a return port. The Integration Service can return one value from the lookup query. Use the return port to specify the return value. If you call the unconnected lookup from an update strategy or filter expression, you are generally checking for null values. In this case, the return port can be anything. If, however, you call the lookup from an expression performing a calculation, the return value needs to be the value you want to include in the calculation.

To continue the update strategy example, you can define the ITEM_ID port as the return port. The update strategy expression checks for null values returned. If the lookup condition is true, the Integration Service returns the ITEM_ID. If the condition is false, the Integration Service returns NULL.

Figure 14-2 shows a return port in a Lookup transformation:

Figure 14-2. Return Port in a Lookup Transformation



Step 4. Call the Lookup Through an Expression

You supply input values for an unconnected Lookup transformation from a :LKP expression in another transformation. The arguments are local input ports that match the Lookup transformation input ports used in the lookup condition. Use the following syntax for a :LKP expression:

```
:LKP.lookup_transformation_name(argument, argument, ...)
```

To continue the example about the retail store, when you write the update strategy expression, the order of ports in the expression must match the order in the lookup condition. In this case, the ITEM_ID condition is the first lookup condition, and therefore, it is the first argument in the update strategy expression.

```
IIF(ISNULL(:LKP.lkpITEMS_DIM(ITEM_ID, PRICE)), DD_UPDATE, DD_REJECT)
```

Use the following guidelines to write an expression that calls an unconnected Lookup transformation:

- ◆ The order in which you list each argument must match the order of the lookup conditions in the Lookup transformation.
- ◆ The datatypes for the ports in the expression must match the datatypes for the input ports in the Lookup transformation. The Designer does not validate the expression if the datatypes do not match.

- ◆ If one port in the lookup condition is not a lookup/output port, the Designer does not validate the expression.
- ◆ The arguments (ports) in the expression must be in the same order as the input ports in the lookup condition.
- ◆ If you use incorrect :LKP syntax, the Designer marks the mapping invalid.
- ◆ If you call a connected Lookup transformation in a :LKP expression, the Designer marks the mapping invalid.

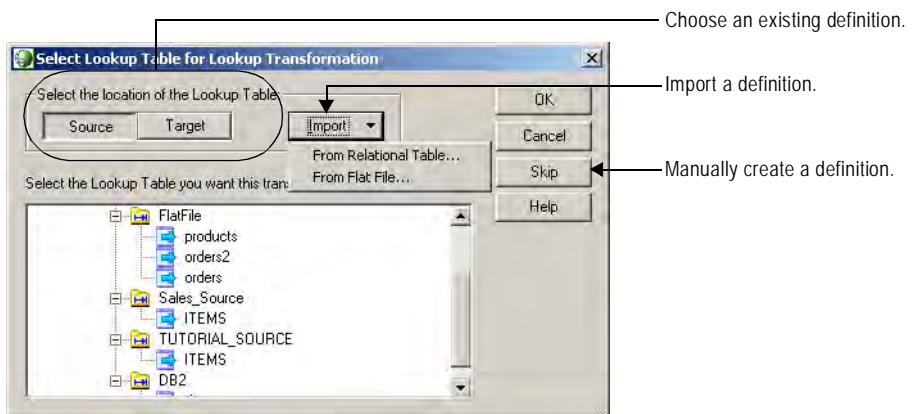
Tip: Avoid syntax errors when you enter expressions by using the point-and-click method to select functions and ports.

Creating a Lookup Transformation

The following steps summarize the process of creating a Lookup transformation.

To create a Lookup transformation:

1. In the Mapping Designer, click Transformation > Create. Select the Lookup transformation. Enter a name for the transformation. The naming convention for Lookup transformations is LKP_*TransformationName*. Click OK.
2. In the Select Lookup Table dialog box, you can choose the following options:
 - ◆ Choose an existing table or file definition.
 - ◆ Choose to import a definition from a relational table or file.
 - ◆ Skip to create a manual definition.



3. Define input ports for each lookup condition you want to define.
4. For an unconnected Lookup transformation, create a return port for the value you want to return from the lookup.
5. Define output ports for the values you want to pass to another transformation.
6. For Lookup transformations that use a dynamic lookup cache, associate an input port or sequence ID with each lookup port.
7. Add the lookup conditions. If you include more than one condition, place the conditions using equal signs first to optimize lookup performance.

For information about lookup conditions, see “Lookup Condition” on page 328.

8. On the Properties tab, set the properties for the Lookup transformation, and click OK.
For a list of properties, see “Lookup Properties” on page 316.
9. For unconnected Lookup transformations, write an expression in another transformation using :LKP to call the unconnected Lookup transformation.

Tips

Use the following tips when you configure the Lookup transformation:

Add an index to the columns used in a lookup condition.

If you have privileges to modify the database containing a lookup table, you can improve performance for both cached and uncached lookups. This is important for very large lookup tables. Since the Integration Service needs to query, sort, and compare values in these columns, the index needs to include every column used in a lookup condition.

Place conditions with an equality operator (=) first.

If a Lookup transformation specifies several conditions, you can improve lookup performance by placing all the conditions that use the equality operator first in the list of conditions that appear under the Condition tab.

Cache small lookup tables.

Improve session performance by caching small lookup tables. The result of the lookup query and processing is the same, whether or not you cache the lookup table.

Join tables in the database.

If the lookup table is on the same database as the source table in the mapping and caching is not feasible, join the tables in the source database rather than using a Lookup transformation.

Use a persistent lookup cache for static lookups.

If the lookup source does not change between sessions, configure the Lookup transformation to use a persistent lookup cache. The Integration Service then saves and reuses cache files from session to session, eliminating the time required to read the lookup source.

Call unconnected Lookup transformations with the :LKP reference qualifier.

When you write an expression using the :LKP reference qualifier, you call unconnected Lookup transformations only. If you try to call a connected Lookup transformation, the Designer displays an error and marks the mapping invalid.

Chapter 15

Lookup Caches

This chapter includes the following topics:

- ◆ Overview, 338
- ◆ Building Connected Lookup Caches, 340
- ◆ Using a Persistent Lookup Cache, 342
- ◆ Working with an Uncached Lookup or Static Cache, 344
- ◆ Working with a Dynamic Lookup Cache, 345
- ◆ Sharing the Lookup Cache, 363
- ◆ Lookup Cache Tips, 369

Overview

You can configure a Lookup transformation to cache the lookup table. The Integration Service builds a cache in memory when it processes the first row of data in a cached Lookup transformation. It allocates memory for the cache based on the amount you configure in the transformation or session properties. The Integration Service stores condition values in the index cache and output values in the data cache. The Integration Service queries the cache for each row that enters the transformation.

The Integration Service also creates cache files by default in the \$PMCacheDir. If the data does not fit in the memory cache, the Integration Service stores the overflow values in the cache files. When the session completes, the Integration Service releases cache memory and deletes the cache files unless you configure the Lookup transformation to use a persistent cache.

If you use a flat file lookup, the Integration Service always caches the lookup source. If you configure a flat file lookup for sorted input, the Integration Service cannot cache the lookup if the condition columns are not grouped. If the columns are grouped, but not sorted, the Integration Service processes the lookup as if you did not configure sorted input. For more information, see “Flat File Lookups” on page 311.

When you configure a lookup cache, you can configure the following cache settings:

- ◆ **Building caches.** You can configure the session to build caches sequentially or concurrently. When you build sequential caches, the Integration Service creates caches as the source rows enter the Lookup transformation. When you configure the session to build concurrent caches, the Integration Service does not wait for the first row to enter the Lookup transformation before it creates caches. Instead, it builds multiple caches concurrently. For more information, see “Building Connected Lookup Caches” on page 340.
- ◆ **Persistent cache.** You can save the lookup cache files and reuse them the next time the Integration Service processes a Lookup transformation configured to use the cache. For more information, see “Using a Persistent Lookup Cache” on page 342.
- ◆ **Recache from source.** If the persistent cache is not synchronized with the lookup table, you can configure the Lookup transformation to rebuild the lookup cache. For more information, see “Building Connected Lookup Caches” on page 340.
- ◆ **Static cache.** You can configure a static, or read-only, cache for any lookup source. By default, the Integration Service creates a static cache. It caches the lookup file or table and looks up values in the cache for each row that comes into the transformation. When the lookup condition is true, the Integration Service returns a value from the lookup cache. The Integration Service does not update the cache while it processes the Lookup transformation. For more information, see “Working with an Uncached Lookup or Static Cache” on page 344.
- ◆ **Dynamic cache.** To cache a target table or flat file source and insert new rows or update existing rows in the cache, use a Lookup transformation with a dynamic cache. The Integration Service dynamically inserts or updates data in the lookup cache and passes data

to the target. For more information, see “Working with a Dynamic Lookup Cache” on page 345.

- ◆ **Shared cache.** You can share the lookup cache between multiple transformations. You can share an unnamed cache between transformations in the same mapping. You can share a named cache between transformations in the same or different mappings. For more information, see “Sharing the Lookup Cache” on page 363.

When you do not configure the Lookup transformation for caching, the Integration Service queries the lookup table for each input row. The result of the Lookup query and processing is the same, whether or not you cache the lookup table. However, using a lookup cache can increase session performance. Optimize performance by caching the lookup table when the source table is large.

For more information about caching properties, see “Lookup Properties” on page 316.

For information about configuring the cache size, see “Session Caches” in the *Workflow Administration Guide*.

Note: The Integration Service uses the same transformation logic to process a Lookup transformation whether you configure it to use a static cache or no cache. However, when you configure the transformation to use no cache, the Integration Service queries the lookup table instead of the lookup cache.

Cache Comparison

Table 15-1 compares the differences between an uncached lookup, a static cache, and a dynamic cache:

Table 15-1. Lookup Caching Comparison

Uncached	Static Cache	Dynamic Cache
You cannot insert or update the cache.	You cannot insert or update the cache.	You can insert or update rows in the cache as you pass rows to the target.
You cannot use a flat file lookup.	Use a relational or a flat file lookup.	Use a relational or a flat file lookup.
When the condition is true, the Integration Service returns a value from the lookup table or cache. When the condition is not true, the Integration Service returns the default value for connected transformations and NULL for unconnected transformations. For more information, see “Working with an Uncached Lookup or Static Cache” on page 344.	When the condition is true, the Integration Service returns a value from the lookup table or cache. When the condition is not true, the Integration Service returns the default value for connected transformations and NULL for unconnected transformations. For more information, see “Working with an Uncached Lookup or Static Cache” on page 344.	When the condition is true, the Integration Service either updates rows in the cache or leaves the cache unchanged, depending on the row type. This indicates that the row is in the cache and target table. You can pass updated rows to a target. When the condition is not true, the Integration Service either inserts rows into the cache or leaves the cache unchanged, depending on the row type. This indicates that the row is not in the cache or target. You can pass inserted rows to a target table. For more information, see “Updating the Dynamic Lookup Cache” on page 356.

Building Connected Lookup Caches

The Integration Service can build lookup caches for connected Lookup transformations in the following ways:

- ♦ **Sequential caches.** The Integration Service builds lookup caches sequentially. The Integration Service builds the cache in memory when it processes the first row of the data in a cached lookup transformation. For more information, see “[Sequential Caches](#)” on page 340.
- ♦ **Concurrent caches.** The Integration Service builds lookup caches concurrently. It does not need to wait for data to reach the Lookup transformation. For more information, see “[Concurrent Caches](#)” on page 341.

Note: The Integration Service builds caches for unconnected Lookup transformations sequentially regardless of how you configure cache building. If you configure the session to build concurrent caches for an unconnected Lookup transformation, the Integration Service ignores this setting and builds unconnected Lookup transformation caches sequentially.

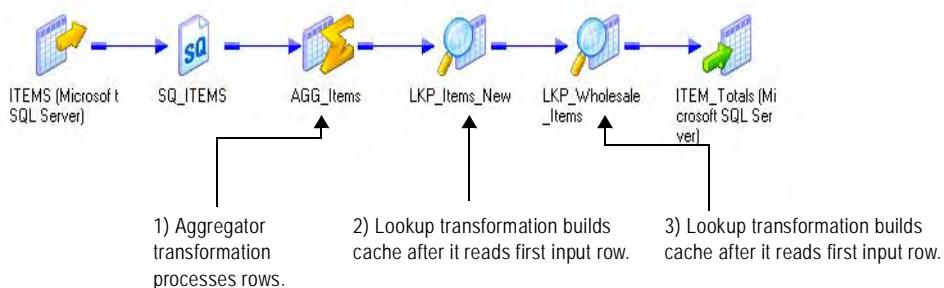
Sequential Caches

By default, the Integration Service builds a cache in memory when it processes the first row of data in a cached Lookup transformation. The Integration Service creates each lookup cache in the pipeline sequentially. The Integration Service waits for any upstream active transformation to complete processing before it starts processing the rows in the Lookup transformation. The Integration Service does not build caches for a downstream Lookup transformation until an upstream Lookup transformation completes building a cache.

For example, the following mapping contains an unsorted Aggregator transformation followed by two Lookup transformations.

Figure 15-1 shows a mapping that contains multiple Lookup transformations:

Figure 15-1. Building Lookup Caches Sequentially



The Integration Service processes all the rows for the unsorted Aggregator transformation and begins processing the first Lookup transformation after the unsorted Aggregator

transformation completes. When it processes the first input row, the Integration Service begins building the first lookup cache. After the Integration Service finishes building the first lookup cache, it can begin processing the lookup data. The Integration Service begins building the next lookup cache when the first row of data reaches the Lookup transformation.

You might want to process lookup caches sequentially if the Lookup transformation may not process row data. The Lookup transformation may not process row data if the transformation logic is configured to route data to different pipelines based on a condition. Configuring sequential caching may allow you to avoid building lookup caches unnecessarily. For example, a Router transformation might route data to one pipeline if a condition resolves to true, and it might route data to another pipeline if the condition resolves to false. In this case, a Lookup transformation might not receive data at all.

Concurrent Caches

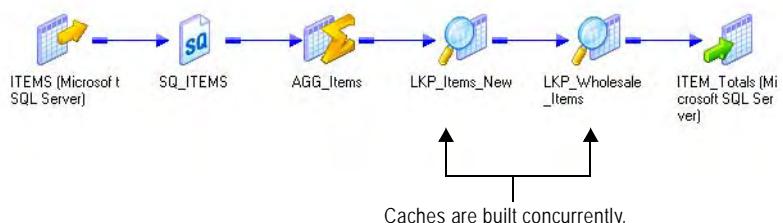
You can configure the Integration Service to create lookup caches concurrently. You may be able to improve session performance using concurrent caches. Performance may especially improve when the pipeline contains an active transformations upstream of the Lookup transformation. You may want to configure the session to create concurrent caches if you are certain that you will need to build caches for each of the Lookup transformations in the session.

When you configure the Lookup transformation to create concurrent caches, it does not wait for upstream transformations to complete before it creates lookup caches, and it does not need to finish building a lookup cache before it can begin building other lookup caches.

For example, you configure the session shown in Figure 15-1 for concurrent cache creation.

Figure 15-2 shows lookup transformation caches built concurrently:

Figure 15-2. Building Lookup Caches Concurrently



When you run the session, the Integration Service builds the Lookup caches concurrently. It does not wait for upstream transformations to complete, and it does not wait for other Lookup transformations to complete cache building.

Note: You cannot process caches for unconnected Lookup transformations concurrently.

To configure the session to create concurrent caches, configure a value for the session configuration attribute, Additional Concurrent Pipelines for Lookup Cache Creation.

Using a Persistent Lookup Cache

You can configure a Lookup transformation to use a non-persistent or persistent cache. The Integration Service saves or deletes lookup cache files after a successful session based on the Lookup Cache Persistent property.

If the lookup table does not change between sessions, you can configure the Lookup transformation to use a persistent lookup cache. The Integration Service saves and reuses cache files from session to session, eliminating the time required to read the lookup table.

Using a Non-Persistent Cache

By default, the Integration Service uses a non-persistent cache when you enable caching in a Lookup transformation. The Integration Service deletes the cache files at the end of a session. The next time you run the session, the Integration Service builds the memory cache from the database.

Using a Persistent Cache

If you want to save and reuse the cache files, you can configure the transformation to use a persistent cache. Use a persistent cache when you know the lookup table does not change between session runs.

The first time the Integration Service runs a session using a persistent lookup cache, it saves the cache files to disk instead of deleting them. The next time the Integration Service runs the session, it builds the memory cache from the cache files. If the lookup table changes occasionally, you can override session properties to recache the lookup from the database.

When you use a persistent lookup cache, you can specify a name for the cache files. When you specify a named cache, you can share the lookup cache across sessions. For more information about the Cache File Name Prefix property, see “Lookup Properties” on page 316. For more information about sharing lookup caches, see “Sharing the Lookup Cache” on page 363.

Rebuilding the Lookup Cache

You can instruct the Integration Service to rebuild the lookup cache if you think that the lookup source changed since the last time the Integration Service built the persistent cache.

When you rebuild a cache, the Integration Service creates new cache files, overwriting existing persistent cache files. The Integration Service writes a message to the session log when it rebuilds the cache.

You can rebuild the cache when the mapping contains one Lookup transformation or when the mapping contains Lookup transformations in multiple target load order groups that share a cache. You do not need to rebuild the cache when a dynamic lookup shares the cache with a static lookup in the same mapping.

If the Integration Service cannot reuse the cache, it either recaches the lookup from the database, or it fails the session, depending on the mapping and session properties.

Table 15-2 summarizes how the Integration Service handles persistent caching for named and unnamed caches:

Table 15-2. Integration Service Handling of Persistent Caches

Mapping or Session Changes Between Sessions	Named Cache	Unnamed Cache
Integration Service cannot locate cache files.	Rebuilds cache.	Rebuilds cache.
Enable or disable the Enable High Precision option in session properties.	Fails session.	Rebuilds cache.
Edit the transformation in the Mapping Designer, Mapplet Designer, or Reusable Transformation Developer.*	Fails session.	Rebuilds cache.
Edit the mapping (excluding Lookup transformation).	Reuses cache.	Rebuilds cache.
Change database connection or the file location used to access the lookup table.	Fails session.	Rebuilds cache.
Change the Integration Service data movement mode.	Fails session.	Rebuilds cache.
Change the sort order in Unicode mode.	Fails session.	Rebuilds cache.
Change the Integration Service code page to a compatible code page.	Reuses cache.	Reuses cache.
Change the Integration Service code page to an incompatible code page.	Fails session.	Rebuilds cache.

**Editing properties such as transformation description or port description does not affect persistent cache handling.*

Working with an Uncached Lookup or Static Cache

By default, the Integration Service creates a static lookup cache when you configure a Lookup transformation for caching. The Integration Service builds the cache when it processes the first lookup request. It queries the cache based on the lookup condition for each row that passes into the transformation. The Integration Service does not update the cache while it processes the transformation. The Integration Service processes an uncached lookup the same way it processes a cached lookup except that it queries the lookup source instead of building and querying the cache.

When the lookup condition is true, the Integration Service returns the values from the lookup source or cache. For connected Lookup transformations, the Integration Service returns the values represented by the lookup/output ports. For unconnected Lookup transformations, the Integration Service returns the value represented by the return port.

When the condition is not true, the Integration Service returns either NULL or default values. For connected Lookup transformations, the Integration Service returns the default value of the output port when the condition is not met. For unconnected Lookup transformations, the Integration Service returns NULL when the condition is not met.

When you create multiple partitions in a pipeline that use a static cache, the Integration Service creates one memory cache for each partition and one disk cache for each transformation.

For more information, see “Session Caches” in the *Workflow Administration Guide*.

Working with a Dynamic Lookup Cache

You can use a dynamic cache with a relational lookup or a flat file lookup. For relational lookups, you might configure the transformation to use a dynamic cache when the target table is also the lookup table. For flat file lookups, the dynamic cache represents the data to update in the target table.

The Integration Service builds the cache when it processes the first lookup request. It queries the cache based on the lookup condition for each row that passes into the transformation. When you use a dynamic cache, the Integration Service updates the lookup cache as it passes rows to the target.

When the Integration Service reads a row from the source, it updates the lookup cache by performing one of the following actions:

- ◆ **Inserts the row into the cache.** The row is not in the cache and you specified to insert rows into the cache. You can configure the transformation to insert rows into the cache based on input ports or generated sequence IDs. The Integration Service flags the row as insert.
- ◆ **Updates the row in the cache.** The row exists in the cache and you specified to update rows in the cache. The Integration Service flags the row as update. The Integration Service updates the row in the cache based on the input ports.
- ◆ **Makes no change to the cache.** The row exists in the cache and you specified to insert new rows only. Or, the row is not in the cache and you specified to update existing rows only. Or, the row is in the cache, but based on the lookup condition, nothing changes. The Integration Service flags the row as unchanged.

The Integration Service either inserts or updates the cache or makes no change to the cache, based on the results of the lookup query, the row type, and the Lookup transformation properties you define. For more information, see “Updating the Dynamic Lookup Cache” on page 356.

The following list describes some situations when you use a dynamic lookup cache:

- ◆ **Updating a master customer table with new and updated customer information.** You want to load new and updated customer information into a master customer table. Use a Lookup transformation that performs a lookup on the target table to determine if a customer exists or not. Use a dynamic lookup cache that inserts and updates rows in the cache as it passes rows to the target.
- ◆ **Loading data into a slowly changing dimension table and a fact table.** You want to load data into a slowly changing dimension table and a fact table. Create two pipelines and use a Lookup transformation that performs a lookup on the dimension table. Use a dynamic lookup cache to load data to the dimension table. Use a static lookup cache to load data to the fact table, making sure you specify the name of the dynamic cache from the first pipeline. For more information, see “Example Using a Dynamic Lookup Cache” on page 360.
- ◆ **Reading a flat file that is an export from a relational table.** You want to read data from a Teradata table, but the ODBC connection is slow. You can export the Teradata table contents to a flat file and use the file as a lookup source. You can pass the lookup cache

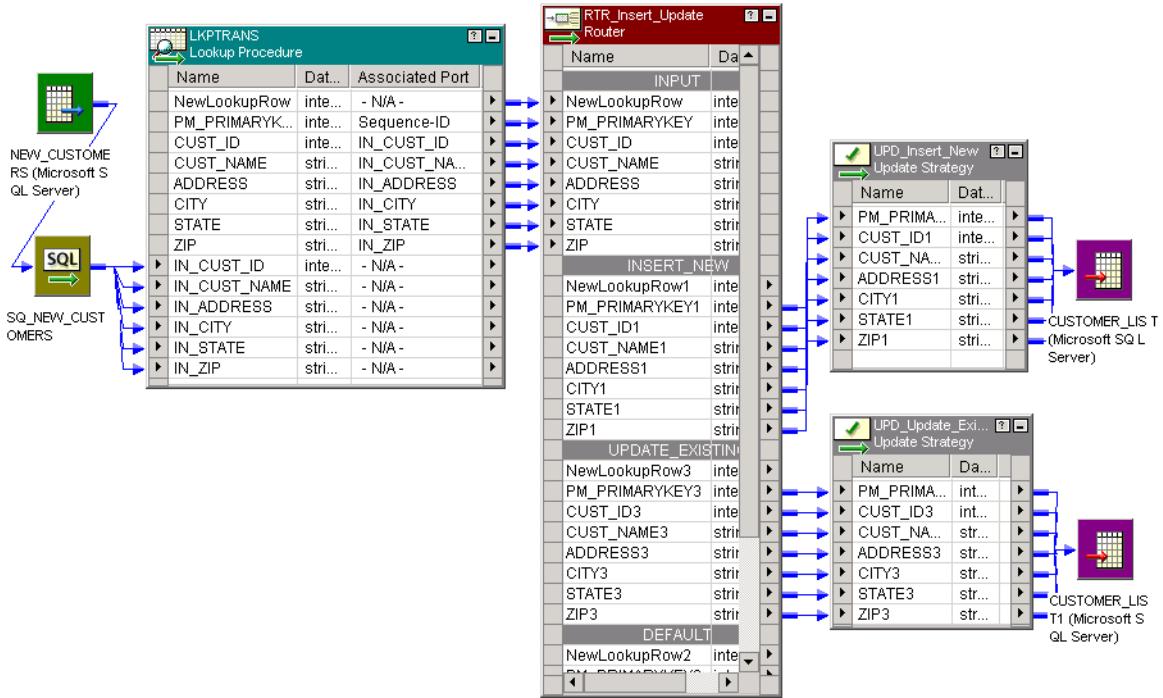
changes back to the Teradata table if you configure the Teradata table as a relational target in the mapping.

Use a Router or Filter transformation with the dynamic Lookup transformation to route inserted or updated rows to the cached target table. You can route unchanged rows to another target table or flat file, or you can drop them.

When you create multiple partitions in a pipeline that use a dynamic lookup cache, the Integration Service creates one memory cache and one disk cache for each transformation. However, if you add a partition point at the Lookup transformation, the Integration Service creates one memory cache for each partition. For more information, see “Session Caches” in the *Workflow Administration Guide*.

Figure 15-3 shows a mapping with a Lookup transformation that uses a dynamic lookup cache:

Figure 15-3. Mapping with a Dynamic Lookup Cache



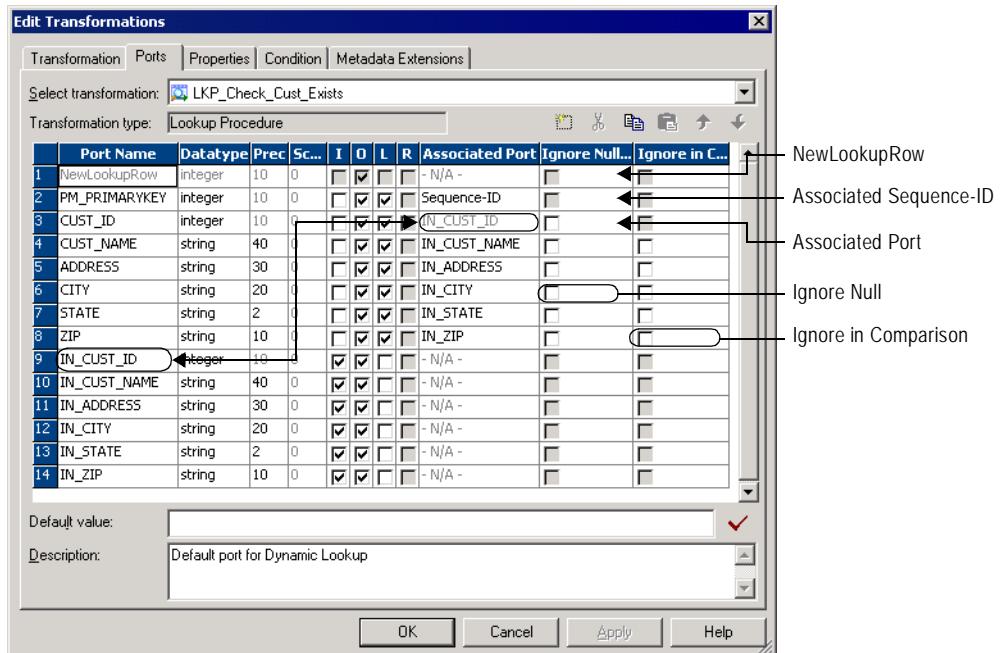
A Lookup transformation using a dynamic cache has the following properties:

- ♦ **NewLookupRow.** The Designer adds this port to a Lookup transformation configured to use a dynamic cache. Indicates with a numeric value whether the Integration Service inserts or updates the row in the cache, or makes no change to the cache. To keep the lookup cache and the target table synchronized, you pass rows to the target when the NewLookupRow value is equal to 1 or 2. For more information, see “Using the NewLookupRow Port” on page 347.

- ◆ **Associated Port.** Associate lookup ports with either an input/output port or a sequence ID. The Integration Service uses the data in the associated ports to insert or update rows in the lookup cache. If you associate a sequence ID, the Integration Service generates a primary key for inserted rows in the lookup cache. For more information, see “Using the Associated Input Port” on page 348.
- ◆ **Ignore Null Inputs for Updates.** The Designer activates this port property for lookup/output ports when you configure the Lookup transformation to use a dynamic cache. Select this property when you do not want the Integration Service to update the column in the cache when the data in this column contains a null value. For more information, see “Using the Ignore Null Property” on page 353.
- ◆ **Ignore in Comparison.** The Designer activates this port property for lookup/output ports not used in the lookup condition when you configure the Lookup transformation to use a dynamic cache. The Integration Service compares the values in all lookup ports with the values in their associated input ports by default. Select this property if you want the Integration Service to ignore the port when it compares values before updating a row. For more information, see “Using the Ignore in Comparison Property” on page 354.

Figure 15-4 shows the output port properties unique to a dynamic Lookup transformation:

Figure 15-4. Dynamic Lookup Transformation Ports Tab



Using the NewLookupRow Port

When you define a Lookup transformation to use a dynamic cache, the Designer adds the NewLookupRow port to the transformation. The Integration Service assigns a value to the port, depending on the action it performs to the lookup cache.

Table 15-3 lists the possible NewLookupRow values:

Table 15-3. NewLookupRow Values

NewLookupRow Value	Description
0	Integration Service does not update or insert the row in the cache.
1	Integration Service inserts the row into the cache.
2	Integration Service updates the row in the cache.

When the Integration Service reads a row, it changes the lookup cache depending on the results of the lookup query and the Lookup transformation properties you define. It assigns the value 0, 1, or 2 to the NewLookupRow port to indicate if it inserts or updates the row in the cache, or makes no change.

For information about how the Integration Service determines to update the cache, see “Updating the Dynamic Lookup Cache” on page 356.

The NewLookupRow value indicates how the Integration Service changes the lookup cache. It does not change the row type. Therefore, use a Filter or Router transformation and an Update Strategy transformation to help keep the target table and lookup cache synchronized.

Configure the Filter transformation to pass new and updated rows to the Update Strategy transformation before passing them to the cached target. Use the Update Strategy transformation to change the row type of each row to insert or update, depending on the NewLookupRow value.

You can drop the rows that do not change the cache, or you can pass them to another target. For more information, see “Using Update Strategy Transformations with a Dynamic Cache” on page 354.

Define the filter condition in the Filter transformation based on the value of NewLookupRow. For example, use the following condition to pass both inserted and updated rows to the cached target:

```
NewLookupRow != 0
```

For more information about the Filter transformation, see “Filter Transformation” on page 189.

Using the Associated Input Port

When you use a dynamic lookup cache, you must associate each lookup/output port with an input/output port or a sequence ID. The Integration Service uses the data in the associated port to insert or update rows in the lookup cache. The Designer associates the input/output ports with the lookup/output ports used in the lookup condition.

For more information about the values of a Lookup transformation when you use a dynamic lookup cache, see “Working with Lookup Transformation Values” on page 349.

Sometimes you need to create a generated key for a column in a target table. For lookup ports with an Integer or Small Integer datatype, you can associate a generated key instead of an input port. To do this, select *Sequence-ID* in the Associated Port column.

When you select Sequence-ID in the Associated Port column, the Integration Service generates a key when it inserts a row into the lookup cache.

The Integration Service uses the following process to generate sequence IDs:

1. When the Integration Service creates the dynamic lookup cache, it tracks the range of values in the cache associated with any port using a sequence ID.
2. When the Integration Service inserts a new row of data into the cache, it generates a key for a port by incrementing the greatest sequence ID existing value by one.
3. When the Integration Service reaches the maximum number for a generated sequence ID, it starts over at one. It then increments each sequence ID by one until it reaches the smallest existing value minus one. If the Integration Service runs out of unique sequence ID numbers, the session fails.

Note: The maximum value for a sequence ID is 2147483647.

The Integration Service only generates a sequence ID for rows it inserts into the cache.

Working with Lookup Transformation Values

When you associate an input/output port or a sequence ID with a lookup/output port, the following values match by default:

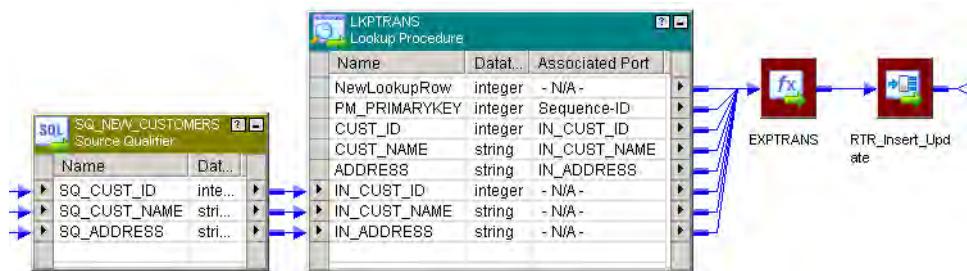
- ◆ **Input value.** Value the Integration Service passes into the transformation.
- ◆ **Lookup value.** Value that the Integration Service inserts into the cache.
- ◆ **Input/output port output value.** Value that the Integration Service passes out of the input/output port.

The lookup/output port output value depends on whether you choose to output old or new values when the Integration Service updates a row:

- ◆ **Output old values on update.** The Integration Service outputs the value that existed in the cache before it updated the row.
- ◆ **Output new values on update.** The Integration Service outputs the updated value that it writes in the cache. The lookup/output port value matches the input/output port value.

Note: You configure to output old or new values using the Output Old Value On Update transformation property. For more information about this property, see “Lookup Properties” on page 316.

For example, you have the following Lookup transformation that uses a dynamic lookup cache:



You define the following lookup condition:

```
IN_CUST_ID = CUST_ID
```

By default, the row type of all rows entering the Lookup transformation is insert. To perform both inserts and updates in the cache and target table, you select the Insert Else Update property in the Lookup transformation.

The following sections describe the values of the rows in the cache, the input rows, lookup rows, and output rows as you run the session.

Initial Cache Values

When you run the session, the Integration Service builds the lookup cache from the target table with the following data:

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100001	80001	Marion James	100 Main St.
100002	80002	Laura Jones	510 Broadway Ave.
100003	80003	Shelley Lau	220 Burnside Ave.

Input Values

The source contains rows that exist and rows that do not exist in the target table. The following rows pass into the Lookup transformation from the Source Qualifier transformation:

SQ_CUST_ID	SQ_CUST_NAME	SQ_ADDRESS
80001	Marion Atkins	100 Main St.
80002	Laura Gomez	510 Broadway Ave.
99001	Jon Freeman	555 6th Ave.

Note: The input values always match the values the Integration Service outputs out of the input/output ports.

Lookup Values

The Integration Service looks up values in the cache based on the lookup condition. It updates rows in the cache for existing customer IDs 80001 and 80002. It inserts a row into the cache for customer ID 99001. The Integration Service generates a new key (PK_PRIMARYKEY) for the new row.

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100001	80001	Marion Atkins	100 Main St.
100002	80002	Laura Gomez	510 Broadway Ave.
100004	99001	Jon Freeman	555 6th Ave.

Output Values

The Integration Service flags the rows in the Lookup transformation based on the inserts and updates it performs on the dynamic cache. These rows pass through an Expression transformation to a Router transformation that filters and passes on the inserted and updated rows to an Update Strategy transformation. The Update Strategy transformation flags the rows based on the value of the NewLookupRow port.

The output values of the lookup/output and input/output ports depend on whether you choose to output old or new values when the Integration Service updates a row. However, the output values of the NewLookupRow port and any lookup/output port that uses the Sequence-ID is the same for new and updated rows.

When you choose to output *new* values, the lookup/output ports output the following values:

NewLookupRow	PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
2	100001	80001	Marion Atkins	100 Main St.
2	100002	80002	Laura Gomez	510 Broadway Ave.
1	100004	99001	Jon Freeman	555 6th Ave.

When you choose to output *old* values, the lookup/output ports output the following values:

NewLookupRow	PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
2	100001	80001	Marion James	100 Main St.
2	100002	80002	Laura Jones	510 Broadway Ave.
1	100004	99001	Jon Freeman	555 6th Ave.

Note that when the Integration Service updates existing rows in the lookup cache and when it passes rows to the lookup/output ports, it always uses the existing primary key (PK_PRIMARYKEY) values for rows that exist in the cache and target table.

The Integration Service uses the sequence ID to generate a new primary key for the customer that it does not find in the cache. The Integration Service inserts the new primary key value into the lookup cache and outputs it to the lookup/output port.

The Integration Service output values from the input/output ports that match the input values. For those values, see “Input Values” on page 350.

Note: If the input value is NULL and you select the Ignore Null property for the associated input port, the input value does not equal the lookup value or the value out of the input/output port. When you select the Ignore Null property, the lookup cache and the target table might become unsynchronized if you pass null values to the target. You must verify that you do not pass null values to the target. For more information, see “Using the Ignore Null Property” on page 353.

Using the Ignore Null Property

When you update a dynamic lookup cache and target table, the source data might contain some null values. The Integration Service can handle the null values in the following ways:

- ◆ **Insert null values.** The Integration Service uses null values from the source and updates the lookup cache and target table using all values from the source.
- ◆ **Ignore null values.** The Integration Service ignores the null values in the source and updates the lookup cache and target table using only the not null values from the source.

If you know the source data contains null values, and you do not want the Integration Service to update the lookup cache or target with null values, select the Ignore Null property for the corresponding lookup/output port.

For example, you want to update the master customer table. The source contains new customers and current customers whose last names have changed. The source contains the customer IDs and names of customers whose names have changed, but it contains null values for the address columns. You want to insert new customers and update the current customer names while retaining the current address information in a master customer table.

For example, the master customer table contains the following data:

PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100001	80001	Marion James	100 Main St.	Mt. View	CA	94040
100002	80002	Laura Jones	510 Broadway Ave.	Raleigh	NC	27601
100003	80003	Shelley Lau	220 Burnside Ave.	Portland	OR	97210

The source contains the following data:

CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
80001	Marion Atkins	NULL	NULL	NULL	NULL
80002	Laura Gomez	NULL	NULL	NULL	NULL
99001	Jon Freeman	555 6th Ave.	San Jose	CA	95051

Select Insert Else Update in the Lookup transformation in the mapping. Select the Ignore Null option for all lookup/output ports in the Lookup transformation. When you run a session, the Integration Service ignores null values in the source data and updates the lookup cache and the target table with not null values:

PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100001	80001	Marion Atkins	100 Main St.	Mt. View	CA	94040
100002	80002	Laura Gomez	510 Broadway Ave.	Raleigh	NC	27601
100003	80003	Shelley Lau	220 Burnside Ave.	Portland	OR	97210
100004	99001	Jon Freeman	555 6th Ave.	San Jose	CA	95051

Note: When you choose to ignore NULLs, you must verify that you output the same values to the target that the Integration Service writes to the lookup cache. When you choose to ignore

NULLs, the lookup cache and the target table might become unsynchronized if you pass null input values to the target. Configure the mapping based on the value you want the Integration Service to output from the lookup/output ports when it updates a row in the cache:

- ◆ **New values.** Connect only lookup/output ports from the Lookup transformation to the target.
- ◆ **Old values.** Add an Expression transformation after the Lookup transformation and before the Filter or Router transformation. Add output ports in the Expression transformation for each port in the target table and create expressions to ensure you do not output null input values to the target.

Using the Ignore in Comparison Property

When you run a session that uses a dynamic lookup cache, the Integration Service compares the values in all lookup ports with the values in their associated input ports by default. It compares the values to determine whether or not to update the row in the lookup cache. When a value in an input port differs from the value in the lookup port, the Integration Service updates the row in the cache.

If you do not want to compare all ports, you can choose the ports you want the Integration Service to ignore when it compares ports. The Designer only enables this property for lookup/output ports when the port is not used in the lookup condition. You can improve performance by ignoring some ports during comparison.

You might want to do this when the source data includes a column that indicates whether or not the row contains data you need to update. Select the Ignore in Comparison property for all lookup ports except the port that indicates whether or not to update the row in the cache and target table.

Note: You must configure the Lookup transformation to compare at least one port. The Integration Service fails the session when you ignore all ports.

Using Update Strategy Transformations with a Dynamic Cache

When you use a dynamic lookup cache, use Update Strategy transformations to define the row type for the following rows:

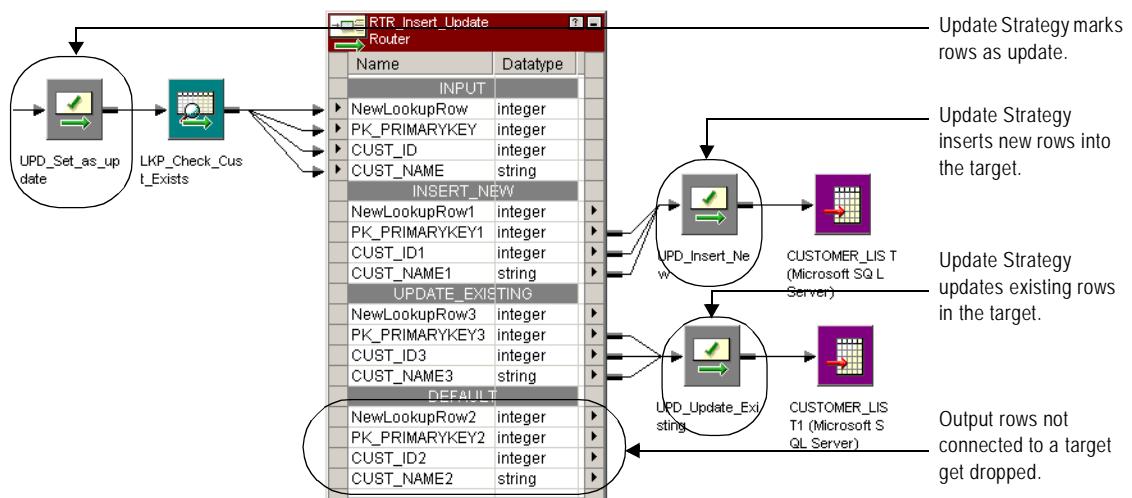
- ◆ **Rows entering the Lookup transformation.** By default, the row type of all rows entering a Lookup transformation is insert. However, use an Update Strategy transformation before a Lookup transformation to define all rows as update, or some as update and some as insert.
- ◆ **Rows leaving the Lookup transformation.** The NewLookupRow value indicates how the Integration Service changed the lookup cache, but it does not change the row type. Use a Filter or Router transformation after the Lookup transformation to direct rows leaving the Lookup transformation based on the NewLookupRow value. Use Update Strategy transformations after the Filter or Router transformation to flag rows for insert or update before the target definition in the mapping.

Note: If you want to drop the unchanged rows, do not connect rows from the Filter or Router transformation with the NewLookupRow equal to 0 to the target definition.

When you define the row type as insert for rows entering a Lookup transformation, use the Insert Else Update property in the Lookup transformation. When you define the row type as update for rows entering a Lookup transformation, use the Update Else Insert property in the Lookup transformation. If you define some rows entering a Lookup transformation as update and some as insert, use either the Update Else Insert or Insert Else Update property, or use both properties. For more information, see “Updating the Dynamic Lookup Cache” on page 356.

Figure 15-5 shows a mapping with multiple Update Strategy transformations and a Lookup transformation using a dynamic cache:

Figure 15-5. Using Update Strategy Transformations with a Lookup Transformation



In this case, the Update Strategy transformation before the Lookup transformation flags all rows as update. Select the Update Else Insert property in the Lookup transformation. The Router transformation sends the inserted rows to the Insert_New Update Strategy transformation and sends the updated rows to the Update_Existing Update Strategy transformation. The two Update Strategy transformations to the right of the Lookup transformation flag the rows for insert or update for the target.

Configuring Sessions with a Dynamic Lookup Cache

When you configure a session using Update Strategy transformations and a dynamic lookup cache, you must define certain session properties.

On the General Options settings on the Properties tab in the session properties, define the Treat Source Rows As option as Data Driven.

You must also define the following update strategy target table options:

- ◆ Select Insert
- ◆ Select Update as Update
- ◆ Do not select Delete

These update strategy target table options ensure that the Integration Service updates rows marked for update and inserts rows marked for insert.

If you do not choose Data Driven, the Integration Service flags all rows for the row type you specify in the Treat Source Rows As option and does not use the Update Strategy transformations in the mapping to flag the rows. The Integration Service does not insert and update the correct rows. If you do not choose Update as Update, the Integration Service does not correctly update the rows flagged for update in the target table. As a result, the lookup cache and target table might become unsynchronized. For more information, see “Setting the Update Strategy for a Session” on page 544.

For more information about configuring target session properties, see “Working with Targets” in the *Workflow Administration Guide*.

Updating the Dynamic Lookup Cache

When you use a dynamic lookup cache, define the row type of the rows entering the Lookup transformation as either insert or update. You can define some rows as insert and some as update, or all insert, or all update. By default, the row type of all rows entering a Lookup transformation is insert. You can add an Update Strategy transformation before the Lookup transformation to define the row type as update. For more information, see “Using Update Strategy Transformations with a Dynamic Cache” on page 354.

The Integration Service either inserts or updates rows in the cache, or does not change the cache. The row type of the rows entering the Lookup transformation and the lookup query result affect how the Integration Service updates the cache. However, you must also configure the following Lookup properties to determine how the Integration Service updates the lookup cache:

- ◆ **Insert Else Update.** Applies to rows entering the Lookup transformation with the row type of *insert*.
- ◆ **Update Else Insert.** Applies to rows entering the Lookup transformation with the row type of *update*.

Note: You can select either the Insert Else Update or Update Else Insert property, or you can select both properties or neither property. The Insert Else Update property only affects rows entering the Lookup transformation with the row type of *insert*. The Update Else Insert property only affects rows entering the Lookup transformation with the row type of *update*.

Insert Else Update

You can select the Insert Else Update property in the Lookup transformation. This property only applies to rows entering the Lookup transformation with the row type of *insert*. When a

row of any other row type, such as update, enters the Lookup transformation, the Insert Else Update property has no effect on how the Integration Service handles the row.

When you select Insert Else Update and the row type entering the Lookup transformation is insert, the Integration Service inserts the row into the cache if it is new. If the row exists in the index cache but the data cache is different than the current row, the Integration Service updates the row in the data cache.

If you do not select Insert Else Update and the row type entering the Lookup transformation is insert, the Integration Service inserts the row into the cache if it is new, and makes no change to the cache if the row exists.

Table 15-4 describes how the Integration Service changes the lookup cache when the row type of the rows entering the Lookup transformation is insert:

Table 15-4. Dynamic Lookup Cache Behavior for Insert Row Type

Insert Else Update Option	Row Found in Cache	Data Cache is Different	Lookup Cache Result	NewLookupRow Value
Cleared (insert only)	Yes	n/a	No change	0
	No	n/a	Insert	1
Selected	Yes	Yes	Update	2*
	Yes	No	No change	0
	No	n/a	Insert	1

*If you select *Ignore Null* for all lookup ports not in the lookup condition and if all those ports contain null values, the Integration Service does not change the cache and the NewLookupRow value equals 0. For more information, see “Using the *Ignore Null* Property” on page 353.

Update Else Insert

You can select the Update Else Insert property in the Lookup transformation. This property only applies to rows entering the Lookup transformation with the row type of *update*. When a row of any other row type, such as insert, enters the Lookup transformation, this property has no effect on how the Integration Service handles the row.

When you select this property and the row type entering the Lookup transformation is update, the Integration Service updates the row in the cache if the row exists in the index cache and the cache data is different than the existing row. The Integration Service inserts the row in the cache if it is new.

If you do not select this property and the row type entering the Lookup transformation is update, the Integration Service updates the row in the cache if it exists, and makes no change to the cache if the row is new.

Table 15-5 describes how the Integration Service changes the lookup cache when the row type of the rows entering the Lookup transformation is update:

Table 15-5. Dynamic Lookup Cache Behavior for Update Row Type

Update Else Insert Option	Row Found in Cache	Data Cache is Different	Lookup Cache Result	NewLookupRow Value
Cleared (update only)	Yes	Yes	Update	2*
	Yes	No	No change	0
	No	n/a	No change	0
Selected	Yes	Yes	Update	2*
	Yes	No	No change	0
	No	n/a	Insert	1

*If you select Ignore Null for all lookup ports not in the lookup condition and if all those ports contain null values, the Integration Service does not change the cache and the NewLookupRow value equals 0. For more information, see “Using the Ignore Null Property” on page 353.

Using the WHERE Clause with a Dynamic Cache

When you add a WHERE clause in a lookup SQL override, the Integration Service uses the WHERE clause to build the cache from the database and to perform a lookup on the database table for an uncached lookup. However, it does not use the WHERE clause to insert rows into a dynamic cache when it runs a session.

When you add a WHERE clause in a Lookup transformation using a dynamic cache, connect a Filter transformation before the Lookup transformation to filter rows you do not want to insert into the cache or target table. If you do not use a Filter transformation, you might get inconsistent data.

For example, you configure a Lookup transformation to perform a dynamic lookup on the employee table, EMP, matching rows by EMP_ID. You define the following lookup SQL override:

```
SELECT EMP_ID, EMP_STATUS FROM EMP ORDER BY EMP_ID, EMP_STATUS WHERE  
EMP_STATUS = 4
```

When you first run the session, the Integration Service builds the lookup cache from the target table based on the lookup SQL override. Therefore, all rows in the cache match the condition in the WHERE clause, `EMP_STATUS = 4`.

Suppose the Integration Service reads a source row that meets the lookup condition you specify (the value for EMP_ID is found in the cache), but the value of EMP_STATUS is 2. The Integration Service does not find the row in the cache, so it inserts the row into the cache and passes the row to the target table. When this happens, not all rows in the cache match the condition in the WHERE clause. When the Integration Service tries to insert this row in the target table, you might get inconsistent data if the row already exists there.

To verify that you only insert rows into the cache that match the WHERE clause, add a Filter transformation before the Lookup transformation and define the filter condition as the condition in the WHERE clause in the lookup SQL override.

For the example above, enter the following filter condition:

```
EMP_STATUS = 4
```

For more information about the lookup SQL override, see “Overriding the Lookup Query” on page 324.

Synchronizing the Dynamic Lookup Cache

When you use a dynamic lookup cache, the Integration Service writes to the lookup cache before it writes to the target table. The lookup cache and target table can become unsynchronized if the Integration Service does not write the data to the target. For example, the target database or Informatica writer might reject the data.

Use the following guidelines to keep the lookup cache synchronized with the lookup table:

- ◆ Use a Router transformation to pass rows to the cached target when the NewLookupRow value equals one or two. Use the Router transformation to drop rows when the NewLookupRow value equals zero, or you can output those rows to a different target.
- ◆ Use Update Strategy transformations after the Lookup transformation to flag rows for insert or update into the target.
- ◆ Set the error threshold to one when you run a session. When you set the error threshold to one, the session fails when it encounters the first error. The Integration Service does not write the new cache files to disk. Instead, it restores the original cache files, if they exist. You must also restore the pre-session target table to the target database. For more information about setting the error threshold, see “Working with Sessions” in the *Workflow Administration Guide*.
- ◆ Verify that you output the same values to the target that the Integration Service writes to the lookup cache. When you choose to output *new* values on update, only connect lookup/output ports to the target table instead of input/output ports. When you choose to output *old* values on update, add an Expression transformation after the Lookup transformation and before the Router transformation. Add output ports in the Expression transformation for each port in the target table and create expressions to ensure you do not output null input values to the target.
- ◆ Set the Treat Source Rows As property to Data Driven in the session properties.
- ◆ Select Insert and Update as Update when you define the update strategy target table options in the session properties. This ensures that the Integration Service updates rows marked for update and inserts rows marked for insert. Select these options in the Transformations View on the Mapping tab in the session properties. For more information, see “Working with Targets” in the *Workflow Administration Guide*.

Null Values in Lookup Condition Columns

Sometimes when you run a session, the source data may contain null values in columns used in the lookup condition. The Integration Service handles rows with null values in lookup condition columns differently, depending on whether the row exists in the cache:

- ♦ If the row *does not* exist in the lookup cache, the Integration Service inserts the row in the cache and passes it to the target table.
- ♦ If the row *does* exist in the lookup cache, the Integration Service does not update the row in the cache or target table.

Note: If the source data contains null values in the lookup condition columns, set the error threshold to one. This ensures that the lookup cache and table remain synchronized if the Integration Service inserts a row in the cache, but the database rejects the row due to a Not Null constraint.

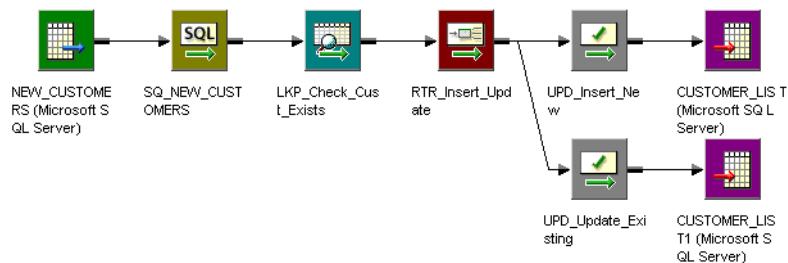
Example Using a Dynamic Lookup Cache

Use a dynamic lookup cache when you need to insert and update rows in the target. When you use a dynamic lookup cache, you can insert and update the cache with the same data you pass to the target to insert and update.

For example, use a dynamic lookup cache to update a table that contains customer data. The source data contains rows that you need to insert into the target and rows you need to update in the target.

Figure 15-6 shows a mapping that uses a dynamic cache:

Figure 15-6. Slowly Changing Dimension Mapping with Dynamic Lookup Cache



The Lookup transformation uses a dynamic lookup cache. When the session starts, the Integration Service builds the lookup cache from the target table. When the Integration Service reads a row that is not in the lookup cache, it inserts the row in the cache and then passes the row out of the Lookup transformation. The Router transformation directs the row to the UPD_Insert_New Update Strategy transformation. The Update Strategy transformation marks the row as insert before passing it to the target.

The target table changes as the session runs, and the Integration Service inserts new rows and updates existing rows in the lookup cache. The Integration Service keeps the lookup cache and target table synchronized.

To generate keys for the target, use Sequence-ID in the associated port. The sequence ID generates primary keys for new rows the Integration Service inserts into the target table.

Without the dynamic lookup cache, you need to use two Lookup transformations in the mapping. Use the first Lookup transformation to insert rows in the target. Use the second Lookup transformation to recache the target table and update rows in the target table.

You increase session performance when you use a dynamic lookup cache because you only need to build the cache from the database once. You can continue to use the lookup cache even though the data in the target table changes.

Rules and Guidelines for Dynamic Caches

Use the following guidelines when you use a dynamic lookup cache:

- ◆ You can create a dynamic lookup cache from a relational or flat file source.
- ◆ The Lookup transformation must be a connected transformation.
- ◆ Use a persistent or a non-persistent cache.
- ◆ If the dynamic cache is not persistent, the Integration Service always rebuilds the cache from the database, even if you do not enable Recache from Lookup Source.
- ◆ You cannot share the cache between a dynamic Lookup transformation and static Lookup transformation in the same target load order group.
- ◆ You can only create an equality lookup condition. You cannot look up a range of data.
- ◆ Associate each lookup port (that is not in the lookup condition) with an input port or a sequence ID.
- ◆ Use a Router transformation to pass rows to the cached target when the NewLookupRow value equals one or two. Use the Router transformation to drop rows when the NewLookupRow value equals zero, or you can output those rows to a different target.
- ◆ Verify that you output the same values to the target that the Integration Service writes to the lookup cache. When you choose to output *new* values on update, only connect lookup/output ports to the target table instead of input/output ports. When you choose to output *old* values on update, add an Expression transformation after the Lookup transformation and before the Router transformation. Add output ports in the Expression transformation for each port in the target table and create expressions to ensure you do not output null input values to the target.
- ◆ When you use a lookup SQL override, make sure you map the correct columns to the appropriate targets for lookup.
- ◆ When you add a WHERE clause to the lookup SQL override, use a Filter transformation before the Lookup transformation. This ensures the Integration Service only inserts rows in the dynamic cache and target table that match the WHERE clause. For more information, see “Using the WHERE Clause with a Dynamic Cache” on page 358.
- ◆ When you configure a reusable Lookup transformation to use a dynamic cache, you cannot edit the condition or disable the Dynamic Lookup Cache property in a mapping.

- ◆ Use Update Strategy transformations after the Lookup transformation to flag the rows for insert or update for the target.
- ◆ Use an Update Strategy transformation before the Lookup transformation to define some or all rows as update if you want to use the Update Else Insert property in the Lookup transformation.
- ◆ Set the row type to Data Driven in the session properties.
- ◆ Select Insert and Update as Update for the target table options in the session properties.

Sharing the Lookup Cache

You can configure multiple Lookup transformations in a mapping to share a single lookup cache. The Integration Service builds the cache when it processes the first Lookup transformation. It uses the same cache to perform lookups for subsequent Lookup transformations that share the cache.

You can share caches that are unnamed and named:

- ◆ **Unnamed cache.** When Lookup transformations in a mapping have compatible caching structures, the Integration Service shares the cache by default. You can only share static unnamed caches.
- ◆ **Named cache.** Use a persistent named cache when you want to share a cache file across mappings or share a dynamic and a static cache. The caching structures must match or be compatible with a named cache. You can share static and dynamic named caches.

When the Integration Service shares a lookup cache, it writes a message in the session log.

Sharing an Unnamed Lookup Cache

By default, the Integration Service shares the cache for Lookup transformations in a mapping that have compatible caching structures. For example, if you have two instances of the same reusable Lookup transformation in one mapping and you use the same output ports for both instances, the Lookup transformations share the lookup cache by default.

When two Lookup transformations share an unnamed cache, the Integration Service saves the cache for a Lookup transformation and uses it for subsequent Lookup transformations that have the same lookup cache structure.

If the transformation properties or the cache structure do not allow sharing, the Integration Service creates a new cache.

Guidelines for Sharing an Unnamed Lookup Cache

Use the following guidelines when you configure Lookup transformations to share an unnamed cache:

- ◆ You can share static unnamed caches.
- ◆ Shared transformations must use the same ports in the lookup condition. The conditions can use different operators, but the ports must be the same.
- ◆ You must configure some of the transformation properties to enable unnamed cache sharing. For more information, see Table 15-6 on page 364.
- ◆ The structure of the cache for the shared transformations must be compatible.
 - If you use hash auto-keys partitioning, the lookup/output ports for each transformation must match.
 - If you do not use hash auto-keys partitioning, the lookup/output ports for the first shared transformation must match or be a superset of the lookup/output ports for subsequent transformations.

- ♦ If the Lookup transformations with hash auto-keys partitioning are in different target load order groups, you must configure the same number of partitions for each group. If you do not use hash auto-keys partitioning, you can configure a different number of partitions for each target load order group.

Table 15-6 shows when you can share an unnamed static and dynamic cache:

Table 15-6. Location for Sharing Unnamed Cache

Shared Cache	Location of Transformations
Static with Static	Anywhere in the mapping.
Dynamic with Dynamic	Cannot share.
Dynamic with Static	Cannot share.

Table 15-7 describes the guidelines to follow when you configure Lookup transformations to share an unnamed cache:

Table 15-7. Properties for Sharing Unnamed Cache

Properties	Configuration for Unnamed Shared Cache
Lookup SQL Override	If you use the Lookup SQL Override property, you must use the same override in all shared transformations.
Lookup Table Name	Must match.
Lookup Caching Enabled	Must be enabled.
Lookup Policy on Multiple Match	n/a
Lookup Condition	Shared transformations must use the same ports in the lookup condition. The conditions can use different operators, but the ports must be the same.
Connection Information	The connection must be the same. When you configure the sessions, the database connection must match.
Source Type	Must match.
Tracing Level	n/a
Lookup Cache Directory Name	Does not need to match.
Lookup Cache Persistent	Optional. You can share persistent and non-persistent.
Lookup Data Cache Size	Integration Service allocates memory for the first shared transformation in each pipeline stage. It does not allocate additional memory for subsequent shared transformations in the same pipeline stage. For information about pipeline stages, see "Pipeline Partitioning" in the <i>Workflow Administration Guide</i> .
Lookup Index Cache Size	Integration Service allocates memory for the first shared transformation in each pipeline stage. It does not allocate additional memory for subsequent shared transformations in the same pipeline stage. For information about pipeline stages, see "Pipeline Partitioning" in the <i>Workflow Administration Guide</i> .

Table 15-7. Properties for Sharing Unnamed Cache

Properties	Configuration for Unnamed Shared Cache
Dynamic Lookup Cache	You cannot share an unnamed dynamic cache.
Output Old Value On Update	Does not need to match.
Cache File Name Prefix	Do not use. You cannot share a named cache with an unnamed cache.
Recache From Lookup Source	If you configure a Lookup transformation to recache from source, subsequent Lookup transformations in the target load order group can share the existing cache whether or not you configure them to recache from source. If you configure subsequent Lookup transformations to recache from source, the Integration Service shares the cache instead of rebuilding the cache when it processes the subsequent Lookup transformation. If you do not configure the first Lookup transformation in a target load order group to recache from source, and you do configure the subsequent Lookup transformation to recache from source, the transformations cannot share the cache. The Integration Service builds the cache when it processes each Lookup transformation.
Lookup/Output Ports	The lookup/output ports for the second Lookup transformation must match or be a subset of the ports in the transformation that the Integration Service uses to build the cache. The order of the ports do not need to match.
Insert Else Update	n/a
Update Else Insert	n/a
Datetime Format	n/a
Thousand Separator	n/a
Decimal Separator	n/a
Case-Sensitive String Comparison	Must match.
Null Ordering	Must match.
Sorted Input	n/a

Sharing a Named Lookup Cache

You can also share the cache between multiple Lookup transformations by using a persistent lookup cache and naming the cache files. You can share one cache between Lookup transformations in the same mapping or across mappings.

The Integration Service uses the following process to share a named lookup cache:

1. When the Integration Service processes the first Lookup transformation, it searches the cache directory for cache files with the same file name prefix. For more information about the Cache File Name Prefix property, see “Lookup Properties” on page 316.
2. If the Integration Service finds the cache files and you do not specify to recache from source, the Integration Service uses the saved cache files.
3. If the Integration Service does not find the cache files or if you specify to recache from source, the Integration Service builds the lookup cache using the database table.

4. The Integration Service saves the cache files to disk after it processes each target load order group.
5. The Integration Service uses the following rules to process the second Lookup transformation with the same cache file name prefix:
 - ◆ The Integration Service uses the memory cache if the transformations are in the same target load order group.
 - ◆ The Integration Service rebuilds the memory cache from the persisted files if the transformations are in different target load order groups.
 - ◆ The Integration Service rebuilds the cache from the database if you configure the transformation to recache from source and the first transformation is in a different target load order group.
 - ◆ The Integration Service fails the session if you configure subsequent Lookup transformations to recache from source, but not the first one in the same target load order group.
 - ◆ If the cache structures do not match, the Integration Service fails the session.

If you run two sessions simultaneously that share a lookup cache, the Integration Service uses the following rules to share the cache files:

- ◆ The Integration Service processes multiple sessions simultaneously when the Lookup transformations only need to read the cache files.
- ◆ The Integration Service fails the session if one session updates a cache file while another session attempts to read or update the cache file. For example, Lookup transformations update the cache file if they are configured to use a dynamic cache or recache from source.

Guidelines for Sharing a Named Lookup Cache

Use the following guidelines when you configure Lookup transformations to share a named cache:

- ◆ You can share any combination of dynamic and static caches, but you must follow the guidelines for location. For more information, see Table 15-8 on page 367.
- ◆ You must configure some of the transformation properties to enable named cache sharing. For more information, see Table 15-9 on page 367.
- ◆ A dynamic lookup cannot share the cache if the named cache has duplicate rows.
- ◆ A named cache created by a dynamic Lookup transformation with a lookup policy of error on multiple match can be shared by a static or dynamic Lookup transformation with any lookup policy.
- ◆ A named cache created by a dynamic Lookup transformation with a lookup policy of use first or use last can be shared by a Lookup transformation with the same lookup policy.
- ◆ Shared transformations must use the same output ports in the mapping. The criteria and result columns for the cache must match the cache files.

The Integration Service might use the memory cache, or it might build the memory cache from the file, depending on the type and location of the Lookup transformations.

Table 15-8 shows when you can share a static and dynamic named cache:

Table 15-8. Location for Sharing Named Cache

Shared Cache	Location of Transformations	Cache Shared
Static with Static	- Same target load order group. - Separate target load order groups. - Separate mappings.	- Integration Service uses memory cache. - Integration Service uses memory cache. - Integration Service builds memory cache from file.
Dynamic with Dynamic	- Separate target load order groups. - Separate mappings.	- Integration Service uses memory cache. - Integration Service builds memory cache from file.
Dynamic with Static	- Separate target load order groups. - Separate mappings.	- Integration Service builds memory cache from file. - Integration Service builds memory cache from file.

For more information about target load order groups, see “Mappings” in the *Designer Guide*.

Table 15-9 describes the guidelines to follow when you configure Lookup transformations to share a named cache:

Table 15-9. Properties for Sharing Named Cache

Properties	Configuration for Named Shared Cache
Lookup SQL Override	If you use the Lookup SQL Override property, you must use the same override in all shared transformations.
Lookup Table Name	Must match.
Lookup Caching Enabled	Must be enabled.
Lookup Policy on Multiple Match	- A named cache created by a dynamic Lookup transformation with a lookup policy of error on multiple match can be shared by a static or dynamic Lookup transformation with any lookup policy. - A named cache created by a dynamic Lookup transformation with a lookup policy of use first or use last can be shared by a Lookup transformation with the same lookup policy.
Lookup Condition	Shared transformations must use the same ports in the lookup condition. The conditions can use different operators, but the ports must be the same.
Connection Information	The connection must be the same. When you configure the sessions, the database connection must match.
Source Type	Must match.
Tracing Level	n/a
Lookup Cache Directory Name	Must match.
Lookup Cache Persistent	Must be enabled.
Lookup Data Cache Size	When transformations within the same mapping share a cache, the Integration Service allocates memory for the first shared transformation in each pipeline stage. It does not allocate additional memory for subsequent shared transformations in the same pipeline stage. For information about pipeline stages, see “Pipeline Partitioning” in the <i>Workflow Administration Guide</i> .

Table 15-9. Properties for Sharing Named Cache

Properties	Configuration for Named Shared Cache
Lookup Index Cache Size	When transformations within the same mapping share a cache, the Integration Service allocates memory for the first shared transformation in each pipeline stage. It does not allocate additional memory for subsequent shared transformations in the same pipeline stage. For information about pipeline stages, see "Pipeline Partitioning" in the <i>Workflow Administration Guide</i> .
Dynamic Lookup Cache	For more information about sharing static and dynamic cache, see Table 15-8 on page 367.
Output Old Value on Update	Does not need to match.
Cache File Name Prefix	Must match. Enter the prefix only. Do not enter .idx or .dat. You cannot share a named cache with an unnamed cache.
Recache from Source	If you configure a Lookup transformation to recache from source, subsequent Lookup transformations in the target load order group can share the existing cache whether or not you configure them to recache from source. If you configure subsequent Lookup transformations to recache from source, the Integration Service shares the cache instead of rebuilding the cache when it processes the subsequent Lookup transformation. If you do not configure the first Lookup transformation in a target load order group to recache from source, and you do configure the subsequent Lookup transformation to recache from source, the session fails.
Lookup/Output Ports	Lookup/output ports must be identical, but they do not need to be in the same order.
Insert Else Update	n/a
Update Else Insert	n/a
Thousand Separator	n/a
Decimal Separator	n/a
Case-Sensitive String Comparison	n/a
Null Ordering	n/a
Sorted Input	Must match.

Note: You cannot share a lookup cache created on a different operating system. For example, only an Integration Service on UNIX can read a lookup cache created on a Integration Service on UNIX, and only an Integration Service on Windows can read a lookup cache created on an Integration Service on Windows.

Lookup Cache Tips

Use the following tips when you configure the Lookup transformation to cache the lookup table:

- ◆ **Cache small lookup tables.**

Improve session performance by caching small lookup tables. The result of the lookup query and processing is the same, whether or not you cache the lookup table.

- ◆ **Use a persistent lookup cache for static lookup tables.**

If the lookup table does not change between sessions, configure the Lookup transformation to use a persistent lookup cache. The Integration Service then saves and reuses cache files from session to session, eliminating the time required to read the lookup table.

Chapter 16

Normalizer Transformation

This chapter includes the following topics:

- ◆ Overview, 372
- ◆ Normalizing Data in a Mapping, 373
- ◆ Differences Between Normalizer Transformations, 378
- ◆ Troubleshooting, 379

Overview

Transformation type:

Active
Connected

Normalization is the process of organizing data to reduce redundancy. In database terms, this includes creating normalized tables and establishing relationships between the tables eliminating redundancy and inconsistent dependencies.

The Normalizer transformation normalizes records from COBOL and relational sources, allowing you to organize the data. A Normalizer transformation can appear anywhere in a pipeline when you normalize a relational source. Use a Normalizer transformation instead of the Source Qualifier transformation when you normalize a COBOL source. When you drag a COBOL source into the Mapping Designer workspace, the Mapping Designer creates a Normalizer transformation with input and output ports for every column in the source.

You primarily use the Normalizer transformation with COBOL sources, which are often stored in a denormalized format. The OCCURS statement in a COBOL file nests multiple records of information in a single record. Using the Normalizer transformation, you break out repeated data within a record into separate records. For each new record it creates, the Normalizer transformation generates a unique identifier. Use this key value to join the normalized records.

You can also use the Normalizer transformation with relational sources to create multiple rows from a single row of data.

Normalizing Data in a Mapping

Although the Normalizer transformation is designed to handle data read from COBOL sources, you can also use it to denormalize data from any type of source in a mapping. You can add a Normalizer transformation to any data flow within a mapping to normalize components of a single record that contains denormalized data.

If you have denormalized data for which the Normalizer transformation has created key values, connect the ports representing the repeated data and the output port for the generated keys to a different pipeline branch in the mapping. Ultimately, you may want to write these values to different targets.

Use a single Normalizer transformation to handle multiple levels of denormalization in the same record. For example, a single record might contain two different detail record sets.

Rather than using two Normalizer transformations to handle the two different detail record sets, you handle both normalizations in the same transformation.

Normalizer Ports

When you create a Normalizer for a COBOL source, or in the mapping pipeline, the Designer identifies the OCCURS and REDEFINES statements and generates the following columns:

- ◆ **Generated key.** One port for each REDEFINES clause. For more information, see “Generated Key” on page 373.
- ◆ **Generated Column ID.** One port for each OCCURS clause. For more information, see “Generated Column ID” on page 374.

Use these ports for primary and foreign key columns. The Normalizer key and column ID columns are also useful when you want to pivot input columns into rows. You cannot delete these ports.

Generated Key

The Designer generates a port for each REDEFINES clause to specify the generated key. Use the generated key as a primary key column in the target table and to create a primary-foreign key relationship. The naming convention for the Normalizer generated key is:

`GK_<redefined_field_name>`

As shown in Figure 16-1 on page 375, the Designer adds one column (GK_FILE_ONE and GK_HST_AMT) for each REDEFINES in the COBOL source. The Normalizer GK columns tell you the order of records in a REDEFINES clause. For example, if a COBOL file has 10 records, when you run the workflow, the Integration Service numbers the first record 1, the second record 2, and so on.

You can create approximately two billion primary or foreign key values with the Normalizer by connecting the GK port to the transformation or target and using the values ranging from 1 to 2147483647. At the end of each session, the Integration Service updates the GK value to the last value generated for the session plus one.

If you have multiple versions of the Normalizer transformation, the Integration Service updates the GK value across all versions when it runs a session.

If you open the mapping after you run the session, the current value displays the last value generated for the session plus one. Since the Integration Service uses the GK value to determine the first value for each session, you should only edit the GK value if you want to reset the sequence.

If you have multiple versions of the Normalizer, and you want to reset the sequence, you must check in the mapping after you modify the GK value.

Generated Column ID

The Designer generates a port for each OCCURS clause to specify the positional index within an OCCURS clause. Use the generated column ID to create a primary-foreign key relationship. The naming convention for the Normalizer generated column ID is:

`GCID_<occurring_field_name>`

As shown in Figure 16-1 on page 375, the Designer adds one column (GCID_HST_MTH and GCID_HST_AMT) for each OCCURS in the COBOL source. The Normalizer GCID columns tell you the order of records in an OCCURS clause. For example, if a record occurs two times, when you run the workflow, the Integration Service numbers the first record 1 and the second record 2.

Adding a COBOL Source to a Mapping

When you add a COBOL source to a mapping, the Mapping Designer inserts and configures a Normalizer transformation. The Normalizer transformation identifies the nested records within the COBOL source and displays them accordingly.

To add a COBOL source to a mapping:

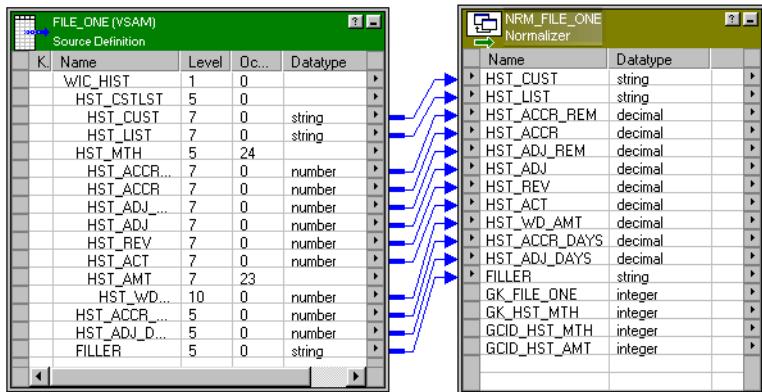
1. In the Designer, create a new mapping or open an existing one.
2. Drag an imported COBOL source definition into the mapping.

If the Designer does not create a Normalizer transformation by default, manually create the Normalizer transformation.

For example, when you add the COBOL source to a mapping, the Designer adds a Normalizer transformation and connects it to the COBOL source definition.

Figure 16-1 shows that the ports representing HST_MTH appear separately within the Normalizer transformation:

Figure 16-1. COBOL Source Definition and a Normalizer Transformation



If you connect the ports directly from the Normalizer transformation to targets, you connect the records from HST_MTH, represented in the Normalizer transformation, to their own target definition, distinct from any other target that may appear in the mapping.

3. Open the new Normalizer transformation.
4. Select the Ports tab and review the ports in the Normalizer transformation.
5. Click the Normalizer tab to review the original organization of the COBOL source.

This tab contains the same information as in the Columns tab of the source definition for this COBOL source. However, you cannot modify the field definitions in the Normalizer transformation. If you need to make modifications, open the source definition in the Source Analyzer.

6. Select the Properties tab and enter the following settings:

Setting	Description
Reset	Resets the generated key value after the session finishes to its original value.
Restart	Restarts the generated key values from 1 every time you run a session.
Tracing level	Determines the amount of information about this transformation that the Integration Service writes to the session log. You can override this tracing level when you configure a session.

7. Click OK.
8. Connect the Normalizer transformation to the rest of the mapping.

If you have denormalized data for which the Normalizer transformation has created key values, connect the ports representing the repeated data and the output port for the generated

keys to a different portion of the data flow in the mapping. Ultimately, you may want to write these values to different targets.

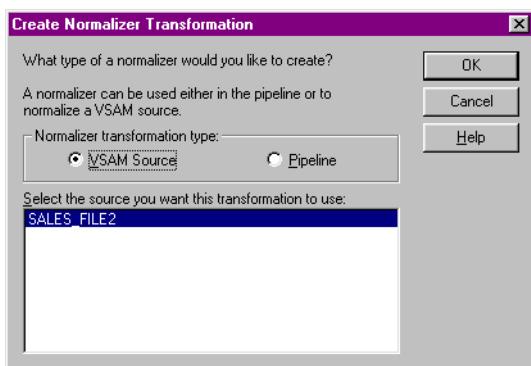
To add a Normalizer transformation to a mapping:

1. In the Mapping Designer, click Transformation > Create. Select Normalizer transformation. Enter a name for the Normalizer transformation. Click Create.

The naming convention for Normalizer transformations is NRM_TransformationName. The Designer creates the Normalizer transformation.

If the mapping contains a COBOL source, and you do not have the option set to create a source qualifier, the Create Normalizer Transformation dialog box appears. For more information about this option, see “Using the Designer” in the *Designer Guide*.

2. If the create Normalizer Transformation dialog box appears, select the Normalizer transformation type.



3. Select the source for this transformation. Click OK.
4. Open the new Normalizer transformation.
5. Select the Normalizer tab and add new output ports.

Add a port corresponding to each column in the source record that contains denormalized data. The new ports only allow the number or string datatypes. You can create only new ports in the Normalizer tab, not the Ports tab.

Using the level controls in the Normalizer transformation, identify which ports belong to the master and detail records. Adjust these ports so that the level setting for detail ports is higher than the level setting for the master record. For example, if ports from the master record are at level 1, the detail ports are at level 2. When you adjust the level setting for the first detail port, the Normalizer transformation creates a heading for the detail record.

Enter the number of times detail records repeat within each master record.

6. After configuring the output ports, click Apply.

The Normalizer transformation creates all the input and output ports needed to connect master and detail records to the rest of the mapping. In addition, the Normalizer

transformation creates a generated key column for joining master and detail records. When you run a session, the Integration Service generates unique IDs for these columns.

7. Select the Properties tab and enter the following settings:

Setting	Description
Reset	Reset generated key sequence values at the end of the session.
Restart	Start the generated key sequence values from 1.
Tracing level	Determines the amount of information Integration Service writes to the session log. You can override this tracing level when you configure a session.

8. Click OK.
9. Connect the Normalizer transformation to the rest of the mapping.
10. Click Repository > Save.

Differences Between Normalizer Transformations

There are a number of differences between a VSAM Normalizer transformation using COBOL sources and a pipeline Normalizer transformation.

Table 16-1 lists the differences between Normalizer transformations:

Table 16-1. VSAM and Relational Normalizer Transformation Differences

	VSAM Normalizer Transformation	Pipeline Normalizer Transformation
Connection	COBOL source	Any transformation
Port creation	Created based on the COBOL source	Created manually
Ni-or-1 rule	Yes	Yes
Transformations allowed before the Normalizer transformation	No	Yes
Transformations allowed after the Normalizer Transformation	Yes	Yes
Reusable	No	Yes
Ports	Input/Output	Input/Output

Note: Concatenation from the Normalizer transformation occurs only when the row sets being concatenated are of the order one. You cannot concatenate row sets in which the order is greater than one.

Troubleshooting

I cannot edit the ports in my Normalizer transformation when using a relational source.

When you create ports manually, you must do so on the Normalizer tab in the transformation, not the Ports tab.

Importing a COBOL file failed with a lot of errors. What should I do?

Check the file heading to see if it follows the COBOL standard, including spaces, tabs, and end of line characters. The header should be similar to the following:

```
identification division.  
    program-id. mead.  
  
environment division.  
    select file-one assign to "fname".  
  
data division.  
    file section.  
        fd FILE-ONE.
```

The import parser does not handle hidden characters or extra spacing very well. Be sure to use a text-only editor to make changes to the COBOL file, such as the DOS edit command. Do not use Notepad or Wordpad.

A session that reads binary data completed, but the information in the target table is incorrect.

Open the session in the Workflow Manager, edit the session, and check the source file format to see if the EBCDIC/ASCII is set correctly. The number of bytes to skip between records must be set to 0.

I have a COBOL field description that uses a non-IBM COMP type. How should I import the source?

In the source definition, clear the IBM COMP option.

In my mapping, I use one Expression transformation and one Lookup transformation to modify two output ports from the Normalizer transformation. The mapping concatenates them into one single transformation. All the ports are under the same level, which does not violate the Ni-or-1 rule. When I check the data loaded in the target, it is incorrect. Why is that?

You can only concatenate ports from level one. Remove the concatenation.

Chapter 17

Rank Transformation

This chapter includes the following topics:

- ◆ Overview, 382
- ◆ Ports in a Rank Transformation, 384
- ◆ Defining Groups, 385
- ◆ Creating a Rank Transformation, 386

Overview

Transformation type:

Active
Connected

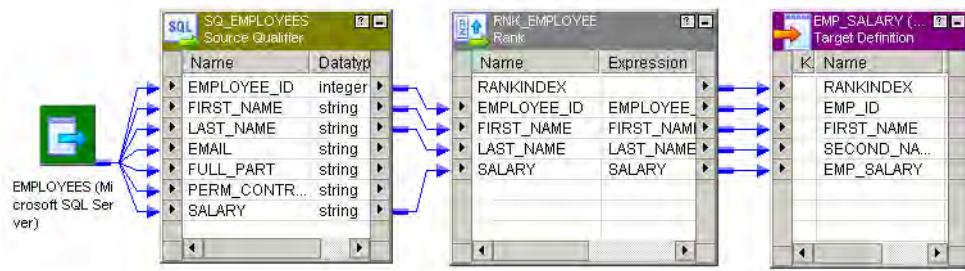
You can select only the top or bottom rank of data with Rank transformation. Use a Rank transformation to return the largest or smallest numeric value in a port or group. You can also use a Rank transformation to return the strings at the top or the bottom of a session sort order. During the session, the Integration Service caches input data until it can perform the rank calculations.

The Rank transformation differs from the transformation functions MAX and MIN, in that it lets you select a group of top or bottom values, not just one value. For example, use Rank to select the top 10 salespersons in a given territory. Or, to generate a financial report, you might also use a Rank transformation to identify the three departments with the lowest expenses in salaries and overhead. While the SQL language provides many functions designed to handle groups of data, identifying top or bottom strata within a set of rows is not possible using standard SQL functions.

You connect all ports representing the same row set to the transformation. Only the rows that fall within that rank, based on some measure you set when you configure the transformation, pass through the Rank transformation. You can also write expressions to transform data or perform calculations.

Figure 17-1 shows a mapping that passes employee data from a human resources table through a Rank transformation. The Rank only passes the rows for the top 10 highest paid employees to the next transformation.

Figure 17-1. Sample Mapping with a Rank Transformation



As an active transformation, the Rank transformation might change the number of rows passed through it. You might pass 100 rows to the Rank transformation, but select to rank only the top 10 rows, which pass from the Rank transformation to another transformation.

You can connect ports from only one transformation to the Rank transformation. You can also create local variables and write non-aggregate expressions.

Ranking String Values

When the Integration Service runs in the ASCII data movement mode, it sorts session data using a binary sort order.

When the Integration Service runs in Unicode data movement mode, the Integration Service uses the sort order configured for the session. You select the session sort order in the session properties. The session properties lists all available sort orders based on the code page used by the Integration Service.

For example, you have a Rank transformation configured to return the top three values of a string port. When you configure the workflow, you select the Integration Service on which you want the workflow to run. The session properties display all sort orders associated with the code page of the selected Integration Service, such as French, German, and Binary. If you configure the session to use a binary sort order, the Integration Service calculates the binary value of each string, and returns the three rows with the highest binary values for the string.

Rank Caches

During a session, the Integration Service compares an input row with rows in the data cache. If the input row out-ranks a cached row, the Integration Service replaces the cached row with the input row. If you configure the Rank transformation to rank across multiple groups, the Integration Service ranks incrementally for each group it finds.

The Integration Service stores group information in an index cache and row data in a data cache. If you create multiple partitions in a pipeline, the Integration Service creates separate caches for each partition. For more information about caching, see “Session Caches” in the *Workflow Administration Guide*.

Rank Transformation Properties

When you create a Rank transformation, you can configure the following properties:

- ◆ Enter a cache directory.
- ◆ Select the top or bottom rank.
- ◆ Select the input/output port that contains values used to determine the rank. You can select only one port to define a rank.
- ◆ Select the number of rows falling within a rank.
- ◆ Define groups for ranks, such as the 10 least expensive products for each manufacturer.

Ports in a Rank Transformation

The Rank transformation includes input or input/output ports connected to another transformation in the mapping. It also includes variable ports and a rank port. Use the rank port to specify the column you want to rank.

Table 17-1 lists the ports in a Rank transformation:

Table 17-1. Rank Transformation Ports

Ports	Number Required	Description
I	Minimum of one	Input port. Create an input port to receive data from another transformation.
O	Minimum of one	Output port. Create an output port for each port you want to link to another transformation. You can designate input ports as output ports.
V	Not Required	Variable port. Can use to store values or calculations to use in an expression. Variable ports cannot be input or output ports. They pass data within the transformation only.
R	One only	Rank port. Use to designate the column for which you want to rank values. You can designate only one Rank port in a Rank transformation. The Rank port is an input/output port. You must link the Rank port to another transformation.

Rank Index

The Designer creates a RANKINDEX port for each Rank transformation. The Integration Service uses the Rank Index port to store the ranking position for each row in a group. For example, if you create a Rank transformation that ranks the top five salespersons for each quarter, the rank index numbers the salespeople from 1 to 5:

RANKINDEX	SALES_PERSON	SALES
1	Sam	10,000
2	Mary	9,000
3	Alice	8,000
4	Ron	7,000
5	Alex	6,000

The RANKINDEX is an output port only. You can pass the rank index to another transformation in the mapping or directly to a target.

Defining Groups

Like the Aggregator transformation, the Rank transformation lets you group information. For example, if you want to select the 10 most expensive items by manufacturer, you would first define a group for each manufacturer. When you configure the Rank transformation, you can set one of its input/output ports as a group by port. For each unique value in the group port (for example, MANUFACTURER_ID or MANUFACTURER_NAME), the transformation creates a group of rows falling within the rank definition (top or bottom, and a particular number in each rank).

Therefore, the Rank transformation changes the number of rows in two different ways. By filtering all but the rows falling within a top or bottom rank, you reduce the number of rows that pass through the transformation. By defining groups, you create one set of ranked rows for each group.

For example, you might create a Rank transformation to identify the 50 highest paid employees in the company. In this case, you would identify the SALARY column as the input/output port used to measure the ranks, and configure the transformation to filter out all rows except the top 50.

After the Rank transformation identifies all rows that belong to a top or bottom rank, it then assigns rank index values. In the case of the top 50 employees, measured by salary, the highest paid employee receives a rank index of 1. The next highest-paid employee receives a rank index of 2, and so on. When measuring a bottom rank, such as the 10 lowest priced products in the inventory, the Rank transformation assigns a rank index from lowest to highest. Therefore, the least expensive item would receive a rank index of 1.

If two rank values match, they receive the same value in the rank index and the transformation skips the next value. For example, if you want to see the top five retail stores in the country and two stores have the same sales, the return data might look similar to the following:

RANKINDEX	SALES	STORE
1	10000	Orange
1	10000	Brea
3	90000	Los Angeles
4	80000	Ventura

Creating a Rank Transformation

You can add a Rank transformation anywhere in the mapping after the source qualifier.

To create a Rank transformation:

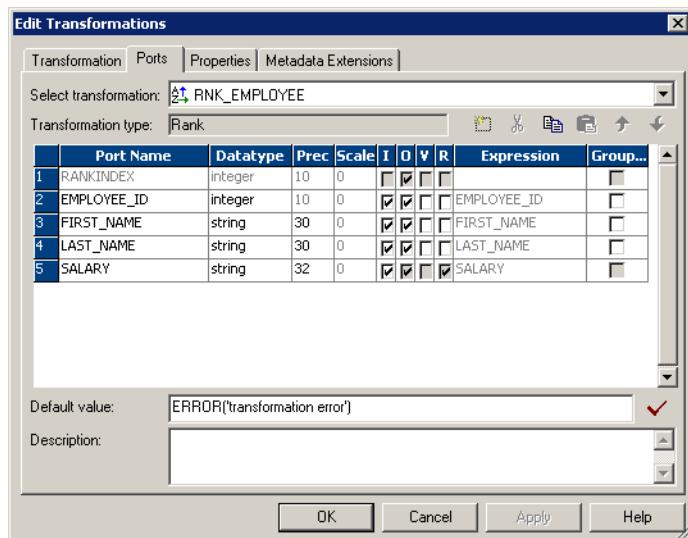
1. In the Mapping Designer, click Transformation > Create. Select the Rank transformation. Enter a name for the Rank. The naming convention for Rank transformations is RNK_*TransformationName*.

Enter a description for the transformation. This description appears in the Repository Manager.

2. Click Create, and then click Done.

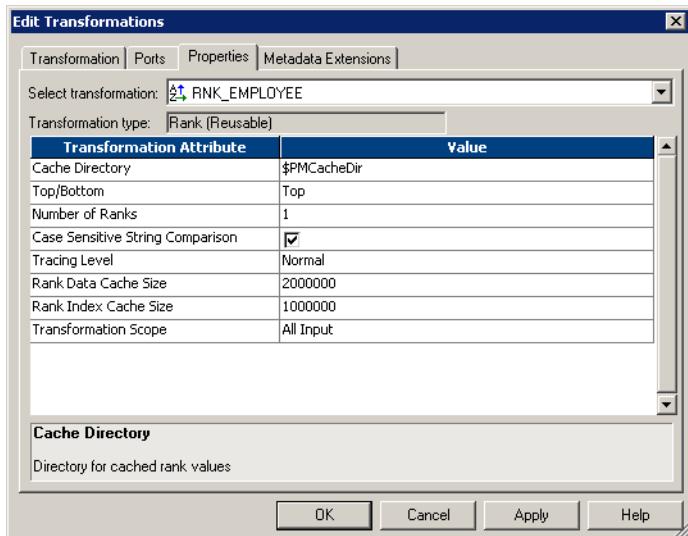
The Designer creates the Rank transformation.

3. Link columns from an input transformation to the Rank transformation.
4. Click the Ports tab, and then select the Rank (R) option for the port used to measure ranks.



If you want to create groups for ranked rows, select Group By for the port that defines the group.

5. Click the Properties tab and select whether you want the top or bottom rank.



6. For the Number of Ranks option, enter the number of rows you want to select for the rank.
7. Change the other Rank transformation properties, if necessary.

Table 17-2 describes the Rank transformation properties:

Table 17-2. Rank Transformation Properties

Setting	Description
Cache Directory	Local directory where the Integration Service creates the index and data cache files. By default, the Integration Service uses the directory entered in the Workflow Manager for the process variable \$PMCacheDir. If you enter a new directory, make sure the directory exists and contains enough disk space for the cache files.
Top/Bottom	Specifies whether you want the top or bottom ranking for a column.
Number of Ranks	Number of rows you want to rank.
Case-Sensitive String Comparison	When running in Unicode mode, the Integration Service ranks strings based on the sort order selected for the session. If the session sort order is case-sensitive, select this option to enable case-sensitive string comparisons, and clear this option to have the Integration Service ignore case for strings. If the sort order is not case-sensitive, the Integration Service ignores this setting. By default, this option is selected.
Tracing Level	Determines the amount of information the Integration Service writes to the session log about data passing through this transformation in a session.

Table 17-2. Rank Transformation Properties

Setting	Description
Rank Data Cache Size	Data cache size for the transformation. Default is 2,000,000 bytes. If the total configured session cache size is 2 GB (2,147,483,648 bytes) or more, you must run the session on a 64-bit Integration Service. You can configure a numeric value, or you can configure the Integration Service to determine the cache size at runtime. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.
Rank Index Cache Size	Index cache size for the transformation. Default is 1,000,000 bytes. If the total configured session cache size is 2 GB (2,147,483,648 bytes) or more, you must run the session on a 64-bit Integration Service. You can configure a numeric value, or you can configure the Integration Service to determine the cache size at runtime. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.
Transformation Scope	<p>Specifies how the Integration Service applies the transformation logic to incoming data:</p> <ul style="list-style-type: none">- Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions.- All Input. Applies the transformation logic on all incoming data. When you choose All Input, the PowerCenter drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source. <p>For more information about transformation scope, see “Understanding Commit Points” in the <i>Workflow Administration Guide</i>.</p>

- 8.** Click OK.
- 9.** Click Repository > Save.

Chapter 18

Router Transformation

This chapter includes the following topics:

- ◆ Overview, 390
- ◆ Working with Groups, 392
- ◆ Working with Ports, 396
- ◆ Connecting Router Transformations in a Mapping, 398
- ◆ Creating a Router Transformation, 400

Overview

Transformation type:

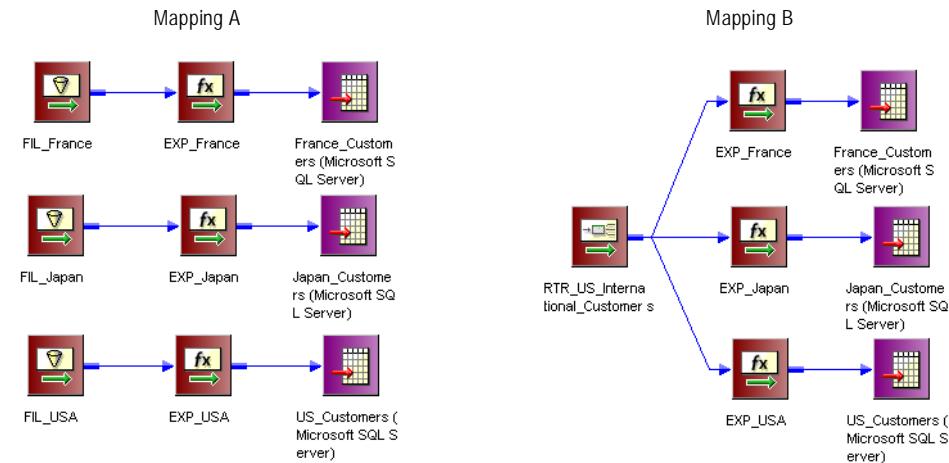
Active
Connected

A Router transformation is similar to a Filter transformation because both transformations allow you to use a condition to test data. A Filter transformation tests data for one condition and drops the rows of data that do not meet the condition. However, a Router transformation tests data for one or more conditions and gives you the option to route rows of data that do not meet any of the conditions to a default output group.

If you need to test the same input data based on multiple conditions, use a Router transformation in a mapping instead of creating multiple Filter transformations to perform the same task. The Router transformation is more efficient. For example, to test data based on three conditions, you only need one Router transformation instead of three filter transformations to perform this task. Likewise, when you use a Router transformation in a mapping, the Integration Service processes the incoming data only once. When you use multiple Filter transformations in a mapping, the Integration Service processes the incoming data for each transformation.

Figure 18-1 shows two mappings that perform the same task. Mapping A uses three Filter transformations while Mapping B produces the same result with one Router transformation:

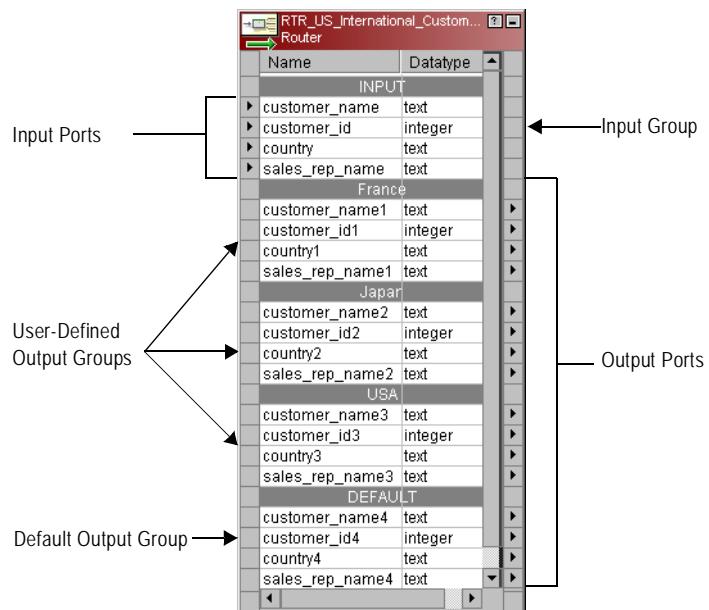
Figure 18-1. Comparing Router and Filter Transformations



A Router transformation consists of input and output groups, input and output ports, group filter conditions, and properties that you configure in the Designer.

Figure 18-2 shows a sample Router transformation and its components:

Figure 18-2. Sample Router Transformation



Working with Groups

A Router transformation has the following types of groups:

- ◆ Input
- ◆ Output

Input Group

The Designer copies property information from the input ports of the input group to create a set of output ports for each output group.

Output Groups

There are two types of output groups:

- ◆ User-defined groups
- ◆ Default group

You cannot modify or delete output ports or their properties.

User-Defined Groups

You create a user-defined group to test a condition based on incoming data. A user-defined group consists of output ports and a group filter condition. You can create and edit user-defined groups on the Groups tab with the Designer. Create one user-defined group for each condition that you want to specify.

The Integration Service uses the condition to evaluate each row of incoming data. It tests the conditions of each user-defined group before processing the default group. The Integration Service determines the order of evaluation for each condition based on the order of the connected output groups. The Integration Service processes user-defined groups that are connected to a transformation or a target in a mapping. The Integration Service only processes user-defined groups that are *not* connected in a mapping if the default group is connected to a transformation or a target.

If a row meets more than one group filter condition, the Integration Service passes this row multiple times.

The Default Group

The Designer creates the default group after you create one new user-defined group. The Designer does not allow you to edit or delete the default group. This group does not have a group filter condition associated with it. If *all* of the conditions evaluate to FALSE, the Integration Service passes the row to the default group. If you want the Integration Service to

drop all rows in the default group, do not connect it to a transformation or a target in a mapping.

The Designer deletes the default group when you delete the last user-defined group from the list.

Using Group Filter Conditions

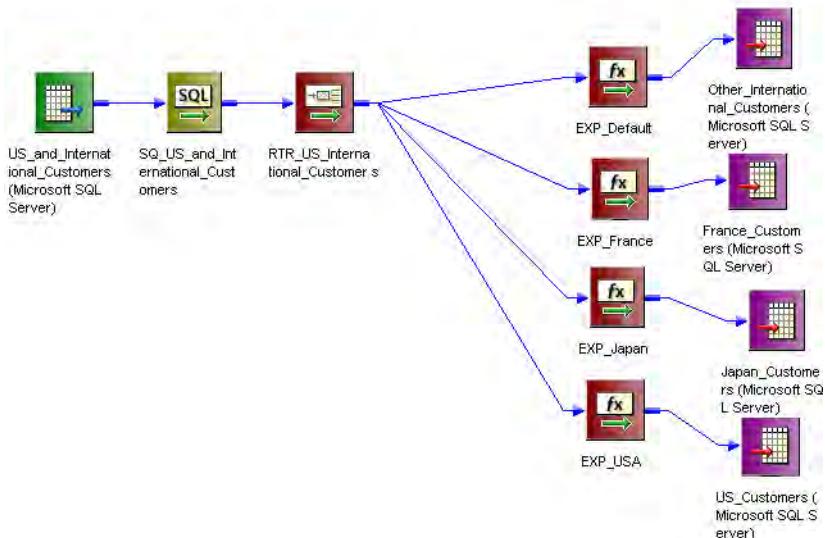
You can test data based on one or more group filter conditions. You create group filter conditions on the Groups tab using the Expression Editor. You can enter any expression that returns a single value. You can also specify a constant for the condition. A group filter condition returns TRUE or FALSE for each row that passes through the transformation, depending on whether a row satisfies the specified condition. Zero (0) is the equivalent of FALSE, and any non-zero value is the equivalent of TRUE. The Integration Service passes the rows of data that evaluate to TRUE to each transformation or target that is associated with each user-defined group.

For example, you have customers from nine countries, and you want to perform different calculations on the data from only three countries. You might want to use a Router transformation in a mapping to filter this data to three different Expression transformations.

There is no group filter condition associated with the default group. However, you can create an Expression transformation to perform a calculation based on the data from the other six countries.

Figure 18-3 shows a mapping with a Router transformation that filters data based on multiple conditions:

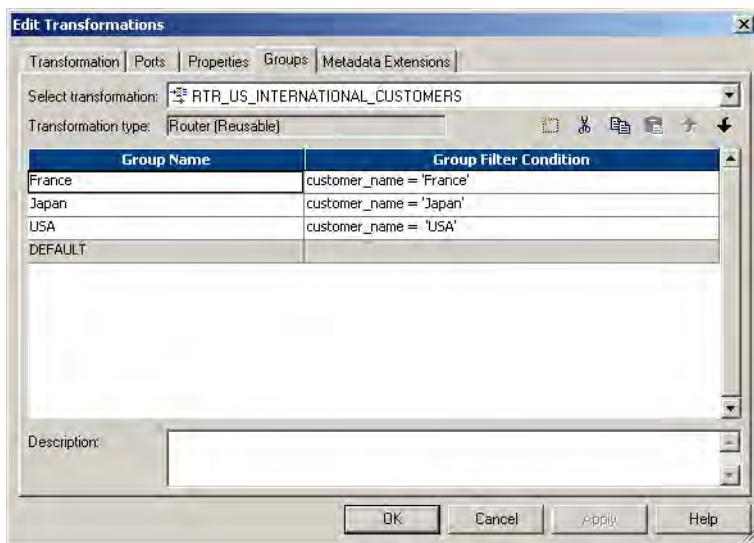
Figure 18-3. Using a Router Transformation in a Mapping



Since you want to perform multiple calculations based on the data from three different countries, create three user-defined groups and specify three group filter conditions on the Groups tab.

Figure 18-4 shows specifying group filter conditions in a Router transformation to filter customer data:

Figure 18-4. Specifying Group Filter Conditions



In the session, the Integration Service passes the rows of data that evaluate to TRUE to each transformation or target that is associated with each user-defined group, such as Japan, France, and USA. The Integration Service passes the row to the default group if *all* of the conditions evaluate to FALSE. If this happens, the Integration Service passes the data of the other six countries to the transformation or target that is associated with the default group. If you want the Integration Service to drop all rows in the default group, do not connect it to a transformation or a target in a mapping.

Adding Groups

Adding a group is similar to adding a port in other transformations. The Designer copies property information from the input ports to the output ports. For more information, see “Working with Groups” on page 392.

To add a group to a Router transformation:

1. Click the Groups tab.
2. Click the Add button.
3. Enter a name for the new group in the Group Name section.
4. Click the Group Filter Condition field and open the Expression Editor.

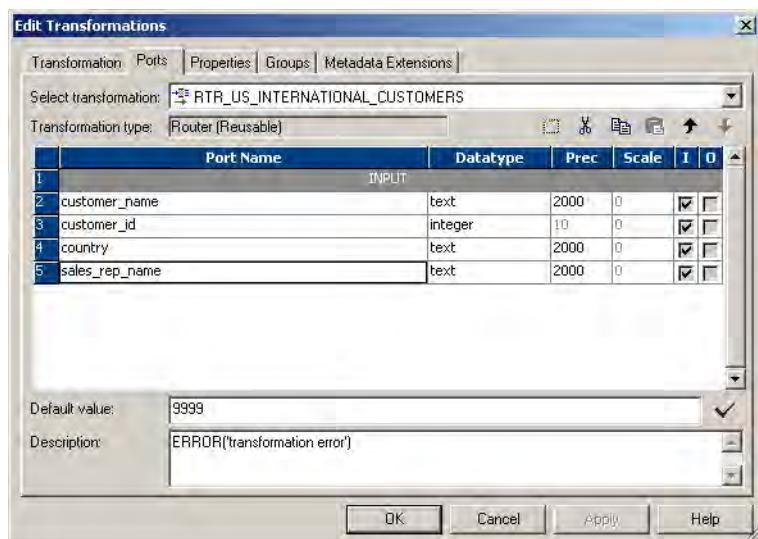
- 5.** Enter the group filter condition.
- 6.** Click Validate to check the syntax of the condition.
- 7.** Click OK.

Working with Ports

A Router transformation has input ports and output ports. Input ports are in the input group, and output ports are in the output groups. You can create input ports by copying them from another transformation or by manually creating them on the Ports tab.

Figure 18-5 shows the Ports tab of a Router transformation:

Figure 18-5. Router Transformation Ports Tab



The Designer creates output ports by copying the following properties from the input ports:

- ◆ Port name
- ◆ Datatype
- ◆ Precision
- ◆ Scale
- ◆ Default value

When you make changes to the input ports, the Designer updates the output ports to reflect these changes. You cannot edit or delete output ports. The output ports display in the Normal view of the Router transformation.

The Designer creates output port names based on the input port names. For each input port, the Designer creates a corresponding output port in each output group.

Figure 18-6 shows the output port names of a Router transformation in Normal view that correspond to the input port names:

Figure 18-6. Input Port Name and Corresponding Output Port Names

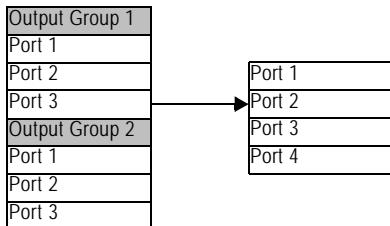
The diagram illustrates the mapping of input port names to corresponding output port names in a Router transformation. On the left, two labels point to specific rows in a table: 'Input Port Name' points to the 'customer_name' row under the 'INPUT' section; 'Corresponding Output Port Names' points to the 'country' row under the 'France' section. Lines connect these labels to their respective rows in the table.

Name	Datatype
INPUT	
customer_name	text
customer_id	integer
country	text
sales_rep_name	text
France	
customer_name1	text
customer_id1	integer
country1	text
sales_rep_name1	text
Japan	
customer_name2	text
customer_id2	integer
country2	text
sales_rep_name2	text
USA	
customer_name3	text
customer_id3	integer
country3	text
sales_rep_name3	text
DEFAULT	
customer_name4	text
customer_id4	integer
country4	text
sales_rep_name4	text

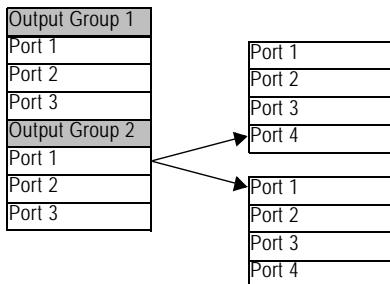
Connecting Router Transformations in a Mapping

When you connect transformations to a Router transformation in a mapping, consider the following rules:

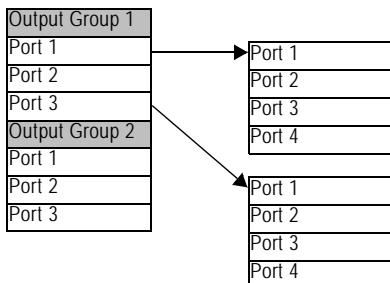
- ◆ You *can* connect one group to one transformation or target.



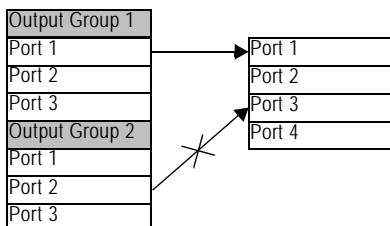
- ◆ You *can* connect one output port in a group to multiple transformations or targets.



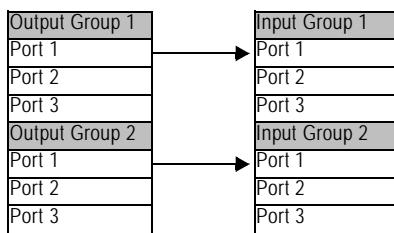
- ◆ You *can* connect multiple output ports in one group to multiple transformations or targets.



- ◆ You *cannot* connect more than one group to one target or a single input group transformation.



- ♦ You *can* connect more than one group to a multiple input group transformation, except for Joiner transformations, when you connect each output group to a different input group.



Creating a Router Transformation

To add a Router transformation to a mapping, complete the following steps.

To create a Router transformation:

1. In the Mapping Designer, open a mapping.
2. Click Transformation > Create.

Select Router transformation, and enter the name of the new transformation. The naming convention for the Router transformation is RTR_*TransformationName*. Click Create, and then click Done.

3. Select and drag all the ports from a transformation to add them to the Router transformation, or you can manually create input ports on the Ports tab.
4. Double-click the title bar of the Router transformation to edit transformation properties.
5. Click the Transformation tab and configure transformation properties.
6. Click the Properties tab and configure tracing levels.

For more information about configuring tracing levels, see “Configuring Tracing Level in Transformations” on page 30.

7. Click the Groups tab, and then click the Add button to create a user-defined group.
The Designer creates the default group when you create the first user-defined group.
8. Click the Group Filter Condition field to open the Expression Editor.
9. Enter a group filter condition.
10. Click Validate to check the syntax of the conditions you entered.
11. Click OK.
12. Connect group output ports to transformations or targets.
13. Click Repository > Save.

Chapter 19

Sequence Generator Transformation

This chapter includes the following topics:

- ◆ Overview, 402
- ◆ Common Uses, 403
- ◆ Sequence Generator Ports, 404
- ◆ Transformation Properties, 407
- ◆ Creating a Sequence Generator Transformation, 412

Overview

Transformation type:

Passive
Connected

The Sequence Generator transformation generates numeric values. Use the Sequence Generator to create unique primary key values, replace missing primary keys, or cycle through a sequential range of numbers.

The Sequence Generator transformation is a connected transformation. It contains two output ports that you can connect to one or more transformations. The Integration Service generates a block of sequence numbers each time a block of rows enters a connected transformation. If you connect CURRVAL, the Integration Service processes one row in each block. When NEXTVAL is connected to the input port of another transformation, the Integration Service generates a sequence of numbers. When CURRVAL is connected to the input port of another transformation, the Integration Service generates the NEXTVAL value plus the Increment By value.

You can make a Sequence Generator reusable, and use it in multiple mappings. You might reuse a Sequence Generator when you perform multiple loads to a single target.

For example, if you have a large input file that you separate into three sessions running in parallel, use a Sequence Generator to generate primary key values. If you use different Sequence Generators, the Integration Service might generate duplicate key values. Instead, use the reusable Sequence Generator for all three sessions to provide a unique value for each target row.

Common Uses

You can complete the following tasks with a Sequence Generator transformation:

- ◆ Create keys.
- ◆ Replace missing values.
- ◆ Cycle through a sequential range of numbers.

Creating Keys

You can create approximately two billion primary or foreign key values with the Sequence Generator transformation by connecting the NEXTVAL port to the transformation or target and using the widest range of values (1 to 2147483647) with the smallest interval (1).

When you create primary or foreign keys, only use the Cycle option to prevent the Integration Service from creating duplicate primary keys. You might do this by selecting the Truncate Target Table option in the session properties (if appropriate) or by creating composite keys.

To create a composite key, you can configure the Integration Service to cycle through a smaller set of values. For example, if you have three stores generating order numbers, you might have a Sequence Generator cycling through values from 1 to 3, incrementing by 1. When you pass the following set of foreign keys, the generated values then create unique composite keys:

COMPOSITE_KEY	ORDER_NO
1	12345
2	12345
3	12345
1	12346
2	12346
3	12346

Replacing Missing Values

Use the Sequence Generator transformation to replace missing keys by using NEXTVAL with the IIF and ISNULL functions.

For example, to replace null values in the ORDER_NO column, you create a Sequence Generator transformation with the properties and drag the NEXTVAL port to an Expression transformation. In the Expression transformation, drag the ORDER_NO port into the transformation (along with any other necessary ports). Then create a new output port, ALL_ORDERS.

In ALL_ORDERS, you can then enter the following expression to replace null orders:

```
IIF( ISNULL( ORDER_NO ), NEXTVAL, ORDER_NO )
```

Sequence Generator Ports

The Sequence Generator transformation provides two output ports: NEXTVAL and CURRVAL. You cannot edit or delete these ports. Likewise, you cannot add ports to the transformation.

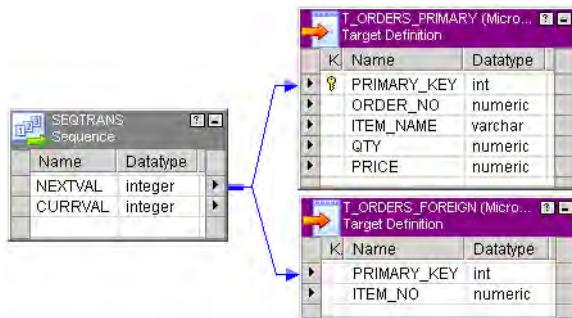
NEXTVAL

Connect NEXTVAL to multiple transformations to generate unique values for each row in each transformation. Use the NEXTVAL port to generate sequence numbers by connecting it to a transformation or target. You connect the NEXTVAL port to a downstream transformation to generate the sequence based on the Current Value and Increment By properties. For more information about Sequence Generator properties, see Table 19-1 on page 407.

For example, you might connect NEXTVAL to two target tables in a mapping to generate unique primary key values. The Integration Service creates a column of unique primary key values for each target table. The column of unique primary key values is sent to one target table as a block of sequence numbers. The second targets receives a block of sequence numbers from the Sequence Generator transformation only after the first target table receives the block of sequence numbers.

Figure 19-1 shows connecting NEXTVAL to two target tables in a mapping:

Figure 19-1. Connecting NEXTVAL to Two Target Tables in a Mapping



For example, you configure the Sequence Generator transformation as follows: Current Value = 1, Increment By = 1. When you run the workflow, the Integration Service generates the following primary key values for the T_ORDERS_PRIMARY and T_ORDERS_FOREIGN target tables:

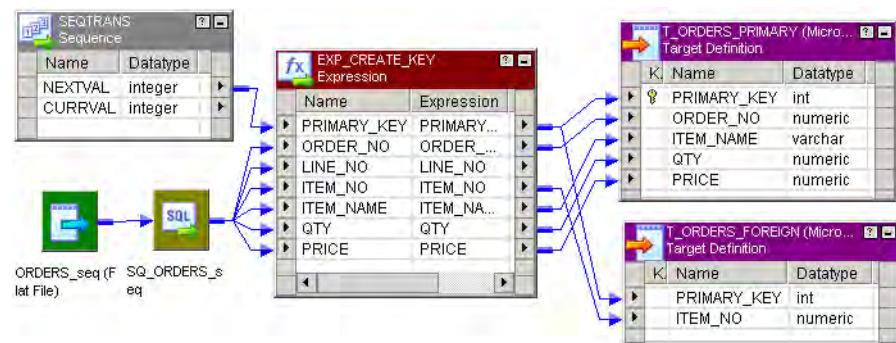
T_ORDERS_PRIMARY TABLE:	T_ORDERS_FOREIGN TABLE:
PRIMARY KEY	PRIMARY KEY
1	6
2	7
3	8

T_ORDERS_PRIMARY TABLE:		T_ORDERS_FOREIGN TABLE:	
PRIMARY KEY		PRIMARY KEY	
4		9	
5		10	

If you want the *same* values to go to more than one target that receives data from a single transformation, you can connect a Sequence Generator transformation to that preceding transformation. The Integration Service processes the values into a block of sequence numbers. This allows the Integration Service to pass unique values to the transformation, and then route rows from the transformation to targets.

Figure 19-2 shows a mapping with a the Sequence Generator that passes unique values to the Expression transformation. The Expression transformation then populates both targets with identical primary key values.

Figure 19-2. Mapping with a Sequence Generator and an Expression Transformation



For example, you configure the Sequence Generator transformation as follows: Current Value = 1, Increment By = 1. When you run the workflow, the Integration Service generates the following primary key values for the T_ORDERS_PRIMARY and T_ORDERS_FOREIGN target tables:

T_ORDERS_PRIMARY TABLE:		T_ORDERS_FOREIGN TABLE:	
PRIMARY KEY		PRIMARY KEY	
1		1	
2		2	
3		3	
4		4	
5		5	

Note: When you run a partitioned session on a grid, the Sequence Generator transformation may skip values depending on the number of rows in each partition.

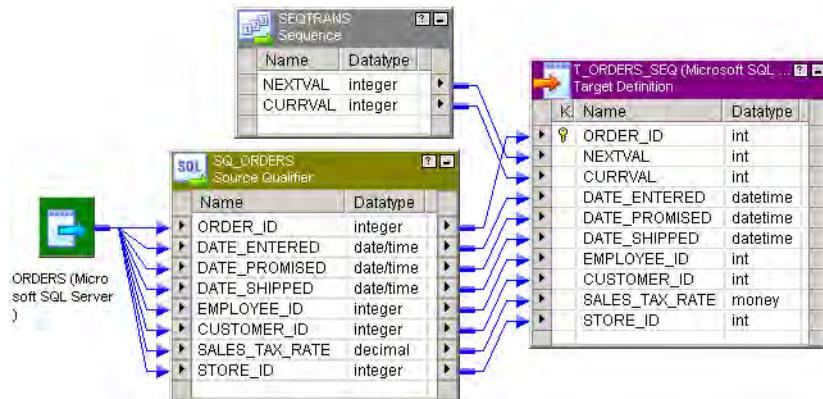
CURRVAL

CURRVAL is NEXTVAL plus the Increment By value. You typically only connect the CURRVAL port when the NEXTVAL port is already connected to a downstream transformation. When a row enters the transformation connected to the CURRVAL port, the Integration Service passes the last-created NEXTVAL value plus one.

For information about the Increment By value, see “Increment By” on page 408.

Figure 19-3 shows connecting CURRVAL and NEXTVAL ports to a target:

Figure 19-3. Connecting CURRVAL and NEXTVAL Ports to a Target



For example, you configure the Sequence Generator transformation as follows: Current Value = 1, Increment By = 1. When you run the workflow, the Integration Service generates the following values for NEXTVAL and CURRVAL:

NEXTVAL	CURRVAL
1	2
2	3
3	4
4	5
5	6

If you connect the CURRVAL port without connecting the NEXTVAL port, the Integration Service passes a constant value for each row.

When you connect the CURRVAL port in a Sequence Generator transformation, the Integration Service processes one row in each block. You can optimize performance by connecting only the NEXTVAL port in a mapping.

Note: When you run a partitioned session on a grid, the Sequence Generator transformation may skip values depending on the number of rows in each partition.

Transformation Properties

The Sequence Generator transformation is unique among all transformations because you cannot add, edit, or delete its default ports (NEXTVAL and CURRVAL).

Table 19-1 lists the Sequence Generator transformation properties you can configure:

Table 19-1. Sequence Generator Transformation Properties

Sequence Generator Setting	Required/Optional	Description
Start Value	Required	Start value of the generated sequence that you want the Integration Service to use if you use the Cycle option. If you select Cycle, the Integration Service cycles back to this value when it reaches the end value. Default is 0.
Increment By	Required	Difference between two consecutive values from the NEXTVAL port. Default is 1.
End Value	Optional	Maximum value the Integration Service generates. If the Integration Service reaches this value during the session and the sequence is not configured to cycle, the session fails.
Current Value	Optional	Current value of the sequence. Enter the value you want the Integration Service to use as the first value in the sequence. If you want to cycle through a series of values, the value must be greater than or equal to the start value and less than the end value. If the Number of Cached Values is set to 0, the Integration Service updates the current value to reflect the last-generated value for the session plus one, and then uses the updated current value as the basis for the next time you run this session. However, if you use the Reset option, the Integration Service resets this value to its original value after each session. Note: If you edit this setting, you reset the sequence to the new setting. If you reset Current Value to 10, and the increment is 1, the next time you use the session, the Integration Service generates a first value of 10.
Cycle	Optional	If selected, the Integration Service cycles through the sequence range. Otherwise, the Integration Service stops the sequence at the configured end value. If disabled, the Integration Service fails the session with overflow errors if it reaches the end value and still has rows to process.
Number of Cached Values	Optional	Number of sequential values the Integration Service caches at a time. Use this option when multiple sessions use the same reusable Sequence Generator at the same time to ensure each session receives unique values. The Integration Service updates the repository as it caches each value. When set to 0, the Integration Service does not cache values. Default value for a standard Sequence Generator is 0. Default value for a reusable Sequence Generator is 1,000.

Table 19-1. Sequence Generator Transformation Properties

Sequence Generator Setting	Required/Optional	Description
Reset	Optional	If selected, the Integration Service generates values based on the original current value for each session. Otherwise, the Integration Service updates the current value to reflect the last-generated value for the session plus one, and then uses the updated current value as the basis for the next session run. This option is disabled for reusable Sequence Generator transformations.
Tracing Level	Optional	Level of detail about the transformation that the Integration Service writes into the session log.

Start Value and Cycle

Use Cycle to generate a repeating sequence, such as numbers 1 through 12 to correspond to the months in a year.

To cycle the Integration Service through a sequence:

1. Enter the lowest value in the sequence that you want the Integration Service to use for the Start Value.
2. Enter the highest value to be used for End Value.
3. Select Cycle.

As it cycles, the Integration Service reaches the configured end value for the sequence, it wraps around and starts the cycle again, beginning with the configured Start Value.

Increment By

The Integration Service generates a sequence (NEXTVAL) based on the Current Value and Increment By properties in the Sequence Generator transformation.

The Current Value property is the value at which the Integration Service starts creating the sequence for each session. Increment By is the integer the Integration Service adds to the existing value to create the new value in the sequence. By default, the Current Value is set to 1, and Increment By is set to 1.

For example, you might create a Sequence Generator transformation with a current value of 1,000 and an increment of 10. If you pass three rows through the mapping, the Integration Service generates the following set of values:

1000
1010
1020

End Value

End Value is the maximum value you want the Integration Service to generate. If the Integration Service reaches the end value and the Sequence Generator is not configured to cycle through the sequence, the session fails with the following error message:

TT_11009 Sequence Generator Transformation: Overflow error.

You can set the end value to any integer between 1 and 2,147,483,647.

Current Value

The Integration Service uses the current value as the basis for generated values for each session. To indicate which value you want the Integration Service to use the first time it uses the Sequence Generator transformation, you must enter that value as the current value. If you want to use the Sequence Generator transformation to cycle through a series of values, the current value must be greater than or equal to Start Value and less than the end value.

At the end of each session, the Integration Service updates the current value to the last value generated for the session plus one if the Sequence Generator Number of Cached Values is 0. For example, if the Integration Service ends a session with a generated value of 101, it updates the Sequence Generator current value to 102 in the repository. The next time the Sequence Generator is used, the Integration Service uses 102 as the basis for the next generated value. If the Sequence Generator Increment By is 1, when the Integration Service starts another session using the Sequence Generator, the first generated value is 102.

If you have multiple versions of a Sequence Generator transformation, the Integration Service updates the current value across all versions when it runs a session. The Integration Service updates the current value across versions regardless of whether you have checked out the Sequence Generator transformation or the parent mapping. The updated current value overrides an edited current value for a Sequence Generator transformation if the two values are different.

For example, User 1 creates Sequence Generator transformation and checks it in, saving a current value of 10 to Sequence Generator version 1. Then User 1 checks out the Sequence Generator transformation and enters a new current value of 100 to Sequence Generator version 2. User 1 keeps the Sequence Generator transformation checked out. Meanwhile, User 2 runs a session that uses the Sequence Generator transformation version 1. The Integration Service uses the checked-in value of 10 as the current value when User 2 runs the session. When the session completes, the current value is 150. The Integration Service updates the current value to 150 for version 1 and version 2 of the Sequence Generator transformation even though User 1 has the Sequence Generator transformation checked out.

If you open the mapping after you run the session, the current value displays the last value generated for the session plus one. Since the Integration Service uses the current value to determine the first value for each session, you should edit the current value only when you want to reset the sequence.

If you have multiple versions of the Sequence Generator transformation, and you want to reset the sequence, you must check in the mapping or Sequence Generator (reusable) transformation after you modify the current value.

Note: If you configure the Sequence Generator to Reset, the Integration Service uses the current value as the basis for the first generated value for each session.

Number of Cached Values

Number of Cached Values determines the number of values the Integration Service caches at one time. When Number of Cached Values is greater than zero, the Integration Service caches the configured number of values and updates the current value each time it caches values.

When multiple sessions use the same reusable Sequence Generator transformation at the same time, there might be multiple instances of the Sequence Generator transformation. To avoid generating the same values for each session, reserve a range of sequence values for each session by configuring Number of Cached Values.

Tip: To increase performance when running a session on a grid, increase the number of cached values for the Sequence Generator transformation. This reduces the communication required between the master and worker DTM processes and the repository.

Non-Reusable Sequence Generators

For non-reusable Sequence Generator transformations, Number of Cached Values is set to zero by default, and the Integration Service does not cache values during the session. When the Integration Service does not cache values, it accesses the repository for the current value at the start of a session. The Integration Service then generates values for the sequence. At the end of the session, the Integration Service updates the current value in the repository.

When you set Number of Cached Values greater than zero, the Integration Service caches values during the session. At the start of the session, the Integration Service accesses the repository for the current value, caches the configured number of values, and updates the current value accordingly. If the Integration Service exhausts the cache, it accesses the repository for the next set of values and updates the current value. At the end of the session, the Integration Service discards any remaining values in the cache.

For non-reusable Sequence Generator transformations, setting Number of Cached Values greater than zero can increase the number of times the Integration Service accesses the repository during the session. It also causes sections of skipped values since unused cached values are discarded at the end of each session.

For example, you configure a Sequence Generator transformation as follows: Number of Cached Values = 50, Current Value = 1, Increment By = 1. When the Integration Service starts the session, it caches 50 values for the session and updates the current value to 50 in the repository. The Integration Service uses values 1 to 39 for the session and discards the unused values, 40 to 49. When the Integration Service runs the session again, it checks the repository for the current value, which is 50. It then caches the next 50 values and updates the current value to 100. During the session, it uses values 50 to 98. The values generated for the two sessions are 1 to 39 and 50 to 98.

Reusable Sequence Generators

When you have a reusable Sequence Generator transformation in several sessions and the sessions run at the same time, use Number of Cached Values to ensure each session receives unique values in the sequence. By default, Number of Cached Values is set to 1000 for reusable Sequence Generators.

When multiple sessions use the same Sequence Generator transformation at the same time, you risk generating the same values for each session. To avoid this, have the Integration Service cache a set number of values for each session by configuring Number of Cached Values.

For example, you configure a reusable Sequence Generator transformation as follows: Number of Cached Values = 50, Current Value = 1, Increment By = 1. Two sessions use the Sequence Generator, and they are scheduled to run at approximately the same time. When the Integration Service starts the first session, it caches 50 values for the session and updates the current value to 50 in the repository. The Integration Service begins using values 1 to 50 in the session. When the Integration Service starts the second session, it checks the repository for the current value, which is 50. It then caches the next 50 values and updates the current value to 100. It then uses values 51 to 100 in the second session. When either session uses all its cached values, the Integration Service caches a new set of values and updates the current value to ensure these values remain unique to the Sequence Generator.

For reusable Sequence Generator transformations, you can reduce Number of Cached Values to minimize discarded values, however it must be greater than one. When you reduce the Number of Cached Values, you might increase the number of times the Integration Service accesses the repository to cache values during the session.

Reset

If you select Reset for a non-reusable Sequence Generator transformation, the Integration Service generates values based on the original current value each time it starts the session. Otherwise, the Integration Service updates the current value to reflect the last-generated value plus one, and then uses the updated value the next time it uses the Sequence Generator transformation.

For example, you might configure a Sequence Generator transformation to create values from 1 to 1,000 with an increment of 1, and a current value of 1 and choose Reset. During the first session run, the Integration Service generates numbers 1 through 234. The next time (and each subsequent time) the session runs, the Integration Service again generates numbers beginning with the current value of 1.

If you do not select Reset, the Integration Service updates the current value to 235 at the end of the first session run. The next time it uses the Sequence Generator transformation, the first value generated is 235.

Note: Reset is disabled for reusable Sequence Generator transformations.

Creating a Sequence Generator Transformation

To use a Sequence Generator transformation in a mapping, add it to the mapping, configure the transformation properties, and then connect NEXTVAL or CURRVAL to one or more transformations.

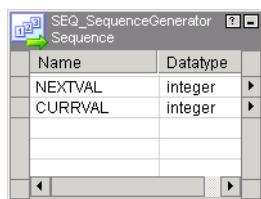
To create a Sequence Generator transformation:

1. In the Mapping Designer, click Transformation > Create. Select the Sequence Generator transformation.

The naming convention for Sequence Generator transformations is SEQ_TransformationName.

2. Enter a name for the Sequence Generator, and click Create. Click Done.

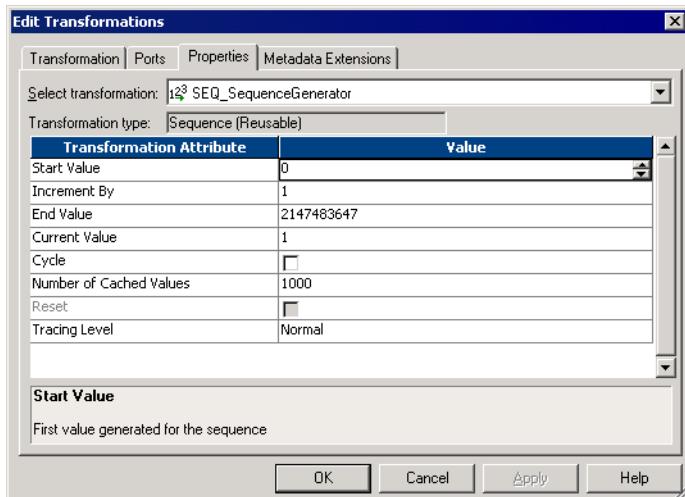
The Designer creates the Sequence Generator transformation.



3. Double-click the title bar of the transformation to open the Edit Transformations dialog box.
4. Enter a description for the transformation. This description appears in the Repository Manager, making it easier for you or others to understand what the transformation does.
5. Select the Properties tab. Enter settings.

For a list of transformation properties, see Table 19-1 on page 407.

Note: You cannot override the Sequence Generator transformation properties at the session level. This protects the integrity of the sequence values generated.



- 6.** Click OK.
- 7.** To generate new sequences during a session, connect the NEXTVAL port to at least one transformation in the mapping.

Use the NEXTVAL or CURRVAL ports in an expression in other transformations.
- 8.** Click Repository > Save.

Chapter 20

Sorter Transformation

This chapter includes the following topics:

- ◆ Overview, 416
- ◆ Sorting Data, 417
- ◆ Sorter Transformation Properties, 419
- ◆ Creating a Sorter Transformation, 423

Overview

Transformation type:

Active
Connected

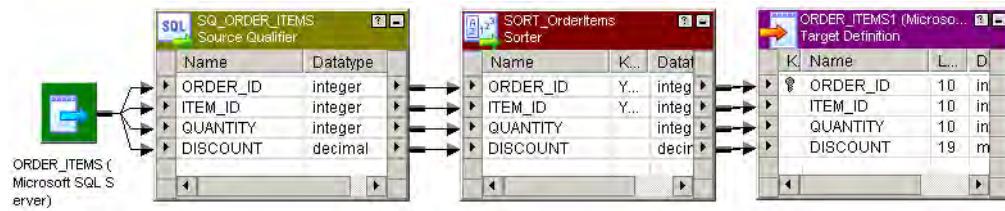
You can sort data with the Sorter transformation. You can sort data in ascending or descending order according to a specified sort key. You can also configure the Sorter transformation for case-sensitive sorting, and specify whether the output rows should be distinct. The Sorter transformation is an active transformation. It must be connected to the data flow.

You can sort data from relational or flat file sources. You can also use the Sorter transformation to sort data passing through an Aggregator transformation configured to use sorted input.

When you create a Sorter transformation in a mapping, you specify one or more ports as a sort key and configure each sort key port to sort in ascending or descending order. You also configure sort criteria the Integration Service applies to all sort key ports and the system resources it allocates to perform the sort operation.

Figure 20-1 shows a simple mapping that uses a Sorter transformation. The mapping passes rows from a sales table containing order information through a Sorter transformation before loading to the target.

Figure 20-1. Sample Mapping with a Sorter Transformation



Sorting Data

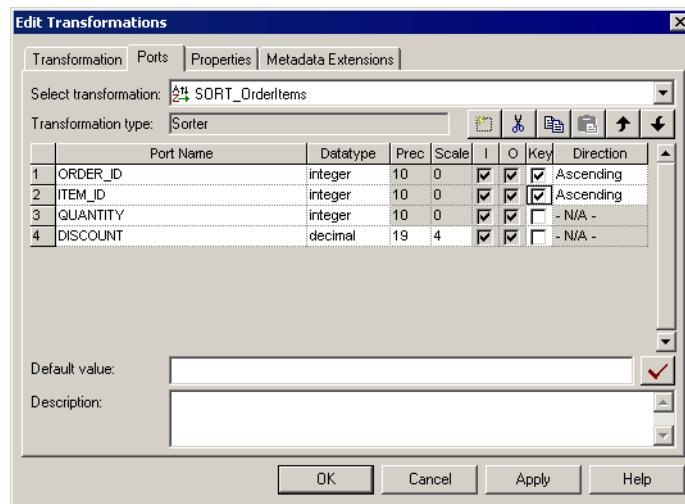
The Sorter transformation contains only input/output ports. All data passing through the Sorter transformation is sorted according to a sort key. The sort key is one or more ports that you want to use as the sort criteria.

You can specify more than one port as part of the sort key. When you specify multiple ports for the sort key, the Integration Service sorts each port sequentially. The order the ports appear in the Ports tab determines the succession of sort operations. The Sorter transformation treats the data passing through each successive sort key port as a secondary sort of the previous port.

At session run time, the Integration Service sorts data according to the sort order specified in the session properties. The sort order determines the sorting criteria for special characters and symbols.

Figure 20-2 shows the Ports tab configuration for the Sorter transformation sorting the data in ascending order by order ID and item ID:

Figure 20-2. Sample Sorter Transformation Ports Configuration



At session run time, the Integration Service passes the following rows into the Sorter transformation:

ORDER_ID	ITEM_ID	QUANTITY	DISCOUNT
45	123456	3	3 .04
45	456789	2	12 .02
43	000246	6	34 .55
41	000468	5	.56

After sorting the data, the Integration Service passes the following rows out of the Sorter transformation:

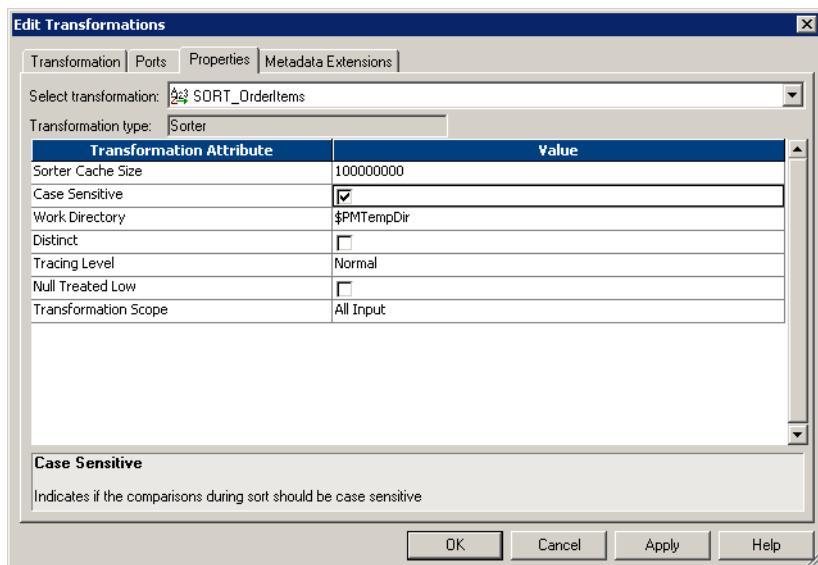
ORDER_ID	ITEM_ID	QUANTITY	DISCOUNT
41	000468	5	.56
43	000246	6	34.55
45	123456	3	3.04
45	456789	2	12.02

Sorter Transformation Properties

The Sorter transformation has several properties that specify additional sort criteria. The Integration Service applies these criteria to all sort key ports. The Sorter transformation properties also determine the system resources the Integration Service allocates when it sorts data.

Figure 20-3 shows the Sorter transformation Properties tab:

Figure 20-3. Sorter Transformation Properties



Sorter Cache Size

The Integration Service uses the Sorter Cache Size property to determine the maximum amount of memory it can allocate to perform the sort operation. The Integration Service passes all incoming data into the Sorter transformation before it performs the sort operation. You can configure a numeric value for the Sorter cache, or you can configure the Integration Service to determine the cache size at runtime. If you configure the Integration Service to determine the cache size, you can also configure a maximum amount of memory for the Integration Service to allocate to the cache.

If the total configured session cache size is 2 GB (2,147,483,648 bytes) or greater, you must run the session on a 64-bit Integration Service.

Before starting the sort operation, the Integration Service allocates the amount of memory configured for the Sorter cache size. If the Integration Service runs a partitioned session, it allocates the specified amount of Sorter cache memory for each partition.

If it cannot allocate enough memory, the Integration Service fails the session. For best performance, configure Sorter cache size with a value less than or equal to the amount of

available physical RAM on the Integration Service machine. Allocate at least 8 MB (8,388,608 bytes) of physical memory to sort data using the Sorter transformation. Sorter cache size is set to 8,388,608 bytes by default.

If the amount of incoming data is greater than the amount of Sorter cache size, the Integration Service temporarily stores data in the Sorter transformation work directory. The Integration Service requires disk space of at least twice the amount of incoming data when storing data in the work directory. If the amount of incoming data is significantly greater than the Sorter cache size, the Integration Service may require much more than twice the amount of disk space available to the work directory.

Use the following formula to determine the size of incoming data:

```
number_of_input_rows [ ( Σ column_size ) + 16 ]
```

Table 20-1 gives the individual column size values by datatype for Sorter data calculations:

Table 20-1. Column Sizes for Sorter Data Calculations

Datatype	Column Size
Binary	precision + 8 Round to nearest multiple of 8
Date/Time	24
Decimal, high precision off (all precision)	16
Decimal, high precision on (precision <=18)	24
Decimal, high precision on (precision >18, <=28)	32
Decimal, high precision on (precision >28)	16
Decimal, high precision on (negative scale)	16
Double	16
Real	16
Integer	16
Small integer	16
NSString, NText, String, Text	Unicode mode: 2*(precision + 5) ASCII mode: precision + 9

The column sizes include the bytes required for a null indicator.

To increase performance for the sort operation, the Integration Service aligns all data for the Sorter transformation memory on an 8-byte boundary. Each Sorter column includes rounding to the nearest multiple of eight.

The Integration Service also writes the row size and amount of memory the Sorter transformation uses to the session log when you configure the Sorter transformation tracing level to Normal. For more information about Sorter transformation tracing levels, see “Tracing Level” on page 421.

Case Sensitive

The Case Sensitive property determines whether the Integration Service considers case when sorting data. When you enable the Case Sensitive property, the Integration Service sorts uppercase characters higher than lowercase characters.

Work Directory

You must specify a work directory the Integration Service uses to create temporary files while it sorts data. After the Integration Service sorts the data, it deletes the temporary files. You can specify any directory on the Integration Service machine to use as a work directory. By default, the Integration Service uses the value specified for the \$PMTempDir process variable.

When you partition a session with a Sorter transformation, you can specify a different work directory for each partition in the pipeline. To increase session performance, specify work directories on physically separate disks on the Integration Service system.

Distinct Output Rows

You can configure the Sorter transformation to treat output rows as distinct. If you configure the Sorter transformation for distinct output rows, the Mapping Designer configures all ports as part of the sort key. When the Integration Service runs the session, it discards duplicate rows compared during the sort operation.

Tracing Level

Configure the Sorter transformation tracing level to control the number and type of Sorter error and status messages the Integration Service writes to the session log. At Normal tracing level, the Integration Service writes the size of the row passed to the Sorter transformation and the amount of memory the Sorter transformation allocates for the sort operation. The Integration Service also writes the time and date when it passes the first and last input rows to the Sorter transformation.

If you configure the Sorter transformation tracing level to Verbose Data, the Integration Service writes the time the Sorter transformation finishes passing all data to the next transformation in the pipeline. The Integration Service also writes the time to the session log when the Sorter transformation releases memory resources and removes temporary files from the work directory.

For more information about configuring tracing levels for transformations, see “Configuring Tracing Level in Transformations” on page 30.

Null Treated Low

You can configure the way the Sorter transformation treats null values. Enable this property if you want the Integration Service to treat null values as lower than any other value when it performs the sort operation. Disable this option if you want the Integration Service to treat null values as higher than any other value.

Transformation Scope

The transformation scope specifies how the Integration Service applies the transformation logic to incoming data:

- ♦ **Transaction.** Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions.
- ♦ **All Input.** Applies the transformation logic on all incoming data. When you choose All Input, the PowerCenter drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source.

For more information about transformation scope, see “Understanding Commit Points” in the *Workflow Administration Guide*.

Creating a Sorter Transformation

To add a Sorter transformation to a mapping, complete the following steps.

To create a Sorter transformation:

1. In the Mapping Designer, click Transformation > Create. Select the Sorter transformation.

The naming convention for Sorter transformations is SRT_*TransformationName*. Enter a description for the transformation. This description appears in the Repository Manager, making it easier to understand what the transformation does.

2. Enter a name for the Sorter and click Create.

The Designer creates the Sorter transformation.

3. Click Done.

4. Drag the ports you want to sort into the Sorter transformation.

The Designer creates the input/output ports for each port you include.

5. Double-click the title bar of the transformation to open the Edit Transformations dialog box.

6. Select the Ports tab.

7. Select the ports you want to use as the sort key.

8. For each port selected as part of the sort key, specify whether you want the Integration Service to sort data in ascending or descending order.

9. Select the Properties tab. Modify the Sorter transformation properties. For information about Sorter transformation properties, see “Sorter Transformation Properties” on page 419.

10. Select the Metadata Extensions tab. Create or edit metadata extensions for the Sorter transformation. For more information about metadata extensions, see “Metadata Extensions” in the *Repository Guide*.

11. Click OK.

12. Click Repository > Save to save changes to the mapping.

Chapter 21

Source Qualifier Transformation

This chapter includes the following topics:

- ◆ Overview, 426
- ◆ Source Qualifier Transformation Properties, 429
- ◆ Default Query, 431
- ◆ Joining Source Data, 434
- ◆ Adding an SQL Query, 438
- ◆ Entering a User-Defined Join, 440
- ◆ Outer Join Support, 442
- ◆ Entering a Source Filter, 450
- ◆ Using Sorted Ports, 452
- ◆ Select Distinct, 454
- ◆ Adding Pre- and Post-Session SQL Commands, 455
- ◆ Creating a Source Qualifier Transformation, 456
- ◆ Troubleshooting, 458

Overview

Transformation type:

Active
Connected

When you add a relational or a flat file source definition to a mapping, you need to connect it to a Source Qualifier transformation. The Source Qualifier transformation represents the rows that the Integration Service reads when it runs a session.

Use the Source Qualifier transformation to complete the following tasks:

- ◆ **Join data originating from the same source database.** You can join two or more tables with primary key-foreign key relationships by linking the sources to one Source Qualifier transformation.
- ◆ **Filter rows when the Integration Service reads source data.** If you include a filter condition, the Integration Service adds a WHERE clause to the default query.
- ◆ **Specify an outer join rather than the default inner join.** If you include a user-defined join, the Integration Service replaces the join information specified by the metadata in the SQL query.
- ◆ **Specify sorted ports.** If you specify a number for sorted ports, the Integration Service adds an ORDER BY clause to the default SQL query.
- ◆ **Select only distinct values from the source.** If you choose Select Distinct, the Integration Service adds a SELECT DISTINCT statement to the default SQL query.
- ◆ **Create a custom query to issue a special SELECT statement for the Integration Service to read source data.** For example, you might use a custom query to perform aggregate calculations.

Transformation Datatypes

The Source Qualifier transformation displays the transformation datatypes. The transformation datatypes determine how the source database binds data when the Integration Service reads it. Do not alter the datatypes in the Source Qualifier transformation. If the datatypes in the source definition and Source Qualifier transformation do not match, the Designer marks the mapping invalid when you save it.

Target Load Order

You specify a target load order based on the Source Qualifier transformations in a mapping. If you have multiple Source Qualifier transformations connected to multiple targets, you can designate the order in which the Integration Service loads data into the targets.

If one Source Qualifier transformation provides data for multiple targets, you can enable constraint-based loading in a session to have the Integration Service load data based on target table primary and foreign key relationships.

For more information, see “Mappings” in the *Designer Guide*.

Parameters and Variables

Use mapping parameters and variables in the SQL query, user-defined join, and source filter of a Source Qualifier transformation. You can also use the system variable `$$$$SessStartTime`.

The Integration Service first generates an SQL query and replaces each mapping parameter or variable with its start value. Then it runs the query on the source database.

When you use a string mapping parameter or variable in the Source Qualifier transformation, use a string identifier appropriate to the source system. Most databases use a single quotation mark as a string identifier. For example, to use the string parameter `$$IPAddress` in a source filter for a Microsoft SQL Server database table, enclose the parameter in single quotes as follows, ‘`$$IPAddress`’. For more information, see the database documentation.

When you use a datetime mapping parameter or variable, or when you use the system variable `$$$$SessStartTime`, you might need to change the date format to the format used in the source. The Integration Service passes datetime parameters and variables to source systems as strings in the SQL query. The Integration Service converts a datetime parameter or variable to a string, based on the source database.

Table 21-1 describes the datetime formats the Integration Service uses for each source system:

Table 21-1. Conversion for Datetime Mapping Parameters and Variables

Source	Date Format
DB2	YYYY-MM-DD-HH24:MI:SS
Informix	YYYY-MM-DD HH24:MI:SS
Microsoft SQL Server	MM/DD/YYYY HH24:MI:SS
ODBC	YYYY-MM-DD HH24:MI:SS
Oracle	MM/DD/YYYY HH24:MI:SS
Sybase	MM/DD/YYYY HH24:MI:SS
Teradata	YYYY-MM-DD HH24:MI:SS

Some databases require you to identify datetime values with additional punctuation, such as single quotation marks or database specific functions. For example, to convert the `$$$$SessStartTime` value for an Oracle source, use the following Oracle function in the SQL override:

```
to_date ('$$$$SessStartTime', 'mm/dd/yyyy hh24:mi:ss')
```

For Informix, use the following Informix function in the SQL override to convert the `$$$$SessStartTime` value:

```
DATETIME ($$$$SessStartTime) YEAR TO SECOND
```

For more information about SQL override, see “Overriding the Default Query” on page 432. For information about database specific functions, see the database documentation.

Tip: To ensure the format of a datetime parameter or variable matches that used by the source, validate the SQL query.

For information about mapping parameters and variables, see “Mapping Parameters and Variables” in the *Designer Guide*.

Source Qualifier Transformation Properties

Configure the Source Qualifier transformation properties on the Properties tab of the Edit Transformations dialog box.

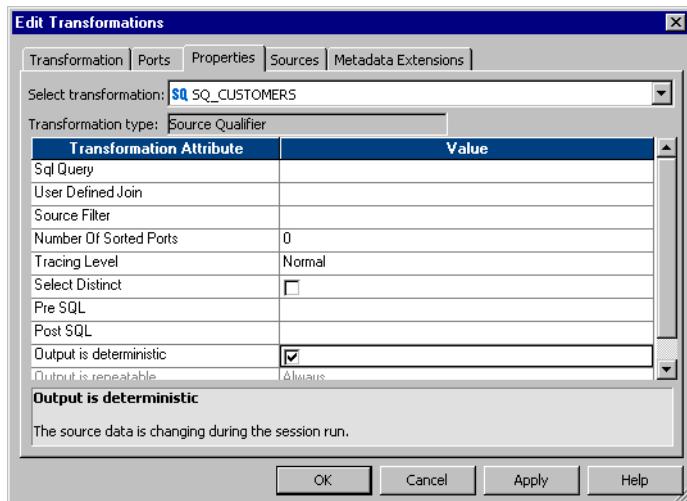


Table 21-2 describes the Source Qualifier transformation properties:

Table 21-2. Source Qualifier Transformation Properties

Option	Description
SQL Query	Defines a custom query that replaces the default query the Integration Service uses to read data from sources represented in this Source Qualifier transformation. For more information, see "Adding an SQL Query" on page 438. A custom query overrides entries for a custom join or a source filter.
User-Defined Join	Specifies the condition used to join data from multiple sources represented in the same Source Qualifier transformation. For more information, see "Entering a User-Defined Join" on page 440.
Source Filter	Specifies the filter condition the Integration Service applies when querying rows. For more information, see "Entering a Source Filter" on page 450.
Number of Sorted Ports	Indicates the number of columns used when sorting rows queried from relational sources. If you select this option, the Integration Service adds an ORDER BY to the default query when it reads source rows. The ORDER BY includes the number of ports specified, starting from the top of the transformation. When selected, the database sort order must match the session sort order.
Tracing Level	Sets the amount of detail included in the session log when you run a session containing this transformation. For more information, see "Configuring Tracing Level in Transformations" on page 30.
Select Distinct	Specifies if you want to select only unique rows. The Integration Service includes a SELECT DISTINCT statement if you choose this option.

Table 21-2. Source Qualifier Transformation Properties

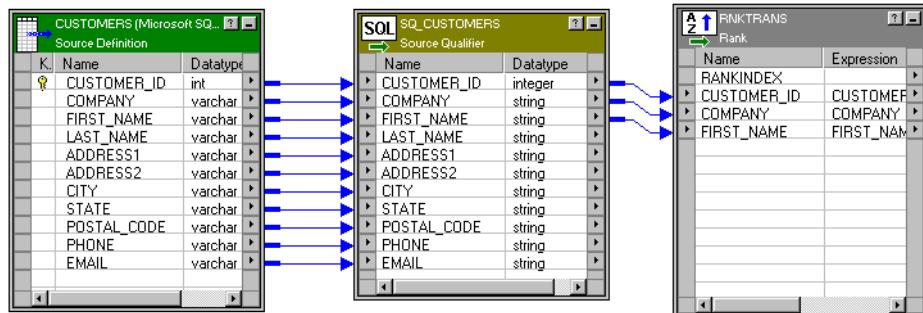
Option	Description
Pre-SQL	Pre-session SQL commands to run against the source database before the Integration Service reads the source. For more information, see "Adding Pre- and Post-Session SQL Commands" on page 455.
Post-SQL	Post-session SQL commands to run against the source database after the Integration Service writes to the target. For more information, see "Adding Pre- and Post-Session SQL Commands" on page 455.
Output is Deterministic	Relational source or transformation output that does not change between session runs when the input data is consistent between runs. When you configure this property, the Integration Service does not stage source data for recovery if transformations in the pipeline always produce repeatable data.
Output is Repeatable	Relational source or transformation output that is in the same order between session runs when the order of the input data is consistent. When output is deterministic and output is repeatable, the Integration Service does not stage source data for recovery.

Default Query

For relational sources, the Integration Service generates a query for each Source Qualifier transformation when it runs a session. The default query is a SELECT statement for each source column used in the mapping. In other words, the Integration Service reads only the columns that are connected to another transformation.

Figure 21-1 shows a single source definition connected to a Source Qualifier transformation:

Figure 21-1. Source Definition Connected to a Source Qualifier Transformation



Although there are many columns in the source definition, only three columns are connected to another transformation. In this case, the Integration Service generates a default query that selects only those three columns:

```
SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME  
FROM CUSTOMERS
```

If any table name or column name contains a database reserved word, you can create and maintain a file, reswords.txt, containing reserved words. When the Integration Service initializes a session, it searches for reswords.txt in the Integration Service installation directory. If the file exists, the Integration Service places quotes around matching reserved words when it executes SQL against the database. If you override the SQL, you must enclose any reserved word in quotes. For more information about the reserved words file, see “Working with Targets” in the *Workflow Administration Guide*.

When generating the default query, the Designer delimits table and field names containing the following characters with double quotes:

```
/ + - = ~ ` ! % ^ & * ( ) [ ] { } ' ; ? , < > \ | <space>
```

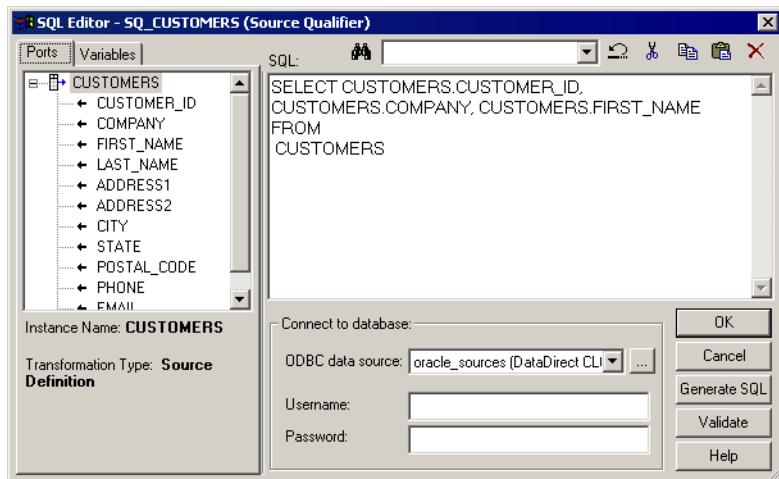
Viewing the Default Query

You can view the default query in the Source Qualifier transformation.

To view the default query:

- From the Properties tab, select SQL Query.

The SQL Editor appears.



The SQL Editor displays the default query the Integration Service uses to select source data.

- Click Generate SQL.

- Click Cancel to exit.

Note: If you do not cancel the SQL query, the Integration Service overrides the default query with the custom SQL query.

Do not connect to the source database. You only connect to the source database when you enter an SQL query that overrides the default query.

Tip: You must connect the columns in the Source Qualifier transformation to another transformation or target before you can generate the default query.

Overriding the Default Query

You can alter or override the default query in the Source Qualifier transformation by changing the default settings of the transformation properties. Do not change the list of selected ports or the order in which they appear in the query. This list must match the connected transformation output ports.

When you edit transformation properties, the Source Qualifier transformation includes these settings in the default query. However, if you enter an SQL query, the Integration Service uses

only the defined SQL statement. The SQL Query overrides the User-Defined Join, Source Filter, Number of Sorted Ports, and Select Distinct settings in the Source Qualifier transformation.

Note: When you override the default SQL query, you must enclose all database reserved words in quotes.

Joining Source Data

Use one Source Qualifier transformation to join data from multiple relational tables. These tables must be accessible from the same instance or database server.

When a mapping uses related relational sources, you can join both sources in one Source Qualifier transformation. During the session, the source database performs the join before passing data to the Integration Service. This can increase performance when source tables are indexed.

Tip: Use the Joiner transformation for heterogeneous sources and to join flat files.

Default Join

When you join related tables in one Source Qualifier transformation, the Integration Service joins the tables based on the related keys in each table.

This default join is an inner equijoin, using the following syntax in the WHERE clause:

```
Source1.column_name = Source2.column_name
```

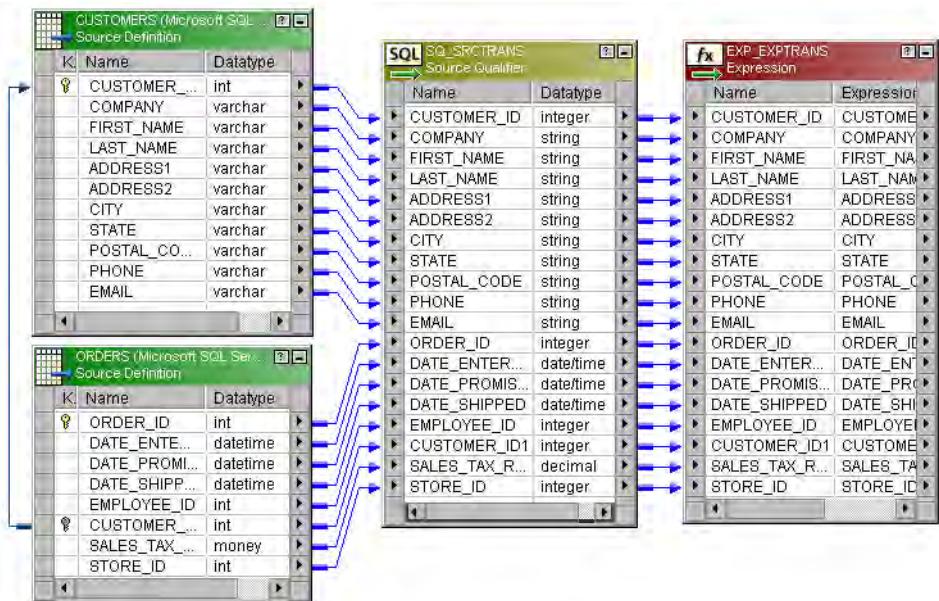
The columns in the default join must have:

- ♦ A primary key-foreign key relationship
- ♦ Matching datatypes

For example, you might see all the orders for the month, including order number, order amount, and customer name. The ORDERS table includes the order number and amount of each order, but not the customer name. To include the customer name, you need to join the ORDERS and CUSTOMERS tables. Both tables include a customer ID, so you can join the tables in one Source Qualifier transformation.

Figure 21-2 shows joining two tables with one Source Qualifier transformation:

Figure 21-2. Joining Two Tables with One Source Qualifier Transformation



When you include multiple tables, the Integration Service generates a SELECT statement for all columns used in the mapping. In this case, the SELECT statement looks similar to the following statement:

```
SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME,
       CUSTOMERS.LAST_NAME, CUSTOMERS.ADDRESS1, CUSTOMERS.ADDRESS2,
       CUSTOMERS.CITY, CUSTOMERS.STATE, CUSTOMERS.POSTAL_CODE, CUSTOMERS.PHONE,
       CUSTOMERS.EMAIL, ORDERS.ORDER_ID, ORDERS.DATE_ENTERED,
       ORDERS.DATE_PROMISED, ORDERS.DATE_SHIPPED, ORDERS.EMPLOYEE_ID,
       ORDERS.CUSTOMER_ID, ORDERS.SALES_TAX_RATE, ORDERS.STORE_ID
  FROM CUSTOMERS, ORDERS
 WHERE CUSTOMERS.CUSTOMER_ID=ORDERS.CUSTOMER_ID
```

The WHERE clause is an equijoin that includes the CUSTOMER_ID from the ORDERS and CUSTOMER tables.

Custom Joins

If you need to override the default join, you can enter contents of the WHERE clause that specifies the join in the custom query. If the query performs an outer join, the Integration Service may insert the join syntax in the WHERE clause or the FROM clause, depending on the database syntax.

You might need to override the default join under the following circumstances:

- ◆ Columns do not have a primary key-foreign key relationship.

- ◆ The datatypes of columns used for the join do not match.
 - ◆ You want to specify a different type of join, such as an outer join.
- For more information about custom joins and queries, see “Entering a User-Defined Join” on page 440.

Heterogeneous Joins

To perform a heterogeneous join, use the Joiner transformation. Use the Joiner transformation when you need to join the following types of sources:

- ◆ Join data from different source databases
- ◆ Join data from different flat file systems
- ◆ Join relational sources and flat files

For more information, see “Joiner Transformation” on page 283.

Creating Key Relationships

You can join tables in the Source Qualifier transformation if the tables have primary key-foreign key relationships. However, you can create primary key-foreign key relationships in the Source Analyzer by linking matching columns in different tables. These columns do not have to be keys, but they should be included in the index for each table.

Tip: If the source table has more than 1,000 rows, you can increase performance by indexing the primary key-foreign keys. If the source table has fewer than 1,000 rows, you might decrease performance if you index the primary key-foreign keys.

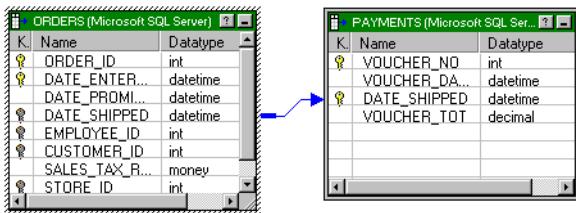
For example, the corporate office for a retail chain wants to extract payments received based on orders. The ORDERS and PAYMENTS tables do not share primary and foreign keys. Both tables, however, include a DATE_SHIPPED column. You can create a primary key-foreign key relationship in the metadata in the Source Analyzer.

Note, the two tables are not linked. Therefore, the Designer does not recognize the relationship on the DATE_SHIPPED columns.

You create a relationship between the ORDERS and PAYMENTS tables by linking the DATE_SHIPPED columns. The Designer adds primary and foreign keys to the DATE_SHIPPED columns in the ORDERS and PAYMENTS table definitions.

Figure 21-3 shows a relationship between two tables:

Figure 21-3. Creating a Relationship Between Two Tables



If you do not connect the columns, the Designer does not recognize the relationships.

The primary key-foreign key relationships exist in the metadata only. You do not need to generate SQL or alter the source tables.

Once the key relationships exist, use a Source Qualifier transformation to join the two tables. The default join is based on DATE_SHIPPED.

Adding an SQL Query

The Source Qualifier transformation provides the SQL Query option to override the default query. You can enter an SQL statement supported by the source database. Before entering the query, connect all the input and output ports you want to use in the mapping.

When you edit the SQL Query, you can generate and edit the default query. When the Designer generates the default query, it incorporates all other configured options, such as a filter or number of sorted ports. The resulting query overrides all other options you might subsequently configure in the transformation.

You can include mapping parameters and variables in the SQL Query. When including a string mapping parameter or variable, use a string identifier appropriate to the source system. For most databases, you should enclose the name of a string parameter or variable in single quotes. For more information, see the database documentation.

When you include a datetime parameter or variable, you might need to change the date format to match the format used by the source. The Integration Service converts a datetime parameter and variable to a string based on the source system. For more information about date conversion, see Table 21-1 on page 427.

When creating a custom SQL query, the SELECT statement must list the port names in the order in which they appear in the transformation.

When you override the default SQL query for a session configured for pushdown optimization, the Integration Service creates a view to represent the SQL override. It then runs an SQL query against this view to push the transformation logic to the database. For more information about working with SQL overrides and pushdown optimization, see “Using Pushdown Optimization” in the *Workflow Administration Guide*.

If you edit the SQL query, you must enclose all database reserved words in quotes. For more information about reserved words, see “Working with Targets” in the *Workflow Administration Guide*.

To override the default query:

1. Open the Source Qualifier transformation, and click the Properties tab.
2. Click the Open button in the SQL Query field. The SQL Editor dialog box appears.
3. Click Generate SQL.

The Designer displays the default query it generates when querying rows from all sources included in the Source Qualifier transformation.

4. Enter a query in the space where the default query appears.

Every column name must be qualified by the name of the table, view, or synonym in which it appears. For example, if you want to include the ORDER_ID column from the ORDERS table, enter ORDERS.ORDER_ID. You can double-click column names appearing in the Ports window to avoid typing the name of every column.

Enclose string mapping parameters and variables in string identifiers. Alter the date format for datetime mapping parameters and variables when necessary.

5. Select the ODBC data source containing the sources included in the query.
6. Enter the user name and password to connect to this database.
7. Click Validate.

The Designer runs the query and reports whether its syntax was correct.

8. Click OK to return to the Edit Transformations dialog box. Click OK again to return to the Designer.
9. Click Repository > Save.

Tip: You can resize the Expression Editor. Expand the dialog box by dragging from the borders. The Designer saves the new size for the dialog box as a client setting.

Entering a User-Defined Join

Entering a user-defined join is similar to entering a custom SQL query. However, you only enter the contents of the WHERE clause, not the entire query. When you perform an outer join, the Integration Service may insert the join syntax in the WHERE clause or the FROM clause of the query, depending on the database syntax.

When you add a user-defined join, the Source Qualifier transformation includes the setting in the default SQL query. However, if you modify the default query after adding a user-defined join, the Integration Service uses only the query defined in the SQL Query property of the Source Qualifier transformation.

You can include mapping parameters and variables in a user-defined join. When including a string mapping parameter or variable, use a string identifier appropriate to the source system. For most databases, you should enclose the name of a string parameter or variable in single quotes. For more information, see the database documentation.

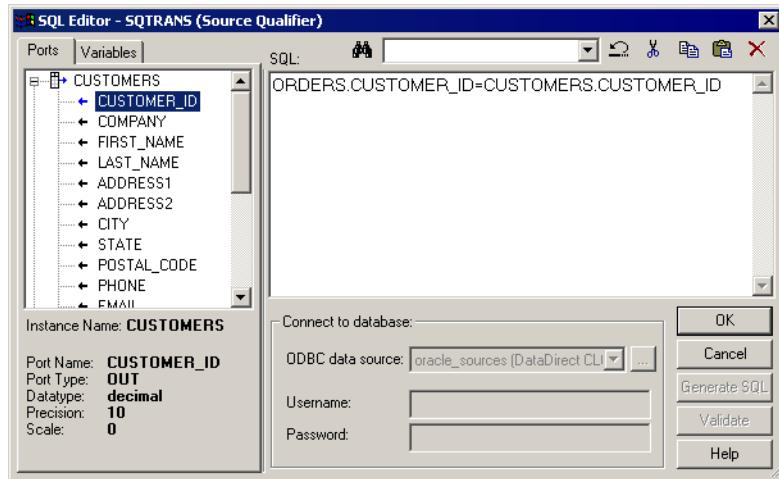
When you include a datetime parameter or variable, you might need to change the date format to match the format used by the source. The Integration Service converts a datetime parameter and variable to a string based on the source system. For more information about automatic date conversion, see Table 21-1 on page 427.

To create a user-defined join:

- 1.** Create a Source Qualifier transformation containing data from multiple sources or associated sources.
- 2.** Open the Source Qualifier transformation, and click the Properties tab.
- 3.** Click the Open button in the User Defined Join field. The SQL Editor dialog box appears.
- 4.** Enter the syntax for the join.

Do not enter the keyword WHERE at the beginning of the join. The Integration Service adds this keyword when it queries rows.

Enclose string mapping parameters and variables in string identifiers. Alter the date format for datetime mapping parameters and variables when necessary.



5. Click OK to return to the Edit Transformations dialog box, and then click OK to return to the Designer.
6. Click Repository > Save.

Outer Join Support

Use the Source Qualifier and the Application Source Qualifier transformations to perform an outer join of two sources in the same database. When the Integration Service performs an outer join, it returns all rows from one source table and rows from the second source table that match the join condition.

Use an outer join when you want to join two tables and return all rows from one of the tables. For example, you might perform an outer join when you want to join a table of registered customers with a monthly purchases table to determine registered customer activity. Using an outer join, you can join the registered customer table with the monthly purchases table and return all rows in the registered customer table, including customers who did not make purchases in the last month. If you perform a normal join, the Integration Service returns only registered customers who made purchases during the month, and only purchases made by registered customers.

With an outer join, you can generate the same results as a master outer or detail outer join in the Joiner transformation. However, when you use an outer join, you reduce the number of rows in the data flow. This can improve performance.

The Integration Service supports two kinds of outer joins:

- ♦ **Left.** Integration Service returns all rows for the table to the left of the join syntax and the rows from both tables that meet the join condition.
- ♦ **Right.** Integration Service returns all rows for the table to the right of the join syntax and the rows from both tables that meet the join condition.

Note: Use outer joins in nested query statements when you override the default query.

Informatica Join Syntax

When you enter join syntax, use the Informatica or database-specific join syntax. When you use the Informatica join syntax, the Integration Service translates the syntax and passes it to the source database during the session.

Note: Always use database-specific syntax for join conditions.

When you use Informatica join syntax, enclose the entire join statement in braces ({Informatica syntax}). When you use database syntax, enter syntax supported by the source database without braces.

When using Informatica join syntax, use table names to prefix column names. For example, if you have a column named FIRST_NAME in the REG_CUSTOMER table, enter “REG_CUSTOMER.FIRST_NAME” in the join syntax. Also, when using an alias for a table name, use the alias within the Informatica join syntax to ensure the Integration Service recognizes the alias.

Table 21-3 lists the join syntax you can enter, in different locations for different Source Qualifier transformations, when you create an outer join:

Table 21-3. Locations for Entering Outer Join Syntax

Transformation	Transformation Setting	Description
Source Qualifier Transformation	User-Defined Join	Create a join override. The Integration Service appends the join override to the WHERE or FROM clause of the default query.
	SQL Query	Enter join syntax immediately after the WHERE in the default query.
Application Source Qualifier Transformation	Join Override	Create a join override. The Integration Service appends the join override to the WHERE clause of the default query.
	Extract Override	Enter join syntax immediately after the WHERE in the default query.

You can combine left outer and right outer joins with normal joins in a single source qualifier. Use multiple normal joins and multiple left outer joins.

When you combine joins, enter them in the following order:

1. Normal
2. Left outer
3. Right outer

Note: Some databases limit you to using one right outer join.

Normal Join Syntax

You can create a normal join using the join condition in a source qualifier. However, if you are creating an outer join, you need to override the default join to perform an outer join. As a result, you need to include the normal join in the join override. When incorporating a normal join in the join override, list the normal join before outer joins. You can enter multiple normal joins in the join override.

To create a normal join, use the following syntax:

```
{ source1 INNER JOIN source2 on join_condition }
```

Table 21-4 displays the syntax for Normal Joins in a Join Override:

Table 21-4. Syntax for Normal Joins in a Join Override

Syntax	Description
source1	Source table name. The Integration Service returns rows from this table that match the join condition.

Table 21-4. Syntax for Normal Joins in a Join Override

Syntax	Description
<code>source2</code>	Source table name. The Integration Service returns rows from this table that match the join condition.
<code>join_condition</code>	Condition for the join. Use syntax supported by the source database. You can combine multiple join conditions with the AND operator.

For example, you have a REG_CUSTOMER table with data for registered customers:

CUST_ID	FIRST_NAME	LAST_NAME
00001	Marvin	Chi
00002	Dinah	Jones
00003	John	Bowden
00004	J.	Marks

The PURCHASES table, refreshed monthly, contains the following data:

TRANSACTION_NO	CUST_ID	DATE	AMOUNT
06-2000-0001	00002	6/3/2000	55.79
06-2000-0002	00002	6/10/2000	104.45
06-2000-0003	00001	6/10/2000	255.56
06-2000-0004	00004	6/15/2000	534.95
06-2000-0005	00002	6/21/2000	98.65
06-2000-0006	NULL	6/23/2000	155.65
06-2000-0007	NULL	6/24/2000	325.45

To return rows displaying customer names for each transaction in the month of June, use the following syntax:

```
{ REG_CUSTOMER INNER JOIN PURCHASES on REG_CUSTOMER.CUST_ID =  
PURCHASES.CUST_ID }
```

The Integration Service returns the following data:

CUST_ID	DATE	AMOUNT	FIRST_NAME	LAST_NAME
00002	6/3/2000	55.79	Dinah	Jones
00002	6/10/2000	104.45	Dinah	Jones
00001	6/10/2000	255.56	Marvin	Chi
00004	6/15/2000	534.95	J.	Marks
00002	6/21/2000	98.65	Dinah	Jones

The Integration Service returns rows with matching customer IDs. It does not include customers who made no purchases in June. It also does not include purchases made by non-registered customers.

Left Outer Join Syntax

You can create a left outer join with a join override. You can enter multiple left outer joins in a single join override. When using left outer joins with other joins, list all left outer joins together, after any normal joins in the statement.

To create a left outer join, use the following syntax:

```
{ source1 LEFT OUTER JOIN source2 on join_condition }
```

Table 21-5 displays syntax for left outer joins in a join override:

Table 21-5. Syntax for Left Outer Joins in a Join Override

Syntax	Description
source1	Source table name. With a left outer join, the Integration Service returns all rows in this table.
source2	Source table name. The Integration Service returns rows from this table that match the join condition.
join_condition	Condition for the join. Use syntax supported by the source database. You can combine multiple join conditions with the AND operator.

For example, using the same REG_CUSTOMER and PURCHASES tables described in “Normal Join Syntax” on page 443, you can determine how many customers bought something in June with the following join override:

```
{ REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID =  
PURCHASES.CUST_ID }
```

The Integration Service returns the following data:

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00001	Marvin	Chi	6/10/2000	255.56
00002	Dinah	Jones	6/3/2000	55.79
00003	John	Bowden	NULL	NULL
00004	J.	Marks	6/15/2000	534.95
00002	Dinah	Jones	6/10/2000	104.45
00002	Dinah	Jones	6/21/2000	98.65

The Integration Service returns all registered customers in the REG_CUSTOMERS table, using null values for the customer who made no purchases in June. It does not include purchases made by non-registered customers.

Use multiple join conditions to determine how many registered customers spent more than \$100.00 in a single purchase in June:

```
{REG_CUSTOMER LEFT OUTER JOIN PURCHASES on (REG_CUSTOMER.CUST_ID =  
PURCHASES.CUST_ID AND PURCHASES.AMOUNT > 100.00) }
```

The Integration Service returns the following data:

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00001	Marvin	Chi	6/10/2000	255.56
00002	Dinah	Jones	6/10/2000	104.45
00003	John	Bowden	NULL	NULL
00004	J.	Marks	6/15/2000	534.95

You might use multiple left outer joins if you want to incorporate information about returns during the same time period. For example, the RETURNS table contains the following data:

CUST_ID	CUST_ID	RETURN
00002	6/10/2000	55.79
00002	6/21/2000	104.45

To determine how many customers made purchases and returns for the month of June, use two left outer joins:

```
{ REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID =  
PURCHASES.CUST_ID LEFT OUTER JOIN RETURNS on REG_CUSTOMER.CUST_ID =  
PURCHASES.CUST_ID }
```

The Integration Service returns the following data:

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT	RET_DATE	RETURN
00001	Marvin	Chi	6/10/2000	255.56	NULL	NULL
00002	Dinah	Jones	6/3/2000	55.79	NULL	NULL
00003	John	Bowden	NULL	NULL	NULL	NULL
00004	J.	Marks	6/15/2000	534.95	NULL	NULL
00002	Dinah	Jones	6/10/2000	104.45	NULL	NULL
00002	Dinah	Jones	6/21/2000	98.65	NULL	NULL
00002	Dinah	Jones	NULL	NULL	6/10/2000	55.79
00002	Dinah	Jones	NULL	NULL	6/21/2000	104.45

The Integration Service uses NULLs for missing values.

Right Outer Join Syntax

You can create a right outer join with a join override. The right outer join returns the same results as a left outer join if you reverse the order of the tables in the join syntax. Use only one right outer join in a join override. If you want to create more than one right outer join, try reversing the order of the source tables and changing the join types to left outer joins.

When you use a right outer join with other joins, enter the right outer join at the end of the join override.

To create a right outer join, use the following syntax:

```
{ source1 RIGHT OUTER JOIN source2 on join_condition }
```

Table 21-6 displays syntax for right outer joins in a join override:

Table 21-6. Syntax for Right Outer Joins in a Join Override

Syntax	Description
<i>source1</i>	Source table name. The Integration Service returns rows from this table that match the join condition.
<i>source2</i>	Source table name. With a right outer join, the Integration Service returns all rows in this table.
<i>join_condition</i>	Condition for the join. Use syntax supported by the source database. You can combine multiple join conditions with the AND operator.

You might use a right outer join with a left outer join to join and return all data from both tables, simulating a full outer join. For example, you can extract all registered customers and all purchases for the month of June with the following join override:

```
{REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID =  
PURCHASES.CUST_ID RIGHT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID =  
PURCHASES.CUST_ID }
```

The Integration Service returns the following data:

CUST_ID	FIRST_NAME	LAST_NAME	TRANSACTION_NO	DATE	AMOUNT
00001	Marvin	Chi	06-2000-0003	6/10/2000	255.56
00002	Dinah	Jones	06-2000-0001	6/3/2000	55.79
00003	John	Bowden	NULL	NULL	NULL
00004	J.	Marks	06-2000-0004	6/15/2000	534.95
00002	Dinah	Jones	06-2000-0002	6/10/2000	104.45
00002	Dinah	Jones	06-2000-0005	6/21/2000	98.65
NULL	NULL	NULL	06-2000-0006	6/23/2000	155.65
NULL	NULL	NULL	06-2000-0007	6/24/2000	325.45

Creating an Outer Join

You can enter an outer join as a join override or as part of an override of the default query.

When you create a join override, the Designer appends the join override to the WHERE clause of the default query. During the session, the Integration Service translates the Informatica join syntax and includes it in the default query used to extract source data. When possible, enter a join override instead of overriding the default query.

When you override the default query, enter the join syntax in the WHERE clause of the default query. During the session, the Integration Service translates Informatica join syntax and then uses the query to extract source data. If you make changes to the transformation

after creating the override, the Integration Service ignores the changes. Therefore, when possible, enter outer join syntax as a join override.

To create an outer join as a join override:

1. Open the Source Qualifier transformation, and click the Properties tab.
2. In a Source Qualifier transformation, click the button in the User Defined Join field. In an Application Source Qualifier transformation, click the button in the Join Override field.
3. Enter the syntax for the join.

Do not enter WHERE at the beginning of the join. The Integration Service adds this when querying rows.

Enclose Informatica join syntax in braces ({}).

When using an alias for a table and the Informatica join syntax, use the alias within the Informatica join syntax.

Use table names to prefix columns names, for example, “table.column”.

Use join conditions supported by the source database.

When entering multiple joins, group joins together by type, and then list them in the following order: normal, left outer, right outer. Include only one right outer join per nested query.

Select port names from the Ports tab to ensure accuracy.

4. Click OK.

To create an outer join as an extract override:

1. After connecting the input and output ports for the Application Source Qualifier transformation, double-click the title bar of the transformation and select the Properties tab.
2. In an Application Source Qualifier transformation, click the button in the Extract Override field.
3. Click Generate SQL.
4. Enter the syntax for the join in the WHERE clause immediately after the WHERE.

Enclose Informatica join syntax in braces ({}).

When using an alias for a table and the Informatica join syntax, use the alias within the Informatica join syntax.

Use table names to prefix columns names, for example, “table.column”.

Use join conditions supported by the source database.

When entering multiple joins, group joins together by type, and then list them in the following order: normal, left outer, right outer. Include only one right outer join per nested query.

Select port names from the Ports tab to ensure accuracy.

5. Click OK.

Common Database Syntax Restrictions

Different databases have different restrictions on outer join syntax. Consider the following restrictions when you create outer joins:

- ◆ Do not combine join conditions with the OR operator in the ON clause of outer join syntax.
- ◆ Do not use the IN operator to compare columns in the ON clause of outer join syntax.
- ◆ Do not compare a column to a subquery in the ON clause of outer join syntax.
- ◆ When combining two or more outer joins, do not use the same table as the inner table of more than one outer join. For example, do not use either of the following outer joins:

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNNA = TABLE2.COLUMNNA TABLE3  
LEFT OUTER JOIN TABLE2 ON TABLE3.COLUMNNB = TABLE2.COLUMNNB }
```

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNNA = TABLE2.COLUMNNA TABLE2  
RIGHT OUTER JOIN TABLE3 ON TABLE2.COLUMNNB = TABLE3.COLUMNNB }
```

- ◆ Do not use both tables of an outer join in a regular join condition. For example, do not use the following join condition:

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNNA = TABLE2.COLUMNNA WHERE  
TABLE1.COLUMNNB = TABLE2.COLUMNNC }
```

However, use both tables in a filter condition, like the following:

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNNA = TABLE2.COLUMNNA WHERE  
TABLE1.COLUMNNB = 32 AND TABLE2.COLUMNNC > 0 }
```

Note: Entering a condition in the ON clause might return different results from entering the same condition in the WHERE clause.

- ◆ When using an alias for a table, use the alias to prefix columns in the table. For example, if you call the REG_CUSTOMER table C, when referencing the column FIRST_NAME, use “C.FIRST_NAME”.

For more information, see the database documentation.

Entering a Source Filter

You can enter a source filter to reduce the number of rows the Integration Service queries. If you include the string 'WHERE' or large objects in the source filter, the Integration Service fails the session.

The Source Qualifier transformation includes source filters in the default SQL query. If, however, you modify the default query after adding a source filter, the Integration Service uses only the query defined in the SQL query portion of the Source Qualifier transformation.

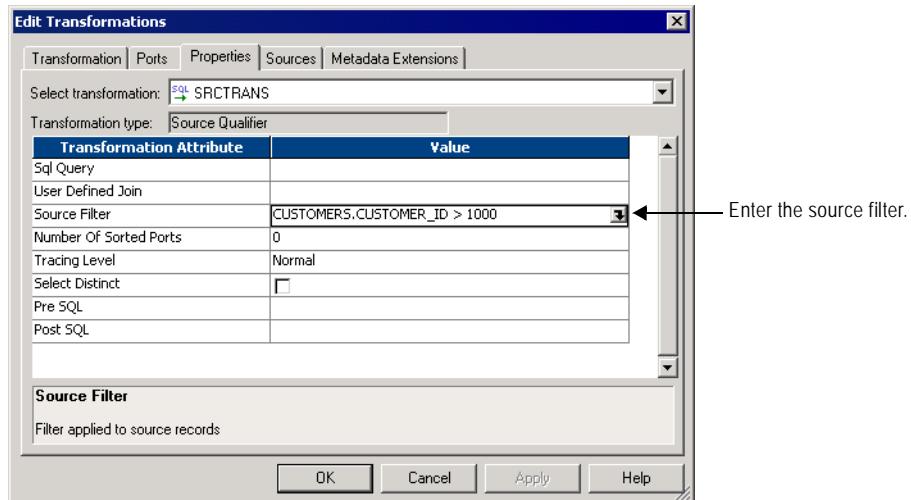
You can include mapping parameters and variables in a source filter. When including a string mapping parameter or variable, use a string identifier appropriate to the source system. For most databases, you should enclose the name of a string parameter or variable in single quotes. For more information, see the database documentation.

When you include a datetime parameter or variable, you might need to change the date format to match the format used by the source. The Integration Service converts a datetime parameter and variable to a string based on the source system. For information about date conversion, see Table 21-1 on page 427.

Note: When you enter a source filter in the session properties, you override the customized SQL query in the Source Qualifier transformation.

To enter a source filter:

1. In the Mapping Designer, open a Source Qualifier transformation.
The Edit Transformations dialog box appears.
2. Select the Properties tab.
3. Click the Open button in the Source Filter field.



4. In the SQL Editor dialog box, enter the filter.

Include the table name and port name. Do not include the keyword WHERE in the filter.

Enclose string mapping parameters and variables in string identifiers. Alter the date format for datetime mapping parameters and variables when necessary.

5. Click OK.

Using Sorted Ports

When you use sorted ports, the Integration Service adds the ports to the ORDER BY clause in the default query. The Integration Service adds the configured number of ports, starting at the top of the Source Qualifier transformation. You might use sorted ports to improve performance when you include any of the following transformations in a mapping:

- ♦ **Aggregator.** When you configure an Aggregator transformation for sorted input, you can send sorted data by using sorted ports. The group by ports in the Aggregator transformation must match the order of the sorted ports in the Source Qualifier transformation. For more information about using a sorted Aggregator transformation, see “Using Sorted Input” on page 45.
- ♦ **Joiner.** When you configure a Joiner transformation for sorted input, you can send sorted data by using sorted ports. Configure the order of the sorted ports the same in each Source Qualifier transformation. For more information about using a sorted Joiner transformation, see “Using Sorted Input” on page 292.

Note: You can also use the Sorter transformation to sort relational and flat file data before Aggregator and Joiner transformations. For more information about sorting data using the Sorter transformation, see “Sorter Transformation” on page 415.

Use sorted ports for relational sources only. When using sorted ports, the sort order of the source database must match the sort order configured for the session. The Integration Service creates the SQL query used to extract source data, including the ORDER BY clause for sorted ports. The database server performs the query and passes the resulting data to the Integration Service. To ensure data is sorted as the Integration Service requires, the database sort order must be the same as the user-defined session sort order.

When you configure the Integration Service for data code page validation and run a workflow in Unicode data movement mode, the Integration Service uses the selected sort order to sort character data.

When you configure the Integration Service for relaxed data code page validation, the Integration Service uses the selected sort order to sort all character data that falls in the language range of the selected sort order. The Integration Service sorts all character data outside the language range of the selected sort order according to standard Unicode sort ordering.

When the Integration Service runs in ASCII mode, it ignores this setting and sorts all character data using a binary sort order. The default sort order depends on the code page of the Integration Service.

The Source Qualifier transformation includes the number of sorted ports in the default SQL query. However, if you modify the default query after choosing the Number of Sorted Ports, the Integration Service uses only the query defined in the SQL Query property.

To use sorted ports:

1. In the Mapping Designer, open a Source Qualifier transformation, and click the Properties tab.
2. Click in Number of Sorted Ports and enter the number of ports you want to sort.

The Integration Service adds the configured number of columns to an ORDER BY clause, starting from the top of the Source Qualifier transformation.

The source database sort order must correspond to the session sort order.

Tip: Sybase supports a maximum of 16 columns in an ORDER BY. If the source is Sybase, do not sort more than 16 columns.

3. Click OK.

Select Distinct

If you want the Integration Service to select unique values from a source, use the Select Distinct option. You might use this feature to extract unique customer IDs from a table listing total sales. Using Select Distinct filters out unnecessary data earlier in the data flow, which might improve performance.

By default, the Designer generates a SELECT statement. If you choose Select Distinct, the Source Qualifier transformation includes the setting in the default SQL query.

For example, in the Source Qualifier transformation in Figure 21-2 on page 435, you enable the Select Distinct option. The Designer adds SELECT DISTINCT to the default query as follows:

```
SELECT DISTINCT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY,  
CUSTOMERS.FIRST_NAME, CUSTOMERS.LAST_NAME, CUSTOMERS.ADDRESS1,  
CUSTOMERS.ADDRESS2, CUSTOMERS.CITY, CUSTOMERS.STATE,  
CUSTOMERS.POSTAL_CODE, CUSTOMERS.EMAIL, ORDERS.ORDER_ID,  
ORDERS.DATE_ENTERED, ORDERS.DATE_PROMISED, ORDERS.DATE_SHIPPED,  
ORDERS.EMPLOYEE_ID, ORDERS.CUSTOMER_ID, ORDERS.SALES_TAX_RATE,  
ORDERS.STORE_ID  
  
FROM  
  
CUSTOMERS, ORDERS  
  
WHERE  
  
CUSTOMERS.CUSTOMER_ID=ORDERS.CUSTOMER_ID
```

However, if you modify the default query after choosing Select Distinct, the Integration Service uses only the query defined in the SQL Query property. In other words, the SQL Query overrides the Select Distinct setting.

To use Select Distinct:

1. Open the Source Qualifier transformation in the mapping, and click on the Properties tab.
2. Check Select Distinct, and Click OK.

Overriding Select Distinct in the Session

You can override the transformation level option to Select Distinct when you configure the session in the Workflow Manager.

To override the Select Distinct option:

1. In the Workflow Manager, open the Session task, and click the Mapping tab.
2. Click the Transformations view, and click the Source Qualifier transformation under the Sources node.
3. In the Properties settings, enable Select Distinct, and click OK.

Adding Pre- and Post-Session SQL Commands

You can add pre- and post-session SQL commands on the Properties tab in the Source Qualifier transformation. You might want to use pre-session SQL to write a timestamp row to the source table when a session begins.

The Integration Service runs pre-session SQL commands against the source database before it reads the source. It runs post-session SQL commands against the source database after it writes to the target.

You can override the SQL commands in the Transformations view on the Mapping tab in the session properties. You can also configure the Integration Service to stop or continue when it encounters errors running pre- or post-session SQL commands. For more information about stopping on errors, see “Working with Sessions” in the *Workflow Administration Guide*.

Use the following guidelines when you enter pre- and post-session SQL commands in the Source Qualifier transformation:

- ◆ Use any command that is valid for the database type. However, the Integration Service does not allow nested comments, even though the database might.
- ◆ Use mapping parameters and variables in the source pre- and post-session SQL commands.
- ◆ Use a semicolon (;) to separate multiple statements.
- ◆ The Integration Service ignores semicolons within /*...*/.
- ◆ If you need to use a semicolon outside of comments, you can escape it with a backslash (\). When you escape the semicolon, the Integration Service ignores the backslash, and it does not use the semicolon as a statement separator.
- ◆ The Designer does not validate the SQL.

Note: You can also enter pre- and post-session SQL commands on the Properties tab of the target instance in a mapping.

Creating a Source Qualifier Transformation

You can configure the Designer to create a Source Qualifier transformation by default when you drag a source into a mapping, or you can create a Source Qualifier transformation manually.

Creating a Source Qualifier Transformation By Default

You can configure the Designer to create a Source Qualifier transformation when you drag a source into a mapping.

To create a Source Qualifier transformation automatically:

1. In the Designer, click Tools > Options.
2. Select the Format tab.
3. In the Tools options, select Mapping Designer.
4. Select Create Source Qualifier When Opening Sources.

For more information about configuring Designer options, see “Using the Designer” in the *Designer Guide*.

Creating a Source Qualifier Transformation Manually

You can manually create a Source Qualifier transformation in the Mapping Designer.

To create a Source Qualifier transformation manually:

1. In the Mapping Designer, click Transformation > Create.
2. Enter a name for the transformation, and click Create.
3. Select a source, and click OK.
4. Click Done.

Configuring Source Qualifier Transformation Options

After you create the Source Qualifier transformation, you can configure several options.

To configure a Source Qualifier transformation:

1. In the Designer, open a mapping.
2. Double-click the title bar of the Source Qualifier transformation.
3. In the Edit Transformations dialog box, click Rename, enter a descriptive name for the transformation, and click OK.

The naming convention for Source Qualifier transformations is `SQ_TransformationName`, such as `SQ_AllSources`.

- 4.** Click the Properties tab.
- 5.** Enter the Source Qualifier transformation properties. For information about the Source Qualifier transformation properties, see “Source Qualifier Transformation Properties” on page 429.
- 6.** Click the Sources tab and indicate any associated source definitions you want to define for this transformation.

Identify associated sources only when you need to join data from multiple databases or flat file systems.
- 7.** Click OK to return to the Designer.

Troubleshooting

I cannot perform a drag and drop operation, such as connecting ports.

Review the error message on the status bar for details.

I cannot connect a source definition to a target definition.

You cannot directly connect sources to targets. Instead, you need to connect them through a Source Qualifier transformation for relational and flat file sources, or through a Normalizer transformation for COBOL sources.

I cannot connect multiple sources to one target.

The Designer does not allow you to connect multiple Source Qualifier transformations to a single target. There are two workarounds:

- ◆ **Reuse targets.** Since target definitions are reusable, you can add the same target to the mapping multiple times. Then connect each Source Qualifier transformation to each target.
- ◆ **Join the sources in a Source Qualifier transformation.** Then remove the WHERE clause from the SQL query.

I entered a custom query, but it is not working when I run the workflow containing the session.

Be sure to test this setting for the Source Qualifier transformation before you run the workflow. Return to the Source Qualifier transformation and reopen the dialog box in which you entered the custom query. You can connect to a database and click the Validate button to test the SQL. The Designer displays any errors. Review the session log file if you need further information.

The most common reason a session fails is because the database login in both the session and Source Qualifier transformation is not the table owner. You need to specify the table owner in the session and when you generate the SQL Query in the Source Qualifier transformation.

You can test the SQL Query by cutting and pasting it into the database client tool (such as Oracle Net) to see if it returns an error.

I used a mapping variable in a source filter and now the session fails.

Try testing the query by generating and validating the SQL in the Source Qualifier transformation. If the variable or parameter is a string, you probably need to enclose it in single quotes. If it is a datetime variable or parameter, you might need to change its format for the source system.

Chapter 22

SQL Transformation

This chapter includes the following topics:

- ◆ Overview, 460
- ◆ Script Mode, 461
- ◆ Query Mode, 464
- ◆ Connecting to Databases, 470
- ◆ Session Processing, 474
- ◆ Creating an SQL Transformation, 480
- ◆ SQL Transformation Properties, 482
- ◆ SQL Statements, 488

Overview

Transformation type:

Active/Passive
Connected

The SQL transformation processes SQL queries midstream in a pipeline. You can insert, delete, update, and retrieve rows from a database. You can pass the database connection information to the SQL transformation as input data at run time. The transformation processes external SQL scripts or SQL queries that you create in an SQL editor. The SQL transformation processes the query and returns rows and database errors.

For example, you might need to create database tables before adding new transactions. You can create an SQL transformation to create the tables in a workflow. The SQL transformation returns database errors in an output port. You can configure another workflow to run if the SQL transformation returns no errors.

When you create an SQL transformation, you configure the following options:

- ◆ **Mode.** The SQL transformation runs in one of the following modes:
 - **Script mode.** The SQL transformation runs ANSI SQL scripts that are externally located. You pass a script name to the transformation with each input row. The SQL transformation outputs one row for each input row. For more information about script mode, see “Script Mode” on page 461.
 - **Query mode.** The SQL transformation executes a query that you define in a query editor. You can pass strings or parameters to the query to define dynamic queries or change the selection parameters. You can output multiple rows when the query has a SELECT statement. For more information about query mode, see “Query Mode” on page 464.
- ◆ **Database type.** The type of database the SQL transformation connects to.
- ◆ **Connection type.** Pass database connection information to the SQL transformation or use a connection object. For more information about connection types, see “Connecting to Databases” on page 470.

Script Mode

An SQL transformation running in script mode runs SQL scripts from text files. You pass each script file name from the source to the SQL transformation ScriptName port. The script file name contains the complete path to the script file.

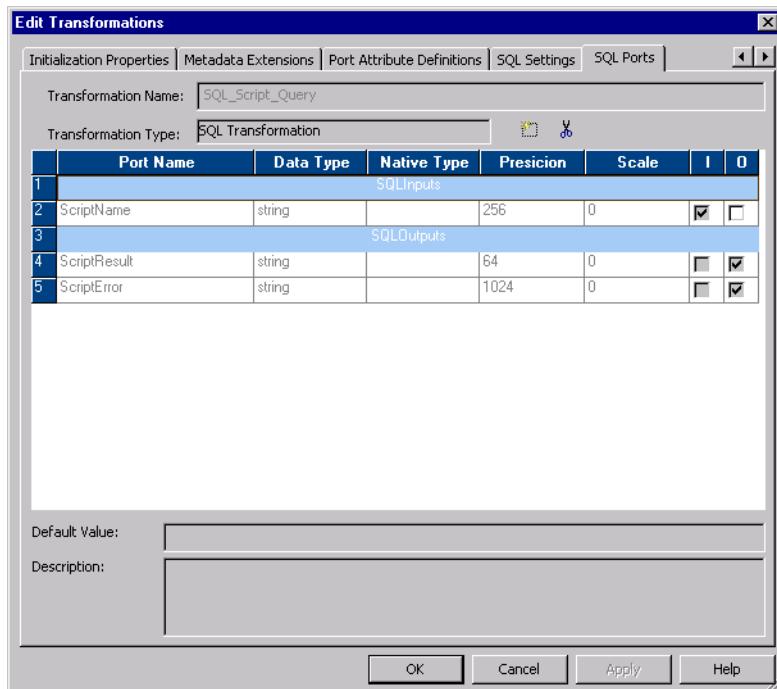
When you configure the transformation to run in script mode, you create a passive transformation. The transformation returns one row for each input row. The output row contains results of the query and any database error.

When the SQL transformation runs in script mode, the query statement and query data do not change. When you need to run different queries in script mode, you pass the scripts in the source data. Use script mode to run data definition queries such as creating or dropping tables.

When you configure an SQL transformation to run in script mode, the Designer adds the ScriptName input port to the transformation. When you create a mapping, you connect the ScriptName port to a port that contains the name of a script to execute for each row. You can execute a different SQL script for each input row. The Designer creates default ports that return information about query results.

Figure 22-1 shows the default ports for an SQL transformation configured to run in script mode:

Figure 22-1. SQL Transformation Script Mode Ports



An SQL transformation configured for script mode has the following default ports:

Port	Type	Description
ScriptName	Input	Receives the name of the script to execute for the current row.
ScriptResult	Output	Returns PASSED if the script execution succeeds for the row. Otherwise contains FAILED.
ScriptError	Output	Returns errors that occur when a script fails for a row.

Example

You need to create order and inventory tables before adding new data to the tables. You can create an SQL script to create the tables and configure an SQL transformation to run the script.

You create a file called `create_order_inventory.txt` that contains the SQL statements to create the tables.

The following mapping shows how to pass the script name to the SQL transformation:



The Integration Service reads a row from the source. The source row contains the SQL script file name and path:

```
C:\81\server\shared\SrcFiles\create_order_inventory.txt
```

The transformation receives the file name in the `ScriptName` port. The Integration Service locates the script file and parses the script. It creates an SQL procedure and sends it to the database to process. The database validates the SQL and executes the query.

The SQL transformation returns the `ScriptResults` and `ScriptError`. If the script executes successfully, the `ScriptResult` output port returns PASSED. Otherwise, the `ScriptResult` port returns FAILED. When the `ScriptResult` is FAILED, the SQL transformation returns error messages in the `ScriptError` port. The SQL transformation returns one row for each input row it receives.

Script Mode Rules and Guidelines

Use the following rules and guidelines for an SQL transformation that runs in script mode:

- ◆ You can use a static or dynamic database connection with script mode. For more information about configuring database connections with the SQL transformation, see "Connecting to Databases" on page 470.

- ◆ To include multiple query statements in a script, you can separate them with a semicolon.
- ◆ You can use mapping variables or parameters in the script file name.
- ◆ The script code page defaults to the locale of the operating system. You can change the locale of the script. For more information about the script locale property, see “SQL Settings Tab” on page 485.
- ◆ The script file must be accessible by the Integration Service. The Integration Service must have read permissions on the directory that contains the script.
- ◆ The Integration Service ignores the output of any SELECT statement you include in the SQL script. The SQL transformation in script mode does not output more than one row of data for each input row.
- ◆ You cannot use scripting languages such as Oracle PL/SQL or Microsoft/Sybase T-SQL in the script.
- ◆ You cannot use nested scripts where the SQL script calls another SQL script.
- ◆ A script cannot accept run-time arguments.

Query Mode

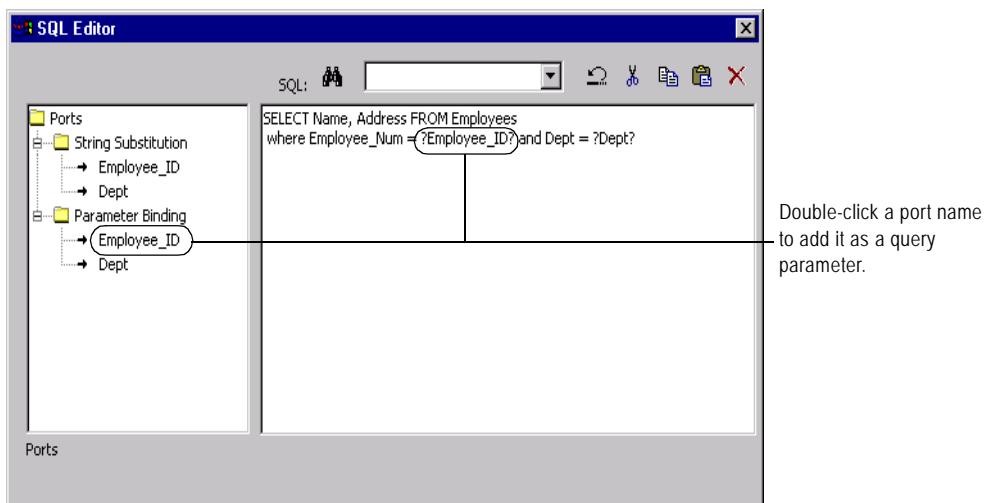
When an SQL transformation runs in query mode, it executes an SQL query that you define in the transformation. You pass strings or parameters to the query from the transformation input ports to change the query statement or the query data.

When you configure the SQL transformation to run in query mode, you create an active transformation. The transformation can return multiple rows for each input row.

Create queries in the SQL transformation SQL Editor. To create a query, type the query statement in the SQL Editor main window. The SQL Editor provides a list of the transformation ports that you can reference in the query.

Figure 22-2 shows an SQL query in the SQL Editor:

Figure 22-2. SQL Editor for an SQL Transformation Query



When you create a query, the SQL Editor validates the port names in the query. It also verifies that the ports you use for string substitution are string datatypes. The SQL Editor does not validate the syntax of the SQL query.

You can create the following types of SQL queries in the SQL transformation:

- ◆ **Static SQL query.** The query statement does not change, but you can use query parameters to change the data. The Integration Service prepares the query once and runs the query for all input rows. For more information about static queries, see “Using Static SQL Queries” on page 465.
- ◆ **Dynamic SQL query.** You can change the query statements and the data. The Integration Service prepares a query for each input row. For more information about dynamic queries, see “Using Dynamic SQL Queries” on page 466.

When you create a static query, the Integration Service prepares the SQL procedure once and executes it for each row. When you create a dynamic query, the Integration Service prepares the SQL for each input row. You can optimize performance by creating static queries.

Using Static SQL Queries

Create a static SQL query when you need to run the same query statements for each input row, but you want to change the data in the query for each input row. When you create a static SQL query, you use parameter binding in the SQL Editor to define parameters for query data.

To change the data in the query, configure query parameters and bind them to input ports in the transformation. When you bind a parameter to an input port, you identify the port by name in the query. The SQL Editor encloses the name in question marks (?). The query data changes based on the value of the data in the input port.

The SQL transformation input ports receive the data for the data values in the query, or the values in the WHERE clause of the query.

The following static queries use parameter binding:

```
DELETE FROM Employee WHERE Dept = ?Dept?  
INSERT INTO Employee(Employee_ID, Dept) VALUES (?Employee_ID?, ?Dept?)  
UPDATE Employee SET Dept = ?Dept? WHERE Employee_ID > 100
```

The following static SQL query has query parameters that bind to the Employee_ID and Dept input ports of an SQL transformation:

```
SELECT Name, Address FROM Employees WHERE Employee_Num =?Employee_ID? and  
Dept = ?Dept?
```

The source might have the following rows:

Employee_ID	Dept
100	Products
123	HR
130	Accounting

The Integration Service generates the following query statements from the rows:

```
SELECT Name, Address FROM Employees WHERE Employee_ID = '100' and DEPT = 'Products'  
SELECT Name, Address FROM Employees WHERE Employee_ID = '123' and DEPT = 'HR'  
SELECT Name, Address FROM Employees WHERE Employee_ID = '130' and DEPT = 'Accounting'
```

Selecting Multiple Database Rows

When the SQL query contains a SELECT statement, the transformation returns one row for each database row it retrieves. You must configure an output port for each column in the SELECT statement.

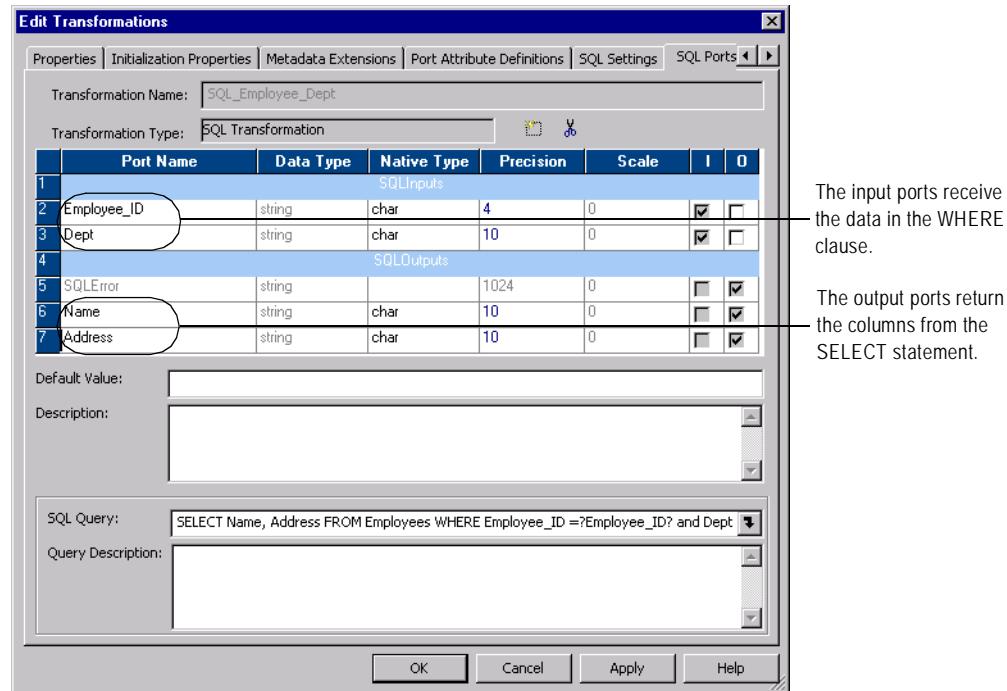
When you configure output ports for database columns, you need to configure the datatype of each database column you select. Select a native datatype from the list. When you select the native datatype, the Designer configures the transformation datatype for you.

The native datatype in the transformation must match the database column datatype. The Integration Service matches the column datatype in the database with the native database type in the transformation at run time. If the datatypes do not match, the Integration Service generates a row error.

For more information about transformation datatypes you can use for each database, see “Datatype Reference” in the *Designer Guide*.

Figure 22-3 shows the ports in the transformation configured to run in query mode:

Figure 22-3. SQL Transformation Static Query Mode Ports



The SQL query selects name and address from the employees table. The SQL transformation writes a row to the target for each database row it retrieves.

Using Dynamic SQL Queries

A dynamic SQL query can execute different query statements for each input row. When you create a dynamic SQL query, you use string substitution to define string parameters in the query and link them to input ports in the transformation.

To change a query statement, configure a string variable in the query for the portion of the query you want to change. To configure the string variable, identify an input port by name in

the query and enclose the name with the tilde (~). The query changes based on the value of the data in the port. The transformation input port that contains the query parameter must be a string datatype. You can use string substitution to change the query statement and the query data.

When you create a dynamic SQL query, the Integration Service prepares a query for each input row. You can pass the full query or pass part of the query in an input port:

- ◆ **Full query.** You can substitute the entire SQL query with query statements from source data. For more information, see “Passing the Full Query” on page 467.
- ◆ **Partial query.** You can substitute a portion of the query statement, such as the table name. For more information, see “Substituting the Table Name in a String” on page 468.

Passing the Full Query

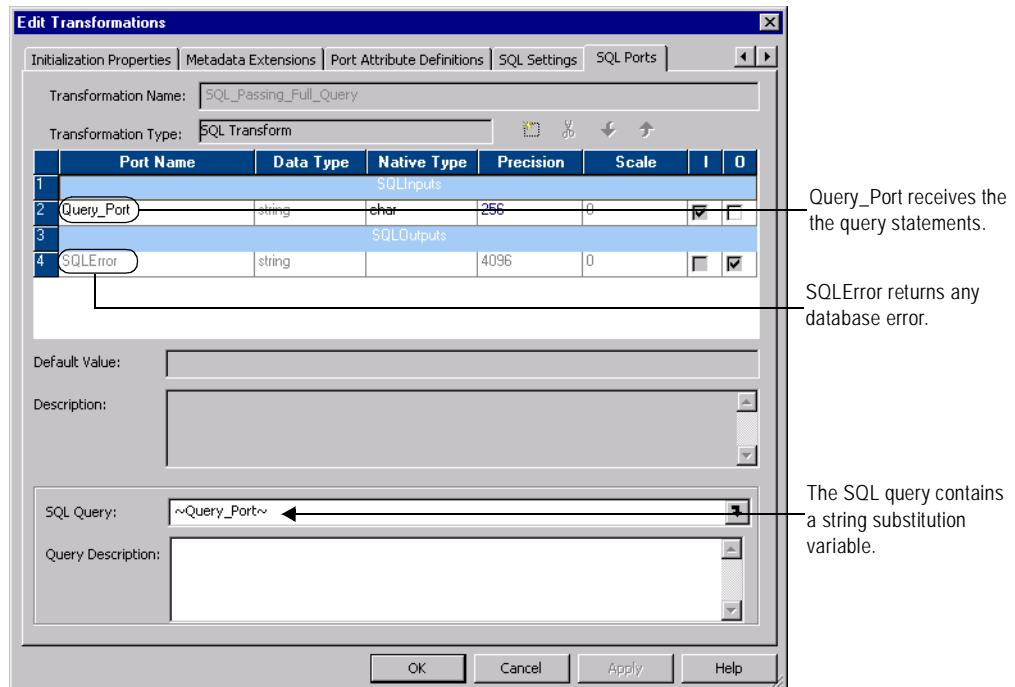
You can pass the full SQL query through an input port in the transformation. To pass the full query, create a query in the SQL Editor that consists of one string variable to represent the full query:

```
~Query_Port~
```

The transformation receives the query in the Query_Port input port.

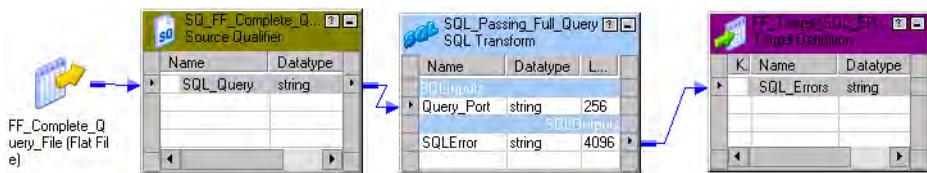
Figure 22-4 shows ports in the SQL transformation:

Figure 22-4. SQL Transformation Ports to Pass a Full Dynamic Query



The Integration Service replaces the `-Query_Port-` variable in the dynamic query with the SQL statements from the source. It prepares the query and sends it to the database to process. The database executes the query. The SQL transformation returns database errors to the `SQL_Error` port.

The following mapping shows how to pass the query to the SQL transformation:



When you pass the full query, you can pass more than one query statement for each input row. For example, the source might contain the following rows:

```

DELETE FROM Person WHERE LastName = 'Jones'; INSERT INTO Person (LastName, Address) VALUES
('Smith', '38 Summit Drive')

DELETE FROM Person WHERE LastName = 'Jones'; INSERT INTO Person (LastName, Address) VALUES
('Smith', '38 Summit Drive')

DELETE FROM Person WHERE LastName = 'Russell';
  
```

You can pass any type of query in the source data. When you configure SELECT statements in the query, you must configure output ports for the database columns you retrieve from the database. When you mix SELECT statements and other types of queries, the output ports that represent database columns contain null values when no database columns are retrieved.

Substituting the Table Name in a String

You can substitute the table name in a query. To substitute the table name, configure an input port to receive the table name from each input row. Identify the input port by name in the query and enclose the name with the tilde (~).

The following dynamic query contains a string variable, `-Table_Port-`:

```
SELECT Emp_ID, Address from ~Table_Port~ where Dept = 'HR'
```

The source might pass the following values to the `Table_Port` column:

Table_Port

```

Employees_USA
Employees_England
Employees_Australia
  
```

The Integration Service replaces the `-Table_Port-` variable with the table name in the input port:

```

SELECT Emp_ID, Address from Employees_USA where Dept = 'HR'
SELECT Emp_ID, Address from Employees_England where Dept = 'HR'
SELECT Emp_ID, Address from Employees_Australia where Dept = 'HR'
  
```

Query Mode Rules and Guidelines

Use the following rules and guidelines when you configure the SQL transformation to run in query mode:

- ◆ The number and the order of the output ports must match the number and order of the fields in the query SELECT clause.
- ◆ The native datatype of an output port in the transformation must match the datatype of the corresponding column in the database. The Integration Service generates a row error when the datatypes do not match.
- ◆ When the SQL query contains an INSERT, UPDATE, or DELETE clause, the transformation returns data to the SQLError port and the NumRowsAffected port when it is enabled. If you add output ports, you can configure default values for them. Otherwise the ports receive NULL data values.
- ◆ You can add pass-through ports to pass data in and out of the transformation. To create a pass-through port, create an input port and enable it for output. The Designer creates an output port and adds an “_output” suffix to the port name.
- ◆ When you drag a port from a Source Qualifier transformation to the SQL transformation, the Designer creates a pass-through port by default. To change the port to input, disable it for output.
- ◆ You cannot add the “_output” suffix to output port names that you create.
- ◆ You cannot use the pass-through port to return data from a SELECT query.
- ◆ When the number of output ports is more than the number of columns in the SELECT clause, the extra ports receive a NULL or default value in a session.
- ◆ When the number of output ports is less than the number of columns in the SELECT clause, the Integration Service generates a row error.
- ◆ You can use string substitution instead of parameter binding in a query. However, the input ports must be string datatypes.

Connecting to Databases

You can use a static database connection or you can pass database connection information to the SQL transformation at run time.

Use one of the following types of connections to connect the SQL transformation to a database:

- ◆ **Static connection.** Configure the connection object in the session. You must first create the connection object in Workflow Manager. For more information, see “Using a Static Database Connection” on page 470.
- ◆ **Logical connection.** Pass a connection name to the SQL transformation as input data at run time. You must first create the connection object in Workflow Manager. For more information about logical connections, see “Passing a Logical Database Connection” on page 470.
- ◆ **Full Database Connection.** Pass the connect string, user name, password, and other connection information to SQL transformation input ports at run time. For more information about passing the full database connection information, see “Passing Full Connection Information” on page 470.

Using a Static Database Connection

You can configure the SQL transformation to connect to a database with a static connection. A static database connection is a database connection defined in the Workflow Manager.

To use a static connection, choose a relational connection object when you configure the session. To avoid datatype conversion errors, use a relational connection for the same database type that is configured in the transformation.

Passing a Logical Database Connection

You can configure the SQL transformation to connect to a database with a logical database connection. A logical database connection is a connection object name that you pass to the transformation at run time. Define the relational connection object in the Workflow Manager. When you configure the transformation to use a logical database connection, the Designer creates the LogicalConnectionObject input port.

You can pass a logical connection for each input row. Configure the mapping to pass the connection object name to the LogicalConnectionObject port. To avoid datatype conversion errors, use a relational connection for the same database type that is configured in the transformation.

Passing Full Connection Information

You can pass all the database connection information to an SQL transformation as input port data. When you configure the SQL transformation to connect to a database with a full

connection, the Designer creates input ports for connection components. The database type defaults to the database type you configured for the transformation.

When you configure an SQL transformation to connect to a database with a full connection, the Designer creates the following ports:

Table 22-1. Full Database Connection Information

Port	Required/ Optional	Description
ConnectionString	Required	Contains the database name and database server name. For information about passing the connect string, see "Passing the Connect String" on page 471.
DBUser	Required	Name of the user with permissions to read and write from the database.
DBPasswd	Required	DBUser password.
CodePage	Optional	Code page the Integration Service uses to read from or write to the database. Use the ISO code page name, such as ISO-8859-6. The code page name is not case sensitive. For a list of supported code page names, see "Code Pages" in the <i>Administrator Guide</i> .
AdvancedOptions	Optional	Connection attributes. Pass the attributes as name-value pairs. Delimit each attribute from another with a semicolon. Attribute names are not case sensitive. For more information about advanced options, see "Passing Advanced Options" on page 472.

Passing the Connect String

The native connect string contains the database name and database server name. The connect string allows PowerCenter and the database client to direct calls to the correct database.

Table 22-2 lists the native connect string syntax for each database:

Table 22-2. Native Connect String Syntax

Database	Connect String Syntax	Example
IBM DB2	<code>dbname</code>	<code>mydatabase</code>
Informix	<code>dbname@servername</code>	<code>mydatabase@informix</code>
Microsoft SQL Server	<code>servername@dbname</code>	<code>sqlserver@mydatabase</code>
Oracle	<code>dbname.world</code> (same as TNSNAMES entry)	<code>oracle.world</code>
Sybase ASE*	<code>servername@dbname</code>	<code>sambrown@mydatabase</code>
Teradata**	<code>ODBC_data_source_name</code> or <code>ODBC_data_source_name@db_name</code> or <code>ODBC_data_source_name@db_user_name</code>	<code>TeradataODBC</code> <code>TeradataODBC@mydatabase</code> <code>TeradataODBC@sambrown</code>

*Sybase ASE `servername` is the name of the Adaptive Server from the interfaces file.

**Use Teradata ODBC drivers to connect to source and target databases.

Passing Advanced Options

You can configure optional connection attributes. To configure the attributes, pass the attributes as name-value pairs. Delimit attributes with a semicolon. Attribute names are not case sensitive.

For example, you might pass the following string to configure connection options:

```
Use_Trusted_Connection = 1; Connection_Retry_Period = 5
```

You can configure the following advanced options:

Attribute	Database Type	Description
Connection Retry Period	All	<p>Integer.</p> <p>The number of seconds the Integration Service attempts to reconnect to the database if the connection fails.</p>
Data Source Name	Teradata	<p>String.</p> <p>The name of the Teradata ODBC data source.</p>
Database Name	Sybase ASE Microsoft SQL Server Teradata	<p>String.</p> <p>Override the default database name in the ODBC. If you do not enter a database name, messages related to the connection do not show a database name.</p>
Domain Name	Microsoft SQL Server	<p>String.</p> <p>The name of the domain where Microsoft SQL Server is running.</p>
Enable Parallel Mode	Oracle	<p>Integer.</p> <p>Enable parallel processing when you load data in bulk mode.</p> <p>0 is not enabled.</p> <p>1 is enabled.</p> <p>Default is enabled.</p>
Owner Name	All	<p>String.</p> <p>The table owner name.</p>
Packet Size	Sybase ASE Microsoft SQL Server	<p>Integer.</p> <p>Optimize the ODBC connection to Sybase ASE and Microsoft SQL Server.</p>
Server Name	Sybase ASE Microsoft SQL Server	<p>String.</p> <p>The name of the database server.</p>
Use Trusted Connection	Microsoft SQL Server	<p>Integer.</p> <p>When enabled, the Integration Service uses Windows authentication to access the Microsoft SQL Server database.</p> <p>The user name that starts the Integration Service must be a valid Windows user with access to the Microsoft SQL Server database.</p> <p>0 is not enabled.</p> <p>1 is enabled.</p>

Database Connections Rules and Guidelines

Use the following rules and guidelines when configuring database connections for the SQL transformation:

- ◆ You need the PowerCenter license key to connect different database types. A session fails if PowerCenter is not licensed to connect to the database.
- ◆ To improve performance, use a static database connection. When you configure a dynamic connection, the Integration Service establishes a new connection for each input row.
- ◆ When you have a limited number of connections to use in a session, you can configure multiple SQL transformations. Configure each SQL transformation to use a different static connection. Use a Router transformation to route rows to a SQL transformation based on connectivity information in the row.
- ◆ When you configure the SQL transformation to use full connection data, the database password is plain text. You can pass logical connections when you have a limited number of connections you need to use in a session. A logical connection provides the same functionality as the full connection, and the database password is secure.
- ◆ When you pass logical database connections to the SQL transformation, the Integration Service accesses the repository to retrieve the connection information for each input row. When you have many rows to process, passing logical database connections might have a performance impact.

Session Processing

When the Integration Service processes an SQL transformation, it runs SQL queries midstream in the pipeline. When a SELECT query retrieves database rows, the SQL transformation returns the database columns in the output ports. For other types of queries, the SQL transformation returns query results or database errors in output ports.

The SQL transformation configured to run in script mode always returns one row for each input row. A SQL transformation that runs in query mode can return a different number of rows for each input row. The number of rows the SQL transformation returns is based on the type of query it runs and the success of the query. For more information, see “Input Row to Output Row Cardinality” on page 474.

You can use transaction control with the SQL transformation when you configure the transformation to use a static database connection. You can also issue commit and rollback statements in the query. For more information, see “Transaction Control” on page 478.

The SQL transformation provides database connection resiliency. However, you cannot recover an SQL transformation with a resume from last checkpoint recovery strategy. For more information, see “High Availability” on page 478.

Input Row to Output Row Cardinality

When the Integration Service runs a SELECT query, the SQL transformation returns a row for each row it retrieves. When the query does not retrieve data, the SQL transformation returns zero or one row for each input row.

The SQL transformation returns a number of output rows based on the following factors:

- ◆ **Query statement processing.** When the query contains a SELECT statement, the Integration Service can retrieve multiple output rows.
- ◆ **NumRowsAffected port configuration.** The NumRowsAffected output port contains the total number of rows affected by updates, inserts, or deletes for one input row.
- ◆ **Query results.** When a SELECT query is successful, the SQL transformation might retrieve multiple rows. When the query contains other statements, the Integration Service might generate a row that contains SQL errors or the number of rows affected.
- ◆ **The maximum row count configuration.** The Max Output Row Count limits the number of rows the SQL transformation returns from SELECT queries.

Query Statement Processing

The type of query determines how many rows the SQL transformation returns. The SQL transformation running in query mode can return zero, one, or multiple rows. When the query contains a SELECT statement, the SQL transformation returns each column from the database to an output port. The transformation returns all qualifying rows.

Table 22-3 lists the output rows the SQL transformation generates for different types of query statements when no errors occur in query mode:

Table 22-3. Output Rows By Query Statement - Query Mode

Query Statement	Output Rows
UPDATE, INSERT, DELETE only	Zero rows.
One or more SELECT statements	Total number of database rows retrieved.
DDL queries such as CREATE, DROP, TRUNCATE	Zero rows.

Number of Rows Affected

You can enable the NumRowsAffected output port to return the number of rows affected by the INSERT, UPDATE, or DELETE query statements for each input row. When you have multiple statements in a query, the NumRowsAffected port contains the sum of all rows that the query changed. NumRowsAffected is disabled by default.

When you enable NumRowsAffected in query mode, and the SQL query does not contain an INSERT, UPDATE, or DELETE statement, NumRowsAffected is zero in each output row.

Note: When you enable NumRowsAffected and the transformation is configured to run in script mode, NumRowsAffected is always NULL.

Table 22-4 lists the output rows the SQL transformation generates when you enable NumRowsAffected in query mode:

Table 22-4. NumRowsAffected Rows by Query Statement - Query Mode

Query Statement	Output Rows
UPDATE, INSERT, DELETE only	One row with the NumRowsAffected total.
One or more SELECT statements	Total number of database rows retrieved. NumRowsAffected is zero in each row.
DDL queries such as CREATE, DROP, TRUNCATE	One row with zero NumRowsAffected.

When the SQL transformation runs in query mode and a query contains multiple statements, the Integration Service returns the NumRowsAffected sum in the last row it returns for an input row. NumRowsAffected contains the sum of the rows affected by all INSERT, UPDATE, and DELETE statements from an input row.

For example, a query contains the following statements:

```
DELETE from Employees WHERE Employee_ID = '101';
SELECT Employee_ID, LastName from Employees WHERE Employee_ID = '103';
INSERT into Employees (Employee_ID, LastName, Address)VALUES ('102',
'Gein', '38 Beach Rd')
```

The DELETE statement affects one row. The SELECT statement does not affect any row. The INSERT statement affects one row. The value of NumRowsAffected is two. The Integration Service returns this value in the last output row it returns for the input row.

The Integration Service returns no output rows from the DELETE statement. It returns one row from the SELECT statement. It returns one row from the INSERT statement that contains the NumRowsAffected total.

The NumRowsAffected port returns zero when all of the following conditions are true:

- ◆ The database is Informix.
- ◆ The transformation is running in query mode.
- ◆ The query contains no parameters.

Maximum Output Row Count

You can limit the number of rows the SQL transformation returns for SELECT queries. Configure the Max Output Row Count property to limit number of rows. When a query contains multiple SELECT statements, the SQL transformation limits total rows from all the SELECT statements.

For example, you set Max Output Row Count to 100. The query contains two SELECT statements:

```
SELECT * FROM table1; SELECT * FROM table2;
```

If the first SELECT statement returns 200 rows, and the second SELECT statement returns 50 rows, the SQL transformation returns 100 rows from the first SELECT statement. It returns no rows from the second statement.

To configure unlimited output rows, set Max Output Row Count to zero.

Understanding Error Rows

The Integration Service returns row errors when it encounters a connection error or syntax error. The SQL transformation has the following default ports to output error text:

- ◆ **SQLError.** Returns database errors when the SQL transformation runs in query mode.
- ◆ **ScriptError.** Returns database errors when the SQL transformation runs in script mode.

When the SQL query contains syntax errors, the error port contains the error text from the database. For example, the following SQL query generates a row error from an Oracle database:

```
SELECT Product_ID FROM Employees
```

The Employees table does not contain Product_ID. The Integration Service generates one row. The SQLError port contains the error text in one line:

```
ORA-0094: "Product_ID": invalid identifier Database driver error...
Function Name: Execute SQL Stmt: SELECT Product_ID from Employees Oracle
Fatal Error
```

When a query contains multiple statements, and you configure the SQL transformation to continue on SQL error, the SQL transformation might return rows from the database for one query statement, but return database errors for another query statement. The SQL transformation returns any database error in a separate row. For more information about continuing on SQL errors, see “Continuing on SQL Error” on page 477.

Table 22-5 lists the rows the SQL transformation generates based on whether the queries have errors or not:

Table 22-5. Output Rows by Query Statement - Query Mode

Query Statement	SQLError	NumRowsAffected	Rows Output
UPDATE, INSERT, DELETE only	No	- Disabled - Enabled	- Zero rows. - One row with the NumRowsAffected column.
	Yes	- Disabled - Enabled	- One row with errors in the SQLError port. - One row with errors in the SQLError port and the number of rows affected in the NumRowsAffected column.
One or more SELECT statements	No	- Disabled - Enabled	- Zero or more rows, based on what rows are returned for each SELECT statement. - One or more rows, based on what rows are returned for each SELECT statement. Each row contains a NumRowsAffected column with a value zero.
	Yes	- Disabled - Enabled	- The total number of rows is one greater than the sum of the output rows for the successful statements. The last row contains the errors in the SQLError port. - Each row contains a NumRowsAffected column with a value zero.
DDL queries such as CREATE, DROP, TRUNCATE	No	- Disabled - Enabled	- Zero rows. - One row that includes the NumRowsAffected column with a value zero.
	Yes	- Disabled - Enabled	- One row that contains errors in the SQLError port. - One row that contains errors in the SQLError port and a NumRowsAffected column with a value zero.

Continuing on SQL Error

You can choose to ignore an SQL error in a statement by enabling the Continue on SQL Error within a Row option. The Integration Service continues to run the rest of the SQL statements for the row. The Integration Service does not generate a row error. However, the SQLError port contains the failed SQL statement and error messages. A session fails when the row error count exceeds the session error threshold.

For example, a query might have the following statements:

```
DELETE FROM Persons WHERE FirstName = 'Ed';
INSERT INTO Persons (LastName, Address)VALUES ('Gein', '38 Beach Rd')
```

If the DELETE statement fails, the SQL transformation returns an error message from the database. The Integration Service continues processing the INSERT statement.

Tip: Disable the Continue on SQL Error option to debug database errors. Otherwise you might not be able to associate errors with the query statements that caused them.

Transaction Control

An SQL transformation that runs in script mode drops any incoming transaction boundary from an upstream source or transaction generator. The Integration Service issues a commit after executing the script for each input row in the SQL transformation. The transaction contains the set of rows affected by the script.

An SQL transformation that runs in query mode commits transactions at different points based on the database connection type:

- ◆ **Dynamic database connection.** The Integration Service issues a commit after executing the SQL for each input row. The transaction is the set of rows affected by the script. You cannot use a Transaction Control transformation with dynamic connections in query mode.
- ◆ **Static connection.** The Integration Service issues a commit after processing all the input rows. The transaction includes all the database rows to update. You can override the default behavior by using a Transaction Control transformation to control the transaction, or by using commit and rollback statements in the SQL query.

When you configure an SQL statement to commit or rollback rows, configure the SQL transformation to generate transactions with the Generate Transaction transformation property. Configure the session for user-defined commit.

For more information about transaction control, see “Understanding Commit Points” in the *Workflow Administration Guide*.

The following transaction control SQL statements are not valid with the SQL transformation:

- ◆ **SAVEPOINT.** Identifies a rollback point in the transaction.
- ◆ **SET TRANSACTION.** Changes transaction options.

High Availability

When you have high availability, the SQL transformation provides database connection resiliency for static and dynamic connections. When the Integration Service fails to connect to the database, it retries the connection. You can configure the Connection Retry Period for a connection. When the Integration Service cannot connect to the database in the time period that you configure, it generates a row error for a dynamic connection or fails the session for a static connection.

The Integration Service is not resilient to temporary network failures or relational database unavailability when the Integration Service retrieves data from a database for the SQL transformation. When a connection fails during processing, the session fails. The Integration Service cannot reconnect and continue processing from the last commit point.

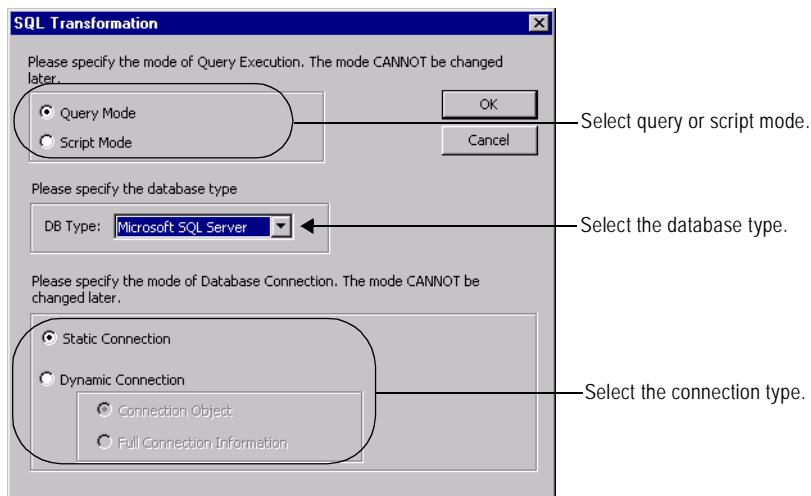
You cannot resume a session from the last checkpoint when it contains the SQL transformation. When the Integration Service resumes a session, the recovery session must produce the same data as the original session. The SQL transformation cannot produce repeatable data between session runs. You can recover the workflow when you configure the session to restart.

Creating an SQL Transformation

You can create an SQL transformation in the Transformation Developer or the Mapping Designer.

To create an SQL transformation:

1. Click Transformation > Create.
2. Select the SQL transformation.
3. Enter a name for the transformation. The naming convention for an SQL transformation is SQL_TransformationName. Enter a description for the transformation and click Create.



4. Configure the SQL transformation mode:
 - ◆ **Query mode.** Configure an active transformation that executes dynamic SQL queries.
 - ◆ **Script mode.** Configure a passive transformation that executes external SQL scripts.
5. Configure the database type that the SQL transformation connects to. Choose the database type from the list.
6. Configure the SQL transformation connection options.

Figure 22-6 lists the SQL transformation connection options:

Table 22-6. SQL Transformation Connection Options

Option	Description
Static Connection	Use a connection object that you configure for the session. The SQL transformation connects to the database once during a session.
Dynamic Connection	Connect to databases depending on connection information you pass to the transformation in a mapping. When you configure a dynamic connection, choose whether the transformation requires a connection object name or the transformation requires all the connection information. Default is connection object.
Connection Object	Dynamic connection only. Use a LogicalConnectionObject port to receive the connection object name. The connection object must be defined in the Workflow Manager connections.
Full Connection Information	Dynamic connection only. Use input ports to receive all the connection components.

7. Click OK to configure the transformation.

The Designer creates default ports in the transformation depending on the options you choose. You cannot change the configuration except for the database type. For more information about the default SQL transformation input and output ports, see “SQL Ports Tab” on page 434.

SQL Transformation Properties

After you create the SQL transformation, you can define ports and set attributes in the following transformation tabs:

- ◆ **Ports.** Displays the transformation ports and attributes that you create on the SQL Ports tab.
- ◆ **Properties.** SQL Transformation general properties. For more information, see “Properties Tab” on page 482.
- ◆ **Initialization Properties.** Run-time properties that the transformation uses during initialization. For more information about creating initialization properties, see “Working with Procedure Properties” on page 72.
- ◆ **Metadata Extensions.** Property name and value pairs you can use with a procedure when the Integration Service runs the procedure. For more information about creating metadata extensions, see “Metadata Extensions” in the *Repository Guide*.
- ◆ **Port Attribute Definitions.** User-defined port attributes that apply to all ports in the transformation. For more information about port attributes, see “Working with Port Attributes” on page 62.
- ◆ **SQL Setting.** Attributes unique to the SQL transformation. For more information, see “SQL Settings Tab” on page 485.
- ◆ **SQL Ports.** SQL transformation ports and attributes. For more information, see “SQL Ports Tab” on page 486.

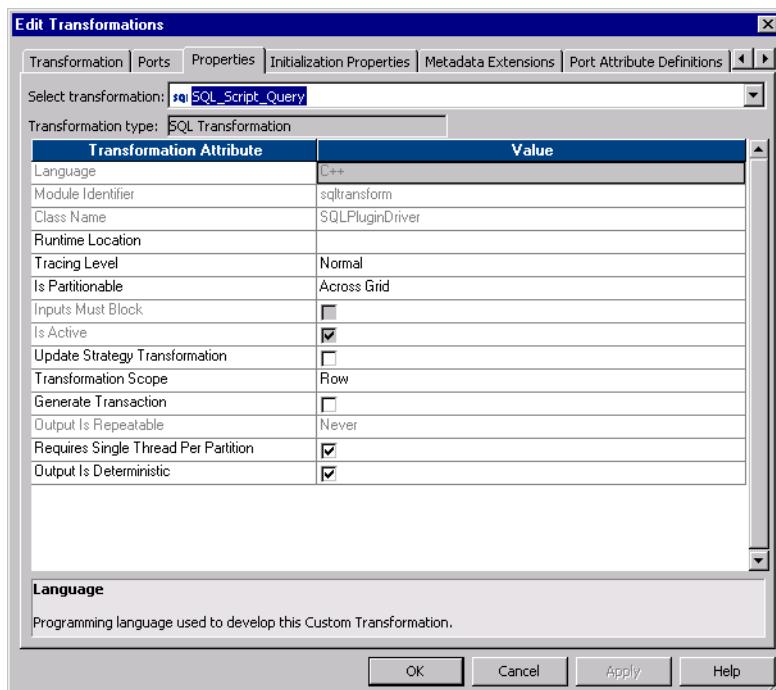
Note: You cannot update the columns on the Ports tab. When you define ports on the SQL Ports tab, they display on the Ports tab.

Properties Tab

Configure the SQL transformation general properties on the Properties tab. Some transformation properties do not apply to the SQL transformation or are not configurable.

Figure 22-5 shows the SQL transformation Properties tab:

Figure 22-5. SQL Transformation Properties Tab



You can configure the following SQL transformation properties:

Table 22-7. SQL Transformation Properties

Property	Required/ Optional	Description
RunTime Location	Optional	Location that contains the DLL or shared library. Enter a path relative to the Integration Service machine that runs the session using the SQL transformation. If this property is blank, the Integration Service uses the environment variable defined on the Integration Service machine to locate the DLL or shared library. You must copy all DLLs or shared libraries to the run-time location or to the environment variable defined on the Integration Service machine. The Integration Service fails to load the procedure when it cannot locate the DLL, shared library, or a referenced file.
Tracing Level	Required	Sets the amount of detail included in the session log when you run a session containing this transformation. When you configure the SQL transformation tracing level to Verbose Data, the Integration Service writes each SQL query it prepares to the session log. For more information, see "Configuring Tracing Level in Transformations" on page 30.

Table 22-7. SQL Transformation Properties

Property	Required/ Optional	Description
IsPartitionable	Required	<p>Multiple partitions in a pipeline can use this transformation. Use the following options:</p> <ul style="list-style-type: none">- No. The transformation cannot be partitioned. The transformation and other transformations in the same pipeline are limited to one partition. You might choose No if the transformation processes all the input data together, such as data cleansing.- Locally. The transformation can be partitioned, but the Integration Service must run all partitions in the pipeline on the same node. Choose Locally when different partitions of the transformation must share objects in memory.- Across Grid. The transformation can be partitioned, and the Integration Service can distribute each partition to different nodes. <p>Default is No.</p> <p>For more information about using partitioning with transformations, see "Working with Partition Points" in the <i>Workflow Administration Guide</i>.</p>
Update Strategy Transformation	Optional	<p>The transformation defines the update strategy for output rows. You can enable this property for query mode SQL transformations.</p> <p>Default is disabled.</p>
Transformation Scope	Required	<p>The method in which the Integration Service applies the transformation logic to incoming data. Use the following options:</p> <ul style="list-style-type: none">- Row- Transaction- All Input <p>Set transaction scope to transaction when you use transaction control in static query mode.</p> <p>Default is Row for script mode transformations.</p> <p>Default is All Input for query mode transformations.</p>
Generate Transaction	Optional	<p>The transformation generates transaction rows. Enable this property for query mode SQL transformations that commit data in an SQL query.</p> <p>Default is disabled.</p>
Requires Single Thread Per Partition	Optional	<p>Indicates if the Integration Service processes each partition of a procedure with one thread. When you enable this option, the procedure code can use thread-specific operations. Default is enabled.</p> <p>For more information about writing thread-specific operations, see "Working with Thread-Specific Procedure Code" on page 66.</p>
Output is Deterministic	Optional	<p>The transformation generates consistent output data between session runs.</p>

SQL Settings Tab

Configure SQL transformation attributes on the SQL Settings tab. The SQL attributes are unique to the SQL transformation.

Figure 22-6 shows the SQL Settings tab attributes:

Figure 22-6. SQL Settings Tab

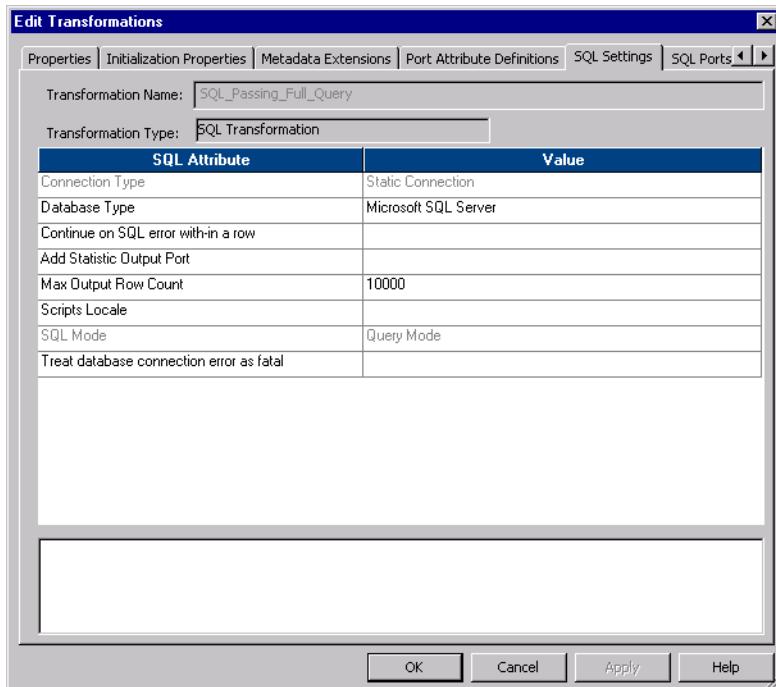


Table 22-8 lists the attributes you can configure on the SQL Setting tab:

Table 22-8. SQL Settings Tab Attributes

Option	Description
Continue on SQL Error within row	Continues processing the remaining SQL statements in a query after an SQL error occurs.
Add Statistic Output Port	Adds a NumRowsAffected output port. The port returns the total number of database rows affected by INSERT, DELETE, and UPDATE query statements for an input row.
Max Output Row Count	Defines the maximum number of rows the SQL transformation can output from a SELECT query. To configure unlimited rows, set Max Output Row Count to zero.
Scripts Locale	Identifies the code page for a SQL script. Choose the code page from the list. Default is operating system locale.

SQL Ports Tab

When you create an SQL transformation, the Designer adds default ports depending on how you configure the transformation. After you create the transformation, you can add ports to the transformation on the SQL Ports tab.

Figure 22-7 shows the ports for a query mode transformation that uses a dynamic database connection:

Figure 22-7. SQL Transformation SQL Ports Tab

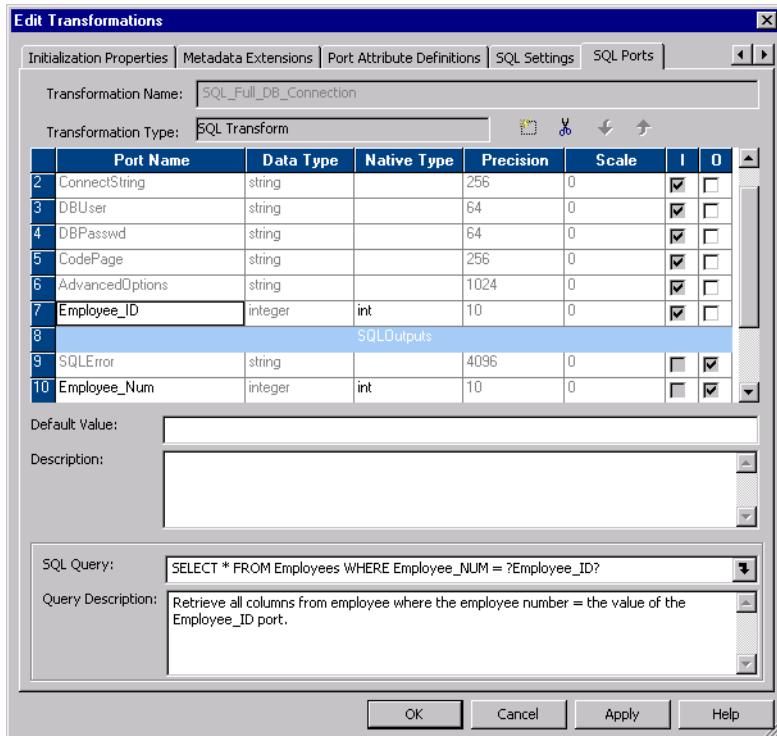


Table 22-9 lists the SQL transformation ports:

Table 22-9. SQL Transformation Ports

Port	Type	Mode	Description
ScriptName	Input	Script	The name of the script to execute for each row.
ScriptError	Output	Script	SQL errors the database passes back to the transformation when the script fails.
ScriptResult	Output	Script	Results from query execution. Contains PASSED when the query executes successfully. Contains FAILED when the query is not successful.
SQLLError	Output	Query	SQL errors the database passes back to the transformation.

Table 22-9. SQL Transformation Ports

Port	Type	Mode	Description
LogicalConnectionObject	Input	Query	Dynamic connection. The name of a connection defined in Workflow Manager connections.
Connect String	Input	Query Script	Connection object only. The database name and the database server name.
DBUser	Input	Query Script	Full connection only. The name of the user with permissions to read and write from the database.
DBPasswd	Input	Query Script	Full connection only. The database password.
CodePage	Input	Query Script	Full connection only. The code page the Integration Service uses to read from or write to the database.
Advanced Options	Input	Query Script	Full connection only. Optional connection attributes such as packet size, connection retry period, and enable parallel mode.
NumRowsAffected	Output	Query Script	The total number of database rows affected by INSERT, DELETE, and UPDATE query statements for an input row.

SQL Statements

Table 22-10 lists the statements you can use with the SQL transformation:

Table 22-10. Standard SQL Statements

Statement Type	Statement	Description
Data Definition	ALTER	Modifies the structure of the database.
	COMMENT	Adds comments to the data dictionary.
	CREATE	Creates a database, table, or index.
	DROP	Deletes an index, table, or database.
	RENAME	Renames a database object.
	TRUNCATE	Removes all rows from a table.
Data Manipulation	CALL	Calls a PL/SQL or Java subprogram.
	DELETE	Deletes rows from a table.
	EXPLAIN PLAN	Writes the access plan for a statement into the database Explain tables.
	INSERT	Inserts row into a table.
	LOCK TABLE	Prevents concurrent application processes from using or changing a table.
	MERGE	Updates a table with source data.
Data Control Language	SELECT	Retrieves data from the database.
	UPDATE	Updates the values of rows of a table.
	GRANT	Grants privileges to a database user.
	REVOKE	Removes access privileges for a database user.
Transaction Control	COMMIT	Saves a unit of work and performs the database changes for that unit of work.
	ROLLBACK	Reverses changes to the database since the last COMMIT.

Chapter 23

Stored Procedure Transformation

This chapter includes the following topics:

- ◆ Overview, 490
- ◆ Using a Stored Procedure in a Mapping, 494
- ◆ Writing a Stored Procedure, 495
- ◆ Creating a Stored Procedure Transformation, 498
- ◆ Configuring a Connected Transformation, 504
- ◆ Configuring an Unconnected Transformation, 506
- ◆ Error Handling, 512
- ◆ Supported Databases, 514
- ◆ Expression Rules, 516
- ◆ Tips, 517
- ◆ Troubleshooting, 518

Overview

Transformation type:

Passive

Connected/Unconnected

A Stored Procedure transformation is an important tool for populating and maintaining databases. Database administrators create stored procedures to automate tasks that are too complicated for standard SQL statements.

A stored procedure is a precompiled collection of Transact-SQL, PL-SQL or other database procedural statements and optional flow control statements, similar to an executable script. Stored procedures are stored and run within the database. You can run a stored procedure with the EXECUTE SQL statement in a database client tool, just as you can run SQL statements. Unlike standard SQL, however, stored procedures allow user-defined variables, conditional statements, and other powerful programming features.

Not all databases support stored procedures, and stored procedure syntax varies depending on the database. You might use stored procedures to complete the following tasks:

- ◆ Check the status of a target database before loading data into it.
- ◆ Determine if enough space exists in a database.
- ◆ Perform a specialized calculation.
- ◆ Drop and recreate indexes.

Database developers and programmers use stored procedures for various tasks within databases, since stored procedures allow greater flexibility than SQL statements. Stored procedures also provide error handling and logging necessary for critical tasks. Developers create stored procedures in the database using the client tools provided with the database.

The stored procedure must exist in the database before creating a Stored Procedure transformation, and the stored procedure can exist in a source, target, or any database with a valid connection to the Integration Service.

You might use a stored procedure to perform a query or calculation that you would otherwise make part of a mapping. For example, if you already have a well-tested stored procedure for calculating sales tax, you can perform that calculation through the stored procedure instead of recreating the same calculation in an Expression transformation.

Input and Output Data

One of the most useful features of stored procedures is the ability to send data to the stored procedure, and receive data from the stored procedure. There are three types of data that pass between the Integration Service and the stored procedure:

- ◆ Input/output parameters
- ◆ Return values
- ◆ Status codes

Some limitations exist on passing data, depending on the database implementation, which are discussed throughout this chapter. Additionally, not all stored procedures send and receive data. For example, if you write a stored procedure to rebuild a database index at the end of a session, you cannot receive data, since the session has already finished.

Input/Output Parameters

For many stored procedures, you provide a value and receive a value in return. These values are known as input and output parameters. For example, a sales tax calculation stored procedure can take a single input parameter, such as the price of an item. After performing the calculation, the stored procedure returns two output parameters, the amount of tax, and the total cost of the item including the tax.

The Stored Procedure transformation sends and receives input and output parameters using ports, variables, or by entering a value in an expression, such as 10 or SALES.

Return Values

Most databases provide a return value after running a stored procedure. Depending on the database implementation, this value can either be user-definable, which means that it can act similar to a single output parameter, or it may only return an integer value.

The Stored Procedure transformation captures return values in a similar manner as input/output parameters, depending on the method that the input/output parameters are captured. In some instances, only a parameter or a return value can be captured.

If a stored procedure returns a result set rather than a single return value, the Stored Procedure transformation takes only the first value returned from the procedure.

Note: An Oracle stored function is similar to an Oracle stored procedure, except that the stored function supports output parameters or return values. In this chapter, any statements regarding stored procedures also apply to stored functions, unless otherwise noted.

Status Codes

Status codes provide error handling for the Integration Service during a workflow. The stored procedure issues a status code that notifies whether or not the stored procedure completed successfully. You cannot see this value. The Integration Service uses it to determine whether to continue running the session or stop. You configure options in the Workflow Manager to continue or stop the session in the event of a stored procedure error.

Connected and Unconnected

Stored procedures run in either connected or unconnected mode. The mode you use depends on what the stored procedure does and how you plan to use it in a session. You can configure connected and unconnected Stored Procedure transformations in a mapping.

- ◆ **Connected.** The flow of data through a mapping in connected mode also passes through the Stored Procedure transformation. All data entering the transformation through the input ports affects the stored procedure. You should use a connected Stored Procedure

transformation when you need data from an input port sent as an input parameter to the stored procedure, or the results of a stored procedure sent as an output parameter to another transformation.

- ♦ **Unconnected.** The unconnected Stored Procedure transformation is not connected directly to the flow of the mapping. It either runs before or after the session, or is called by an expression in another transformation in the mapping.

Table 23-1 compares connected and unconnected transformations:

Table 23-1. Comparison of Connected and Unconnected Stored Procedure Transformations

If you want to	Use this mode
Run a stored procedure before or after a session.	Unconnected
Run a stored procedure once during a mapping, such as pre- or post-session.	Unconnected
Run a stored procedure every time a row passes through the Stored Procedure transformation.	Connected or Unconnected
Run a stored procedure based on data that passes through the mapping, such as when a specific port does not contain a null value.	Unconnected
Pass parameters to the stored procedure and receive a single output parameter.	Connected or Unconnected
Pass parameters to the stored procedure and receive multiple output parameters. Note: To get multiple output parameters from an unconnected Stored Procedure transformation, you must create variables for each output parameter. For more information, see "Calling a Stored Procedure From an Expression" on page 506.	Connected or Unconnected
Run nested stored procedures.	Unconnected
Call multiple times within a mapping.	Unconnected

For more information, see “Configuring a Connected Transformation” on page 504 and “Configuring an Unconnected Transformation” on page 506.

Specifying when the Stored Procedure Runs

In addition to specifying the mode of the Stored Procedure transformation, you also specify when it runs. In the case of the unconnected stored procedure above, the Expression transformation references the stored procedure, which means the stored procedure runs every time a row passes through the Expression transformation. However, if no transformation references the Stored Procedure transformation, you have the option to run the stored procedure once before or after the session.

The following list describes the options for running a Stored Procedure transformation:

- ♦ **Normal.** The stored procedure runs where the transformation exists in the mapping on a row-by-row basis. This is useful for calling the stored procedure for each row of data that passes through the mapping, such as running a calculation against an input port. Connected stored procedures run only in normal mode.

- ◆ **Pre-load of the Source.** Before the session retrieves data from the source, the stored procedure runs. This is useful for verifying the existence of tables or performing joins of data in a temporary table.
- ◆ **Post-load of the Source.** After the session retrieves data from the source, the stored procedure runs. This is useful for removing temporary tables.
- ◆ **Pre-load of the Target.** Before the session sends data to the target, the stored procedure runs. This is useful for verifying target tables or disk space on the target system.
- ◆ **Post-load of the Target.** After the session sends data to the target, the stored procedure runs. This is useful for re-creating indexes on the database.

You can run several Stored Procedure transformations in different modes in the same mapping. For example, a pre-load source stored procedure can check table integrity, a normal stored procedure can populate the table, and a post-load stored procedure can rebuild indexes in the database. However, you cannot run the same instance of a Stored Procedure transformation in both connected and unconnected mode in a mapping. You must create different instances of the transformation.

If the mapping calls more than one source or target pre- or post-load stored procedure in a mapping, the Integration Service executes the stored procedures in the execution order that you specify in the mapping.

The Integration Service executes each stored procedure using the database connection you specify in the transformation properties. The Integration Service opens the database connection when it encounters the first stored procedure. The database connection remains open until the Integration Service finishes processing all stored procedures for that connection. The Integration Service closes the database connections and opens a new one when it encounters a stored procedure using a different database connection.

To run multiple stored procedures that use the same database connection, set these stored procedures to run consecutively. If you do not set them to run consecutively, you might have unexpected results in the target. For example, you have two stored procedures: Stored Procedure A and Stored Procedure B. Stored Procedure A begins a transaction, and Stored Procedure B commits the transaction. If you run Stored Procedure C before Stored Procedure B, using another database connection, Stored Procedure B cannot commit the transaction because the Integration Service closes the database connection when it runs Stored Procedure C.

Use the following guidelines to run multiple stored procedures within a database connection:

- ◆ The stored procedures use the same database connect string defined in the stored procedure properties.
- ◆ You set the stored procedures to run in consecutive order.
- ◆ The stored procedures have the same stored procedure type:
 - Source pre-load
 - Source post-load
 - Target pre-load
 - Target post-load

Using a Stored Procedure in a Mapping

You must perform several steps to use a Stored Procedure transformation in a mapping. Since the stored procedure exists in the database, you must configure not only the mapping and session, but the stored procedure in the database as well. The following sections in this chapter detail each of the following steps.

To use a **Stored Procedure** transformation:

1. Create the stored procedure in the database.

Before using the Designer to create the transformation, you must create the stored procedure in the database. You should also test the stored procedure through the provided database client tools.

2. Import or create the Stored Procedure transformation.

Use the Designer to import or create the Stored Procedure transformation, providing ports for any necessary input/output and return values.

3. Determine whether to use the transformation as connected or unconnected.

You must determine how the stored procedure relates to the mapping before configuring the transformation.

4. If connected, map the appropriate input and output ports.

You use connected Stored Procedure transformations just as you would most other transformations. Drag the appropriate input flow ports to the transformation, and create mappings from output ports to other transformations.

5. If unconnected, either configure the stored procedure to run pre- or post-session, or configure it to run from an expression in another transformation.

Since stored procedures can run before or after the session, you may need to specify when the unconnected transformation should run. On the other hand, if the stored procedure is called from another transformation, you write the expression in another transformation that calls the stored procedure. The expression can contain variables, and may or may not include a return value.

6. Configure the session.

The session properties in the Workflow Manager includes options for error handling when running stored procedures and several SQL override options.

Writing a Stored Procedure

You write SQL statements to create a stored procedure in the database, and you can add other Transact-SQL statements and database-specific functions. These can include user-defined datatypes and execution order statements. For more information, see the database documentation.

Sample Stored Procedure

In the following example, the source database has a stored procedure that takes an input parameter of an employee ID number, and returns an output parameter of the employee name. In addition, a return value of 0 is returned as a notification that the stored procedure completed successfully. The database table that contains employee IDs and names appears as follows:

Employee ID	Employee Name
101	Bill Takash
102	Louis Li
103	Sarah Ferguson

The stored procedure receives the employee ID 101 as an input parameter, and returns the name Bill Takash. Depending on how the mapping calls this stored procedure, any or all of the IDs may be passed to the stored procedure.

Since the syntax varies between databases, the SQL statements to create this stored procedure may vary. The client tools used to pass the SQL statements to the database also vary. Most databases provide a set of client tools, including a standard SQL editor. Some databases, such as Microsoft SQL Server, provide tools that create some of the initial SQL statements.

In all cases, consult the database documentation for more detailed descriptions and examples.

Note: The Integration Service fails sessions that contain stored procedure arguments with large objects.

Informix

In Informix, the syntax for declaring an output parameter differs from other databases. With most databases, you declare variables using IN or OUT to specify if the variable acts as an input or output parameter. Informix uses the keyword RETURNING, making it difficult to distinguish input/output parameters from return values. For example, you use the RETURN command to return one or more output parameters:

```
CREATE PROCEDURE GET_NAME_USING_ID (nID integer)
RETURNING varchar(20);
define nID integer;
define outVAR as varchar(20);
```

```

SELECT FIRST_NAME INTO outVAR FROM CONTACT WHERE ID = nID
return outVAR;
END PROCEDURE;

```

Notice that in this case, the RETURN statement passes the value of outVAR. Unlike other databases, however, outVAR is not a return value, but an output parameter. Multiple output parameters would be returned in the following manner:

```
return outVAR1, outVAR2, outVAR3
```

Informix does pass a return value. The return value is not user-defined, but generated as an error-checking value. In the transformation, the R value must be checked.

Oracle

In Oracle, any stored procedure that returns a value is called a stored function. Rather than using the CREATE PROCEDURE statement to make a new stored procedure based on the example, you use the CREATE FUNCTION statement. In this sample, the variables are declared as IN and OUT, but Oracle also supports an INOUT parameter type, which lets you pass in a parameter, modify it, and return the modified value:

```

CREATE OR REPLACE FUNCTION GET_NAME_USING_ID (
    nID IN NUMBER,
    outVAR OUT VARCHAR2)
RETURN VARCHAR2 IS
    RETURN_VAR varchar2(100);
BEGIN
    SELECT FIRST_NAME INTO outVAR FROM CONTACT WHERE ID = nID;
    RETURN_VAR := 'Success';
    RETURN (RETURN_VAR);
END;
/

```

Notice that the return value is a string value (Success) with the datatype VARCHAR2. Oracle is the only database to allow return values with string datatypes.

Sybase ASE/Microsoft SQL Server

Sybase and Microsoft implement stored procedures identically, as the following syntax shows:

```

CREATE PROCEDURE GET_NAME_USING_ID @nID int = 1, @outVar varchar(20)
OUTPUT
AS
SELECT @outVar = FIRST_NAME FROM CONTACT WHERE ID = @nID
return 0

```

Notice that the return value does not need to be a variable. In this case, if the SELECT statement is successful, a 0 is returned as the return value.

IBM DB2

The following text is an example of an SQL stored procedure on IBM DB2:

```
CREATE PROCEDURE get_name_using_id ( IN id_in int,
                                     OUT emp_out char(18),
                                     OUT sqlcode_out int)

LANGUAGE SQL

P1: BEGIN

-- Declare variables

DECLARE SQLCODE INT DEFAULT 0;
DECLARE emp_TMP char(18) DEFAULT ' ';
-- Declare handler

DECLARE EXIT HANDLER FOR SQLEXCEPTION
    SET SQLCODE_OUT = SQLCODE;

select employee into emp_TMP
    from doc_employee
    where id = id_in;

SET emp_out      = EMP_TMP;
SET sqlcode_out = SQLCODE;

END P1
```

Teradata

The following text is an example of an SQL stored procedure on Teradata. It takes an employee ID number as an input parameter and returns the employee name as an output parameter:

```
CREATE PROCEDURE GET_NAME_USING_ID (IN nID integer, OUT outVAR
varchar(40))

BEGIN

SELECT FIRST_NAME INTO :outVAR FROM CONTACT where ID = :nID;

END;
```

Creating a Stored Procedure Transformation

After you configure and test a stored procedure in the database, you must create the Stored Procedure transformation in the Mapping Designer. There are two ways to configure the Stored Procedure transformation:

- ◆ Use the Import Stored Procedure dialog box to configure the ports used by the stored procedure.
- ◆ Configure the transformation manually, creating the appropriate ports for any input or output parameters.

Stored Procedure transformations are created as Normal type by default, which means that they run during the mapping, not before or after the session.

New Stored Procedure transformations are not created as reusable transformations. To create a reusable transformation, click Make Reusable in the Transformation properties after creating the transformation.

Note: Configure the properties of reusable transformations in the Transformation Developer, not the Mapping Designer, to make changes globally for the transformation.

Importing Stored Procedures

When you import a stored procedure, the Designer creates ports based on the stored procedure input and output parameters. You should import the stored procedure whenever possible.

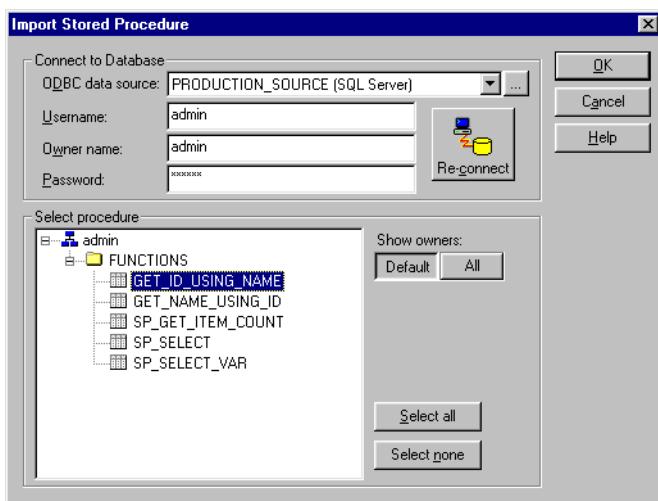
There are three ways to import a stored procedure in the Mapping Designer:

- ◆ Select the stored procedure icon and add a Stored Procedure transformation.
- ◆ Click Transformation > Import Stored Procedure.
- ◆ Click Transformation > Create, and then select Stored Procedure.

Note: When you import a stored procedure containing a period (.) in the stored procedure name, the Designer substitutes an underscore (_) for the period in the Stored Procedure transformation name.

To import a stored procedure:

1. In the Mapping Designer, click Transformation > Import Stored Procedure.
2. Select the database that contains the stored procedure from the list of ODBC sources. Enter the user name, owner name, and password to connect to the database and click Connect.

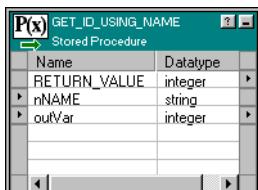


Notice the folder in the dialog box displays FUNCTIONS. The stored procedures listed in this folder contain input parameters, output parameters, or a return value. If stored procedures exist in the database that do not contain parameters or return values, they appear in a folder called PROCEDURES. This applies primarily to Oracle stored procedures. For a normal connected Stored Procedure to appear in the functions list, it requires at least one input and one output port.

Tip: You can select Skip to add a Stored Procedure transformation without importing the stored procedure. In this case, you need to manually add the ports and connect information within the transformation. For more information, see “Manually Creating Stored Procedure Transformations” on page 500.

3. Select the procedure to import and click OK.

The Stored Procedure transformation appears in the mapping. The Stored Procedure transformation name is the same as the stored procedure you selected. If the stored procedure contains input parameters, output parameters, or a return value, you see the appropriate ports that match each parameter or return value in the Stored Procedure transformation.



In this Stored Procedure transformation, you can see that the stored procedure contains the following value and parameters:

- ◆ An integer return value, called RETURN_VALUE, with an output port.
- ◆ A string input parameter, called nNAME, with an input port.
- ◆ An integer output parameter, called outVar, with an input and output port.

Note: If you change the transformation name, you need to configure the name of the stored procedure in the transformation properties. If you have multiple instances of the same stored procedure in a mapping, you must also configure the name of the stored procedure.

4. Open the transformation, and click the Properties tab.

Select the database where the stored procedure exists from the Connection Information row. If you changed the name of the Stored Procedure transformation to something other than the name of the stored procedure, enter the Stored Procedure Name.

5. Click OK.

6. Click Repository > Save to save changes to the mapping.

Manually Creating Stored Procedure Transformations

To create a Stored Procedure transformation manually, you need to know the input parameters, output parameters, and return values of the stored procedure, if there are any. You must also know the datatypes of those parameters, and the name of the stored procedure. All these are configured through Import Stored Procedure.

To create a Stored Procedure transformation:

1. In the Mapping Designer, click Transformation > Create, and then select Stored Procedure.

The naming convention for a Stored Procedure transformation is the name of the stored procedure, which happens automatically. If you change the transformation name, then you need to configure the name of the stored procedure in the Transformation Properties. If you have multiple instances of the same stored procedure in a mapping, you must perform this step.

2. Click Skip.

The Stored Procedure transformation appears in the Mapping Designer.

3. Open the transformation, and click the Ports tab.

You must create ports based on the input parameters, output parameters, and return values in the stored procedure. Create a port in the Stored Procedure transformation for each of the following stored procedure parameters:

- ◆ An integer input parameter
- ◆ A string output parameter
- ◆ A return value

For the integer input parameter, you would create an integer input port. The parameter and the port must be the same datatype and precision. Repeat this for the output parameter and the return value.

The R column should be selected and the output port for the return value. For stored procedures with multiple parameters, you must list the ports in the same order that they appear in the stored procedure.

4. Click the Properties tab.

Enter the name of the stored procedure in the Stored Procedure Name row, and select the database where the stored procedure exists from the Connection Information row.

5. Click OK.

6. Click Repository > Save to save changes to the mapping.

Although the repository validates and saves the mapping, the Designer does not validate the manually entered Stored Procedure transformation. No checks are completed to verify that the proper parameters or return value exist in the stored procedure. If the Stored Procedure transformation is not configured properly, the session fails.

Setting Options for the Stored Procedure

Table 23-2 describes the properties for a Stored Procedure transformation:

Table 23-2. Setting Options for the Stored Procedure Transformation

Setting	Description
Stored Procedure Name	Name of the stored procedure in the database. The Integration Service uses this text to call the stored procedure if the name of the transformation is different than the actual stored procedure name in the database. Leave this field blank if the transformation name matches the stored procedure name. When using the Import Stored Procedure feature, this name matches the stored procedure.
Connection Information	Specifies the database containing the stored procedure. You can select the database or use the \$Source or \$Target variable. By default, the Designer specifies \$Target for Normal stored procedure types. For source pre- and post-load, the Designer specifies \$Source. For target pre- and post-load, the Designer specifies \$Target. You can override these values in the Workflow Manager session properties. If you use one of these variables, the stored procedure must reside in the source or target database you specify when you run the session. If you use \$Source or \$Target, you can specify the database connection for each variable in the session properties. The Integration Service fails the session if it cannot determine the type of database connection. For more information about using \$Source and \$Target, see "Using \$Source and \$Target Variables" on page 502.
Call Text	Text used to call the stored procedure. Only used when the Stored Procedure Type is not Normal. You must include any input parameters passed to the stored procedure within the call text. For more information, see "Calling a Pre- or Post-Session Stored Procedure" on page 509.

Table 23-2. Setting Options for the Stored Procedure Transformation

Setting	Description
Stored Procedure Type	Determines when the Integration Service calls the stored procedure. The options include Normal (during the mapping) or pre- or post-load on the source or target database. Default is Normal.
Execution Order	Order in which the Integration Service calls the stored procedure used in the transformation, relative to any other stored procedures in the same mapping. Only used when the Stored Procedure Type is set to anything except Normal and more than one stored procedure exists.

Using \$Source and \$Target Variables

Use either the \$Source or \$Target variable when you specify the database location for a Stored Procedure transformation. Use these variables in the Connection Information property for a Stored Procedure transformation.

You can also use these variables for Lookup transformations. For more information, see “Lookup Properties” on page 316.

When you configure a session, you can specify a database connection value for \$Source or \$Target. This ensures the Integration Service uses the correct database connection for the variable when it runs the session. You can configure the \$Source Connection Value and \$Target Connection Value properties on the General Options settings of the Properties tab in the session properties.

However, if you do not specify \$Source Connection Value or \$Target Connection Value in the session properties, the Integration Service determines the database connection to use when it runs the session. It uses a source or target database connection for the source or target in the pipeline that contains the Stored Procedure transformation. If it cannot determine which database connection to use, it fails the session.

The following list describes how the Integration Service determines the value of \$Source or \$Target when you do not specify \$Source Connection Value or \$Target Connection Value in the session properties:

- ◆ When you use \$Source and the pipeline contains one relational source, the Integration Service uses the database connection you specify for the source.
- ◆ When you use \$Source and the pipeline contains multiple relational sources joined by a Joiner transformation, the Integration Service uses different database connections, depending on the location of the Stored Procedure transformation in the pipeline:
 - When the Stored Procedure transformation is after the Joiner transformation, the Integration Service uses the database connection for the detail table.
 - When the Stored Procedure transformation is before the Joiner transformation, the Integration Service uses the database connection for the source connected to the Stored Procedure transformation.
- ◆ When you use \$Target and the pipeline contains one relational target, the Integration Service uses the database connection you specify for the target.

- ◆ When you use \$Target and the pipeline contains multiple relational targets, the session fails.
- ◆ When you use \$Source or \$Target in an unconnected Stored Procedure transformation, the session fails.

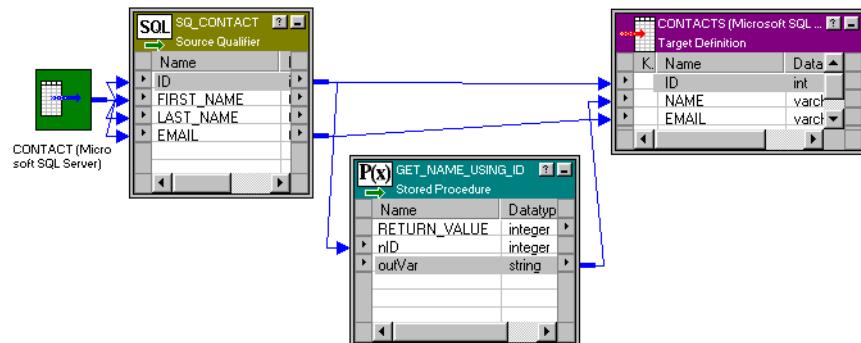
Changing the Stored Procedure

If the number of parameters or the return value in a stored procedure changes, you can either re-import it or edit the Stored Procedure transformation manually. The Designer does not verify the Stored Procedure transformation each time you open the mapping. After you import or create the transformation, the Designer does not validate the stored procedure. The session fails if the stored procedure does not match the transformation.

Configuring a Connected Transformation

Figure 23-1 shows a mapping that sends the ID from the Source Qualifier to an input parameter in the Stored Procedure transformation and retrieves an output parameter from the Stored Procedure transformation that is sent to the target. Every row of data in the Source Qualifier transformation passes data through the Stored Procedure transformation:

Figure 23-1. Sample Mapping with a Stored Procedure Transformation



Although not required, almost all connected Stored Procedure transformations contain input and output parameters. Required input parameters are specified as the input ports of the Stored Procedure transformation. Output parameters appear as output ports in the transformation. A return value is also an output port, and has the R value selected in the transformation Ports configuration. For a normal connected Stored Procedure to appear in the functions list, it requires at least one input and one output port.

Output parameters and return values from the stored procedure are used as any other output port in a transformation. You can map the value of these ports directly to another transformation or target.

To configure a connected Stored Procedure transformation:

1. Create the Stored Procedure transformation in the mapping.

For more information, see “Creating a Stored Procedure Transformation” on page 498.

2. Drag ports from upstream transformations to connect to any available input ports.
3. Drag the output ports of the Stored Procedure to other transformations or targets.
4. Open the Stored Procedure transformation, and select the Properties tab.

Select the appropriate database in the Connection Information if you did not select it when creating the transformation.

Select the Tracing level for the transformation. If you are testing the mapping, select the Verbose Initialization option to provide the most information in the event that the transformation fails. Click OK.

5. Click Repository > Save to save changes to the mapping.

Configuring an Unconnected Transformation

An unconnected Stored Procedure transformation is not directly connected to the flow of data through the mapping. Instead, the stored procedure runs either:

- ◆ **From an expression.** Called from an expression written in the Expression Editor within another transformation in the mapping.
- ◆ **Pre- or post-session.** Runs before or after a session.

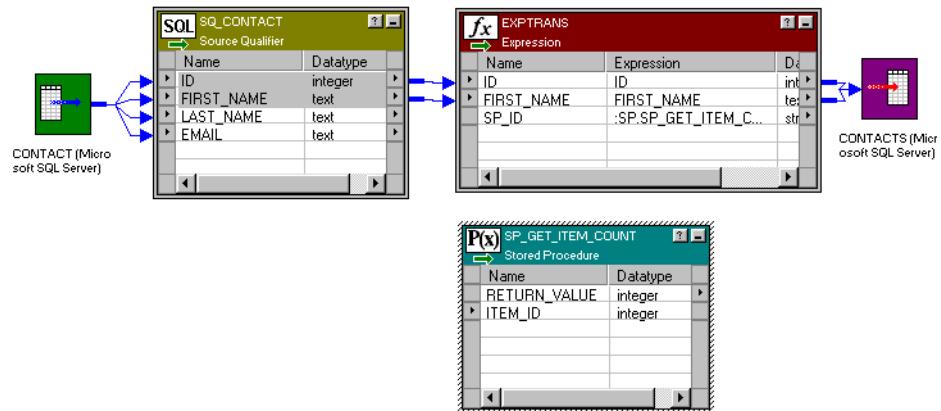
The sections below explain how you can run an unconnected Stored Procedure transformation.

Calling a Stored Procedure From an Expression

In an unconnected mapping, the Stored Procedure transformation does not connect to the pipeline.

Figure 23-2 shows a mapping with an Expression transformation that references the Stored Procedure transformation:

Figure 23-2. Expression Transformation Referencing a Stored Procedure Transformation



However, just like a connected mapping, you can apply the stored procedure to the flow of data through the mapping. In fact, you have greater flexibility since you use an expression to call the stored procedure, which means you can select the data that you pass to the stored procedure as an input parameter.

When using an unconnected Stored Procedure transformation in an expression, you need a method of returning the value of output parameters to a port. Use one of the following methods to capture the output values:

- ◆ Assign the output value to a local variable.
- ◆ Assign the output value to the system variable PROC_RESULT.

By using PROC_RESULT, you assign the value of the return parameter directly to an output port, which can apply directly to a target. You can also combine the two options by assigning one output parameter as PROC_RESULT, and the other parameter as a variable.

Use PROC_RESULT only within an expression. If you do not use PROC_RESULT or a variable, the port containing the expression captures a NULL. You cannot use PROC_RESULT in a connected Lookup transformation or within the Call Text for a Stored Procedure transformation.

If you require nested stored procedures, where the output parameter of one stored procedure passes to another stored procedure, use PROC_RESULT to pass the value.

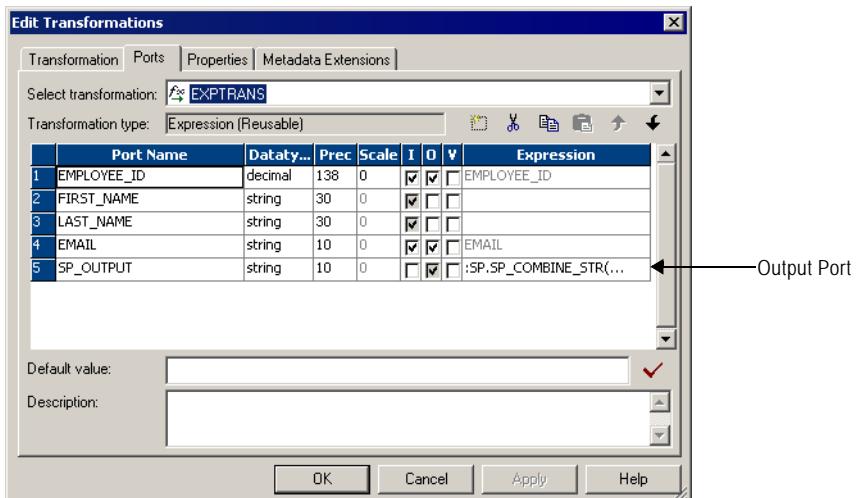
The Integration Service calls the unconnected Stored Procedure transformation from the Expression transformation. Notice that the Stored Procedure transformation has two input ports and one output port. All three ports are string datatypes.

To call a stored procedure from within an expression:

1. Create the Stored Procedure transformation in the mapping.

For more information, see “Creating a Stored Procedure Transformation” on page 498.

2. In any transformation that supports output and variable ports, create a new output port in the transformation that calls the stored procedure. Name the output port.

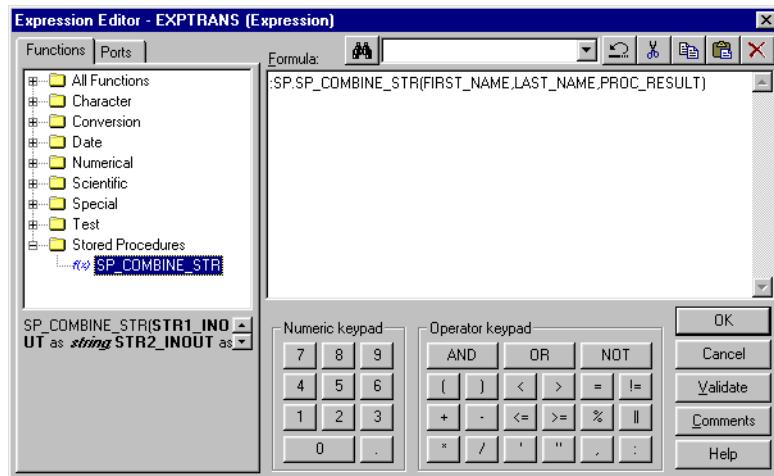


The output port that calls the stored procedure must support expressions. Depending on how the expression is configured, the output port contains the value of the output parameter or the return value.

3. Open the Expression Editor for the port.

The value for the new port is set up in the Expression Editor as a call to the stored procedure using the :SP keyword in the Transformation Language. The easiest way to set this up properly is to select the Stored Procedures node in the Expression Editor, and

click the name of Stored Procedure transformation listed. For a normal connected Stored Procedure to appear in the functions list, it requires at least one input and one output port.



The stored procedure appears in the Expression Editor with a pair of empty parentheses. The necessary input and/or output parameters are displayed in the lower left corner of the Expression Editor.

4. Configure the expression to send input parameters and capture output parameters or return value.

You must know whether the parameters shown in the Expression Editor are input or output parameters. You insert variables or port names between the parentheses in the order that they appear in the stored procedure. The datatypes of the ports and variables must match those of the parameters passed to the stored procedure.

For example, when you click the stored procedure, something similar to the following appears:

```
:SP.GET_NAME_FROM_ID()
```

This particular stored procedure requires an integer value as an input parameter and returns a string value as an output parameter. How the output parameter or return value is captured depends on the number of output parameters and whether the return value needs to be captured.

If the stored procedure returns a single output parameter or a return value (but not both), you should use the reserved variable PROC_RESULT as the output variable. In the previous example, the expression would appear as:

```
:SP.GET_NAME_FROM_ID(inID, PROC_RESULT)
```

inID can be either an input port for the transformation or a variable in the transformation. The value of PROC_RESULT is applied to the output port for the expression.

If the stored procedure returns multiple output parameters, you must create variables for each output parameter. For example, if you create a port called varOUTPUT2 for the stored procedure expression, and a variable called varOUTPUT1, the expression appears as:

```
:SP.GET_NAME_FROM_ID(inID, varOUTPUT1, PROC_RESULT)
```

The value of the second output port is applied to the output port for the expression, and the value of the first output port is applied to varOUTPUT1. The output parameters are returned in the order they are declared in the stored procedure.

With all these expressions, the datatypes for the ports and variables must match the datatypes for the input/output variables and return value.

5. Click Validate to verify the expression, and then click OK to close the Expression Editor.
Validating the expression ensures that the datatypes for parameters in the stored procedure match those entered in the expression.
6. Click OK.
7. Click Repository > Save to save changes to the mapping.

When you save the mapping, the Designer does not validate the stored procedure expression. If the stored procedure expression is not configured properly, the session fails. When testing a mapping using a stored procedure, set the Override Tracing session option to a verbose mode and configure the On Stored Procedure session option to stop running if the stored procedure fails. Configure these session options in the Error Handling settings of the Config Object tab in the session properties. For more information about setting the tracing level, see “Log Files” in the *Workflow Administration Guide*. For more information about the On Stored Procedure Error session property, see “Session Properties Reference” in the *Workflow Administration Guide*.

The stored procedure in the expression entered for a port does not have to affect all values that pass through the port. Using the IIF statement, for example, you can pass only certain values, such as ID numbers that begin with 5, to the stored procedure and skip all other values. You can also set up nested stored procedures so the return value of one stored procedure becomes an input parameter for a second stored procedure.

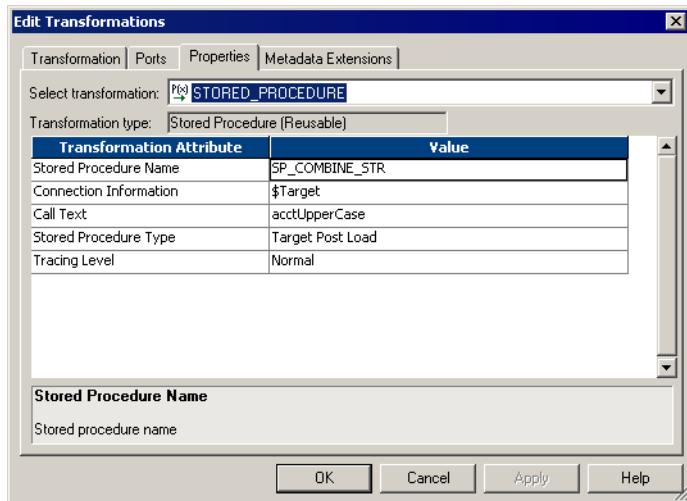
For more information about configuring the stored procedure expression, see “Expression Rules” on page 516.

Calling a Pre- or Post-Session Stored Procedure

You may want to run a stored procedure once per session. For example, if you need to verify that tables exist in a target database before running a mapping, a pre-load target stored procedure can check the tables, and then either continue running the workflow or stop it. You can run a stored procedure on the source, target, or any other connected database.

To create a pre- or post-load stored procedure:

1. Create the Stored Procedure transformation in the mapping.
For more information, see “Creating a Stored Procedure Transformation” on page 498.
2. Double-click the Stored Procedure transformation, and select the Properties tab.



3. Enter the name of the stored procedure.

If you imported the stored procedure, this should be set correctly. If you manually set up the stored procedure, enter the name of the stored procedure.

4. Select the database that contains the stored procedure in Connection Information.
5. Enter the call text of the stored procedure.

This is the name of the stored procedure, followed by any applicable input parameters in parentheses. If there are no input parameters, you must include an empty pair of parentheses, or the call to the stored procedure fails. You do not need to include the SQL statement EXEC, nor do you need to use the :SP keyword. For example, to call a stored procedure called check_disk_space, enter the following:

```
check_disk_space()
```

To pass a string input parameter, enter it without quotes. If the string has spaces in it, enclose the parameter in double quotes. For example, if the stored procedure check_disk_space required a machine name as an input parameter, enter the following:

```
check_disk_space("oracle_db")
```

You must enter values for the parameters, since pre- and post-session procedures cannot pass variables.

When passing a datetime value through a pre- or post-session stored procedure, the value must be in the Informatica default date format and enclosed in double quotes as follows:

```
SP("12/31/2000 11:45:59")
```

6. Select the stored procedure type.

The options for stored procedure type include:

- ◆ **Source Pre-load.** Before the session retrieves data from the source, the stored procedure runs. This is useful for verifying the existence of tables or performing joins of data in a temporary table.
- ◆ **Source Post-load.** After the session retrieves data from the source, the stored procedure runs. This is useful for removing temporary tables.
- ◆ **Target Pre-load.** Before the session sends data to the target, the stored procedure runs. This is useful for verifying target tables or disk space on the target system.
- ◆ **Target Post-load.** After the session sends data to the target, the stored procedure runs. This is useful for re-creating indexes on the database.

7. Select Execution Order, and click the Up or Down arrow to change the order, if necessary.

If you have added several stored procedures that execute at the same point in a session (such as two procedures that both run at Source Post-load), you can set a stored procedure execution plan to determine the order in which the Integration Service calls these stored procedures. You need to repeat this step for each stored procedure you wish to change.

8. Click OK.

9. Click Repository > Save to save changes to the mapping.

Although the repository validates and saves the mapping, the Designer does not validate whether the stored procedure expression runs without an error. If the stored procedure expression is not configured properly, the session fails. When testing a mapping using a stored procedure, set the Override Tracing session option to a verbose mode and configure the On Stored Procedure session option to stop running if the stored procedure fails. Configure these session options on the Error Handling settings of the Config Object tab in the session properties.

You lose output parameters or return values called during pre- or post-session stored procedures, since there is no place to capture the values. If you need to capture values, you might want to configure the stored procedure to save the value in a table in the database.

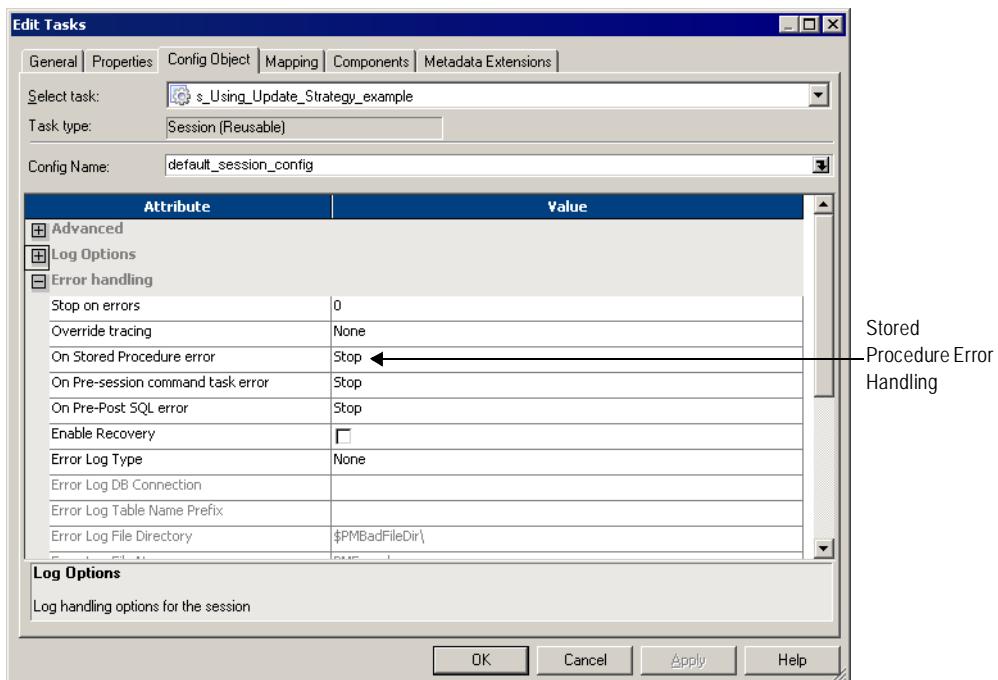
Error Handling

Sometimes a stored procedure returns a database error, such as “divide by zero” or “no more rows”. The final result of a database error during a stored procedure differs depending on when the stored procedure takes place and how the session is configured.

You can configure the session to either stop or continue running the session upon encountering a pre- or post-session stored procedure error. By default, the Integration Service stops a session when a pre- or post-session stored procedure database error occurs.

Figure 23-3 shows the properties you can configure for stored procedures and error handling:

Figure 23-3. Stored Procedure Error Handling



Pre-Session Errors

Pre-read and pre-load stored procedures are considered pre-session stored procedures. Both run before the Integration Service begins reading source data. If a database error occurs during a pre-session stored procedure, the Integration Service performs a different action depending on the session configuration.

- ♦ If you configure the session to stop upon stored procedure error, the Integration Service fails the session.
- ♦ If you configure the session to continue upon stored procedure error, the Integration Service continues with the session.

Post-Session Errors

Post-read and post-load stored procedures are considered post-session stored procedures. Both run after the Integration Service commits all data to the database. If a database errors during a post-session stored procedure, the Integration Service performs a different action depending on the session configuration.

- ◆ If you configure the session to stop upon stored procedure error, the Integration Service fails the session.

However, the Integration Service has already committed all data to session targets.

- ◆ If you configure the session to continue upon stored procedure error, the Integration Service continues with the session.

Session Errors

Connected or unconnected stored procedure errors occurring during the session are not affected by the session error handling option. If the database returns an error for a particular row, the Integration Service skips the row and continues to the next row. As with other row transformation errors, the skipped row appears in the session log.

Supported Databases

The supported options for Oracle, and other databases, such as Informix, Microsoft SQL Server, and Sybase are described below. For more information about database differences, see “Writing a Stored Procedure” on page 495. For more information about supported features, see the database documentation.

SQL Declaration

In the database, the statement that creates a stored procedure appears similar to the following Oracle stored procedure:

```
create or replace procedure sp_combine_str
(str1_inout IN OUT varchar2,
 str2_inout IN OUT varchar2,
 str_out OUT varchar2)
is
begin
str1_inout := UPPER(str1_inout);
str2_inout := upper(str2_inout);
str_out := str1_inout || ' ' || str2_inout;
end;
```

In this case, the Oracle statement begins with CREATE OR REPLACE PROCEDURE. Since Oracle supports both stored procedures and stored functions, only Oracle uses the optional CREATE FUNCTION statement.

Parameter Types

There are three possible parameter types in stored procedures:

- ◆ **IN.** Defines the parameter something that must be passed to the stored procedure.
- ◆ **OUT.** Defines the parameter as a returned value from the stored procedure.
- ◆ **INOUT.** Defines the parameter as both input and output. Only Oracle supports this parameter type.

Input/Output Port in Mapping

Since Oracle supports the INOUT parameter type, a port in a Stored Procedure transformation can act as both an input and output port for the same stored procedure parameter. Other databases should not have both the input and output check boxes selected for a port.

Type of Return Value Supported

Different databases support different types of return value datatypes, and only Informix does not support user-defined return values.

Expression Rules

Unconnected Stored Procedure transformations can be called from an expression in another transformation. Use the following rules and guidelines when configuring the expression:

- ◆ A single output parameter is returned using the variable PROC_RESULT.
- ◆ When you use a stored procedure in an expression, use the :SP reference qualifier. To avoid typing errors, select the Stored Procedure node in the Expression Editor, and double-click the name of the stored procedure.
- ◆ However, the same instance of a Stored Procedure transformation cannot run in both connected and unconnected mode in a mapping. You must create different instances of the transformation.
- ◆ The input/output parameters in the expression must match the input/output ports in the Stored Procedure transformation. If the stored procedure has an input parameter, there must also be an input port in the Stored Procedure transformation.
- ◆ When you write an expression that includes a stored procedure, list the parameters in the same order that they appear in the stored procedure and the Stored Procedure transformation.
- ◆ The parameters in the expression must include all of the parameters in the Stored Procedure transformation. You cannot leave out an input parameter. If necessary, pass a dummy variable to the stored procedure.
- ◆ The arguments in the expression must be the same datatype and precision as those in the Stored Procedure transformation.
- ◆ Use PROC_RESULT to apply the output parameter of a stored procedure expression directly to a target. You cannot use a variable for the output parameter to pass the results directly to a target. Use a local variable to pass the results to an output port within the same transformation.
- ◆ Nested stored procedures allow passing the return value of one stored procedure as the input parameter of another stored procedure. For example, if you have the following two stored procedures:
 - get_employee_id (employee_name)
 - get_employee_salary (employee_id)And the return value for get_employee_id is an employee ID number, the syntax for a nested stored procedure is:

```
:sp.get_employee_salary (:sp.get_employee_id (employee_name))
```

You can have multiple levels of nested stored procedures.
- ◆ Do not use single quotes around string parameters. If the input parameter does not contain spaces, do not use any quotes. If the input parameter contains spaces, use double quotes.

Tips

Do not run unnecessary instances of stored procedures.

Each time a stored procedure runs during a mapping, the session must wait for the stored procedure to complete in the database. You have two possible options to avoid this:

- ◆ **Reduce the row count.** Use an active transformation prior to the Stored Procedure transformation to reduce the number of rows that must be passed the stored procedure. Or, create an expression that tests the values before passing them to the stored procedure to make sure that the value does not really need to be passed.
- ◆ **Create an expression.** Most of the logic used in stored procedures can be easily replicated using expressions in the Designer.

Troubleshooting

I get the error “stored procedure not found” in the session log file.

Make sure the stored procedure is being run in the correct database. By default, the Stored Procedure transformation uses the target database to run the stored procedure. Double-click the transformation in the mapping, select the Properties tab, and check which database is selected in Connection Information.

My output parameter was not returned using a Microsoft SQL Server stored procedure.

Check if the parameter to hold the return value is declared as OUTPUT in the stored procedure. With Microsoft SQL Server, OUTPUT implies input/output. In the mapping, you probably have checked both the I and O boxes for the port. Clear the input port.

The session did not have errors before, but now it fails on the stored procedure.

The most common reason for problems with a Stored Procedure transformation results from changes made to the stored procedure in the database. If the input/output parameters or return value changes in a stored procedure, the Stored Procedure transformation becomes invalid. You must either import the stored procedure again, or manually configure the stored procedure to add, remove, or modify the appropriate ports.

The session has been invalidated since I last edited the mapping. Why?

Any changes you make to the Stored Procedure transformation may invalidate the session. The most common reason is that you have changed the type of stored procedure, such as from a Normal to a Post-load Source type.

Chapter 24

Transaction Control Transformation

This chapter includes the following topics:

- ◆ Overview, 520
- ◆ Transaction Control Transformation Properties, 521
- ◆ Using Transaction Control Transformations in Mappings, 524
- ◆ Mapping Guidelines and Validation, 528
- ◆ Creating a Transaction Control Transformation, 529

Overview

Transformation type:

Active
Connected

PowerCenter lets you control commit and rollback transactions based on a set of rows that pass through a Transaction Control transformation. A transaction is the set of rows bound by commit or rollback rows. You can define a transaction based on a varying number of input rows. You might want to define transactions based on a group of rows ordered on a common key, such as employee ID or order entry date.

In PowerCenter, you define transaction control at two levels:

- ♦ **Within a mapping.** Within a mapping, you use the Transaction Control transformation to define a transaction. You define transactions using an expression in a Transaction Control transformation. Based on the return value of the expression, you can choose to commit, roll back, or continue without any transaction changes.
- ♦ **Within a session.** When you configure a session, you configure it for user-defined commit. You can choose to commit or roll back a transaction if the Integration Service fails to transform or write any row to the target.

When you run the session, the Integration Service evaluates the expression for each row that enters the transformation. When it evaluates a commit row, it commits all rows in the transaction to the target or targets.

When the Integration Service evaluates a rollback row, it rolls back all rows in the transaction from the target or targets.

Note: You can also use the transformation scope in other transformation properties to define transactions. For more information, see “Understanding Commit Points” in the *Workflow Administration Guide*.

Transaction Control Transformation Properties

Use the Transaction Control transformation to define conditions to commit and rollback transactions from relational, XML, and dynamic IBM MQSeries targets. Define these parameters in a transaction control expression on the Properties tab. A transaction is the row or set of rows bound by commit or rollback rows. The number of rows may vary for each transaction.

When you configure a Transaction Control transformation, you define the following components:

- ◆ **Transformation tab.** You can rename the transformation and add a description on the Transformation tab.
- ◆ **Ports tab.** You can add only input/output ports to a Transaction Control transformation.
- ◆ **Properties tab.** You can define the transaction control expression, which flags transactions for commit, rollback, or no action.
- ◆ **Metadata Extensions tab.** You can extend the metadata stored in the repository by associating information with the Transaction Control transformation. For more information, see “Metadata Extensions” in the *Repository Guide*.

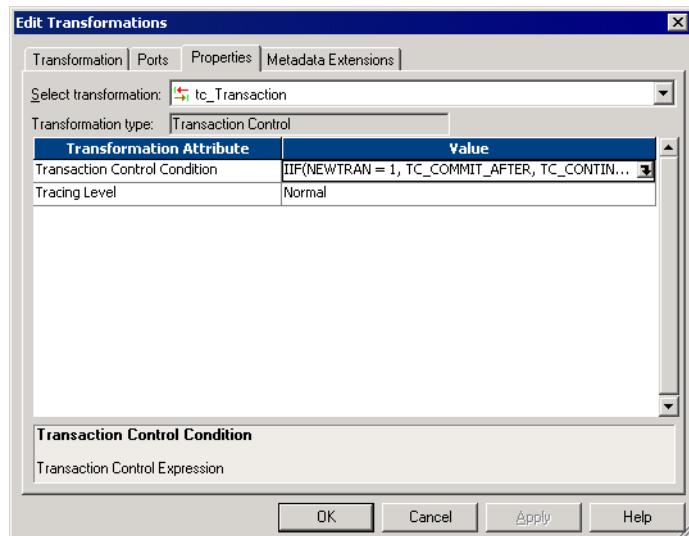
Properties Tab

On the Properties tab, you can configure the following properties:

- ◆ Transaction control expression
- ◆ Tracing level

Figure 24-1 shows the Transaction Control transformation Properties tab:

Figure 24-1. Transaction Control Transformation Properties



Enter the transaction control expression in the Transaction Control Condition field. The transaction control expression uses the IIF function to test each row against the condition. Use the following syntax for the expression:

```
IIF (condition, value1, value2)
```

The expression contains values that represent actions the Integration Service performs based on the return value of the condition. The Integration Service evaluates the condition on a row-by-row basis. The return value determines whether the Integration Service commits, rolls back, or makes no transaction changes to the row. When the Integration Service issues a commit or rollback based on the return value of the expression, it begins a new transaction. Use the following built-in variables in the Expression Editor when you create a transaction control expression:

- ◆ **TC_CONTINUE_TRANSACTION.** The Integration Service does not perform any transaction change for this row. This is the default value of the expression.
- ◆ **TC_COMMIT_BEFORE.** The Integration Service commits the transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.
- ◆ **TC_COMMIT_AFTER.** The Integration Service writes the current row to the target, commits the transaction, and begins a new transaction. The current row is in the committed transaction.
- ◆ **TC_ROLLBACK_BEFORE.** The Integration Service rolls back the current transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.
- ◆ **TC_ROLLBACK_AFTER.** The Integration Service writes the current row to the target, rolls back the transaction, and begins a new transaction. The current row is in the rolled back transaction.

If the transaction control expression evaluates to a value other than commit, rollback, or continue, the Integration Service fails the session.

Example

Suppose you want to use transaction control to write order information based on the order entry date. You want to ensure that all orders entered on any given date are committed to the target in the same transaction. To accomplish this, you can create a mapping with the following transformations:

- ◆ **Sorter transformation.** Sort the source data by order entry date.
- ◆ **Expression transformation.** Use local variables to determine whether the date entered is a new date.

The following table describes the ports in the Expression transformation:

Port Name	Expression	Description
DATE_ENTERED	DATE_ENTERED	Input/Output port. Receives and passes the date entered.
NEW_DATE	IIF(DATE_ENTERED=PREVDATE, 0,1)	Variable port. Tests current value for DATE_ENTERED against the stored value for DATE_ENTERED in the variable port, PREV_DATE.
PREV_DATE	DATE_ENTERED	Variable port. Receives the value for DATE_ENTERED after the Integration Service evaluates the NEW_DATE port.
DATE_OUT	NEW_DATE	Output port. Passes the flag from NEW_DATE to the Transaction Control transformation.

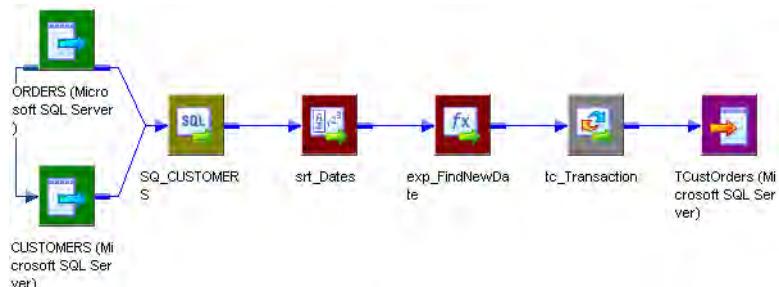
Note: The Integration Service evaluates ports by dependency. The order in which ports display in a transformation must match the order of evaluation: input ports, variable ports, output ports.

- ◆ **Transaction Control transformation.** Create the following transaction control expression to commit data when the Integration Service encounters a new order entry date:

```
IIF(NEW_DATE = 1, TC_COMMIT_BEFORE, TC_CONTINUE_TRANSACTION)
```

Figure 24-2 shows a sample mapping using a Transaction Control transformation:

Figure 24-2. Sample Transaction Control Mapping



Using Transaction Control Transformations in Mappings

Transaction Control transformations are transaction generators. They define and redefine transaction boundaries in a mapping. They drop any incoming transaction boundary from an upstream active source or transaction generator, and they generate new transaction boundaries downstream.

You can also use Custom transformations configured to generate transactions to define transaction boundaries. For more information about configuring a Custom transformation to generate transactions, see “Generate Transaction” on page 68.

Transaction Control transformations can be either effective or ineffective for the downstream transformations and targets in the mapping. The Transaction Control transformation becomes ineffective for downstream transformations or targets if you put a transformation that drops incoming transaction boundaries after it. This includes any of the following active sources or transformations:

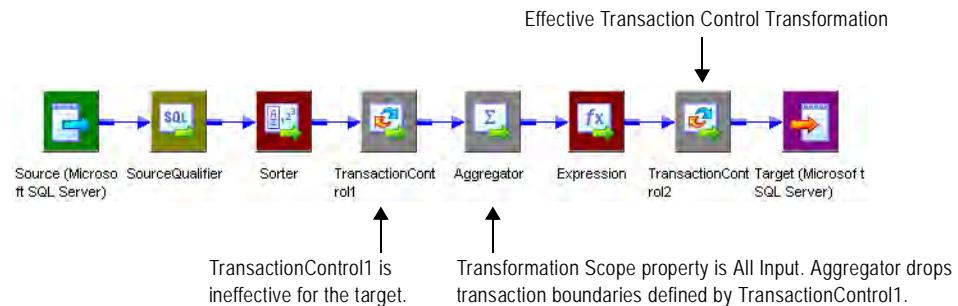
- ◆ Aggregator transformation with the All Input level transformation scope
- ◆ Joiner transformation with the All Input level transformation scope
- ◆ Rank transformation with the All Input level transformation scope
- ◆ Sorter transformation with the All Input level transformation scope
- ◆ Custom transformation with the All Input level transformation scope
- ◆ Custom transformation configured to generate transactions
- ◆ Transaction Control transformation
- ◆ A multiple input group transformation, such as a Custom transformation, connected to multiple upstream transaction control points

For more information about working with transaction control, see “Understanding Commit Points” in the *Workflow Administration Guide*.

Mappings with Transaction Control transformations that are ineffective for targets may be valid or invalid. When you save or validate the mapping, the Designer displays a message indicating which Transaction Control transformations are ineffective for targets.

Figure 24-3 shows a valid mapping with both effective and ineffective Transaction Control transformations:

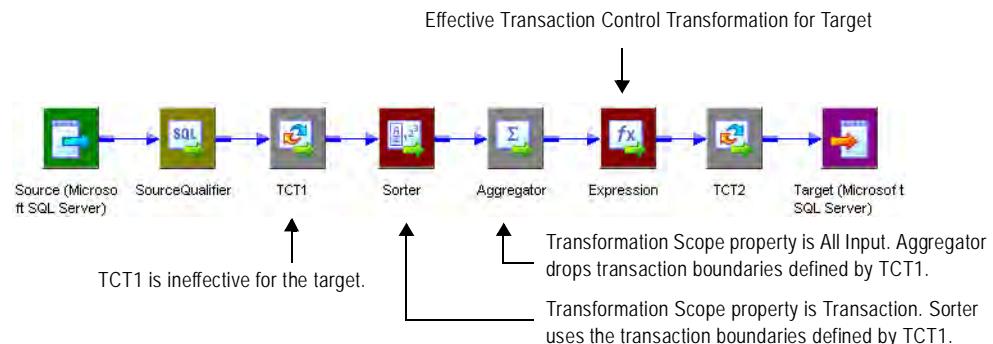
Figure 24-3. Effective and Ineffective Transaction Control Transformations



Although a Transaction Control transformation may be ineffective for a target, it can be effective for downstream transformations. Downstream transformations with the Transaction level transformation scope can use the transaction boundaries defined by an upstream Transaction Control transformation.

Figure 24-4 shows a valid mapping with a Transaction Control transformation that is effective for a Sorter transformation, but ineffective for the target:

Figure 24-4. Transaction Control Transformation Effective for a Transformation



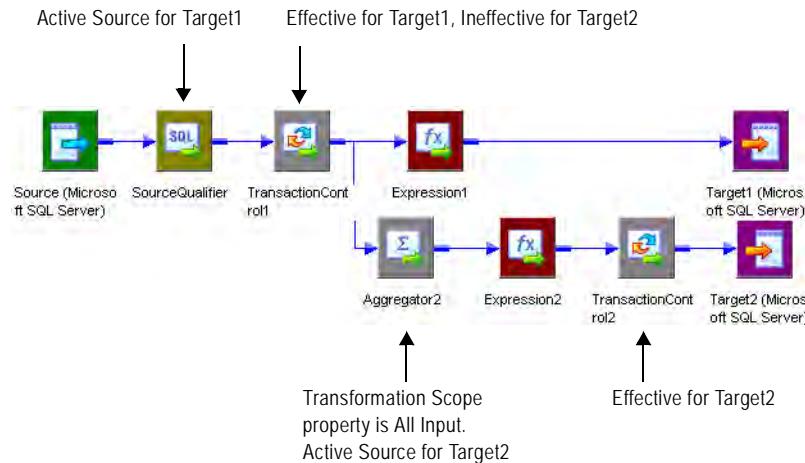
Sample Transaction Control Mappings with Multiple Targets

A Transaction Control transformation may be effective for one target and ineffective for another target.

If each target is connected to an effective Transaction Control transformation, the mapping is valid. If one target in the mapping is not connected to an effective Transaction Control transformation, the mapping is invalid.

Figure 24-5 shows a valid mapping with both an ineffective and an effective Transaction Control transformation:

Figure 24-5. Valid Mapping with Transaction Control Transformations



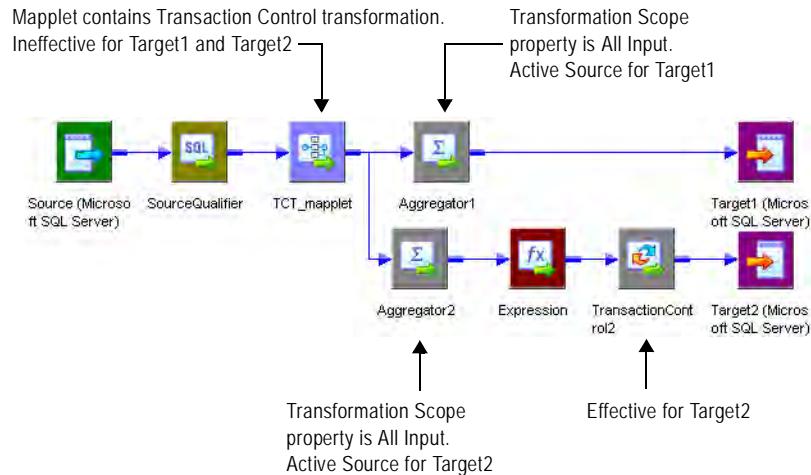
The Integration Service processes TransactionControl1, evaluates the transaction control expression, and creates transaction boundaries. The mapping does not include any transformation that drops transaction boundaries between TransactionControl1 and Target1, making TransactionControl1 effective for Target1. The Integration Service uses the transaction boundaries defined by TransactionControl1 for Target1.

However, the mapping includes a transformation that drops transaction boundaries between TransactionControl1 and Target2, making TransactionControl1 ineffective for Target2. When the Integration Service processes Aggregator2, it drops the transaction boundaries defined by TransactionControl1 and outputs all rows in an open transaction. Then the Integration Service evaluates TransactionControl2, creates transaction boundaries, and uses them for Target2.

If a rollback occurs in TransactionControl1, the Integration Service rolls back only rows from Target1. It does not roll back any rows from Target2.

Figure 24-6 shows an invalid mapping with both an ineffective and an effective Transaction Control transformation:

Figure 24-6. Invalid Mapping with Transaction Control Transformations



The mapping is invalid because Target1 is not connected to an effective Transaction Control transformation.

Mapping Guidelines and Validation

Consider the following rules and guidelines when you create a mapping with a Transaction Control transformation:

- ◆ If the mapping includes an XML target, and you choose to append or create a new document on commit, the input groups must receive data from the same transaction control point.
- ◆ Transaction Control transformations connected to any target other than relational, XML, or dynamic IBM MQSeries targets are ineffective for those targets.
- ◆ You must connect each target instance to a Transaction Control transformation.
- ◆ You can connect multiple targets to a single Transaction Control transformation.
- ◆ You can connect only one effective Transaction Control transformation to a target.
- ◆ You cannot place a Transaction Control transformation in a pipeline branch that starts with a Sequence Generator transformation.
- ◆ If you use a dynamic Lookup transformation and a Transaction Control transformation in the same mapping, a rolled-back transaction might result in unsynchronized target data.
- ◆ A Transaction Control transformation may be effective for one target and ineffective for another target. If each target is connected to an effective Transaction Control transformation, the mapping is valid. See Figure 24-5 on page 526 for an example of a valid mapping with an ineffective Transaction Control transformation.
- ◆ Either all targets or none of the targets in the mapping should be connected to an effective Transaction Control transformation. See Figure 24-6 on page 527 for an example of an invalid mapping where one target has an effective Transaction Control transformation and one target has an ineffective Transaction Control transformation.

Creating a Transaction Control Transformation

To add a Transaction Control transformation to a mapping, complete the following steps.

To create a Transaction Control transformation:

1. In the Mapping Designer, click Transformation > Create. Select the Transaction Control transformation.
2. Enter a name for the transformation.
The naming convention for Transaction Control transformations is *TC_TransformationName*.
3. Enter a description for the transformation. This description appears when you view transformation details in the Repository Manager, making it easier to understand what the transformation does.
4. Click Create. The Designer creates the Transaction Control transformation.
5. Click Done.
6. Drag the ports into the transformation. The Designer creates the input/output ports for each port you include.
7. Open the Edit Transformations dialog box, and select the Ports tab.
You can add ports, edit port names, add port descriptions, and enter default values.
8. Select the Properties tab. Enter the transaction control expression that defines the commit and rollback behavior.
9. Select the Metadata Extensions tab. Create or edit metadata extensions for the Transaction Control transformation. For more information about metadata extensions, see “Metadata Extensions” in the *Repository Guide*.
10. Click OK.
11. Click Repository > Save to save changes to the mapping.

Chapter 25

Union Transformation

This chapter includes the following topics:

- ◆ Overview, 532
- ◆ Working with Groups and Ports, 534
- ◆ Creating a Union Transformation, 536
- ◆ Using a Union Transformation in Mappings, 538

Overview

Transformation type:

Active
Connected

The Union transformation is a multiple input group transformation that you use to merge data from multiple pipelines or pipeline branches into one pipeline branch. It merges data from multiple sources similar to the UNION ALL SQL statement to combine the results from two or more SQL statements. Similar to the UNION ALL statement, the Union transformation does not remove duplicate rows.

The Integration Service processes all input groups in parallel. The Integration Service concurrently reads sources connected to the Union transformation, and pushes blocks of data into the input groups of the transformation. The Union transformation processes the blocks of data based on the order it receives the blocks from the Integration Service.

You can connect heterogeneous sources to a Union transformation. The Union transformation merges sources with matching ports and outputs the data from one output group with the same ports as the input groups.

The Union transformation is developed using the Custom transformation.

Union Transformation Rules and Guidelines

Consider the following rules and guidelines when you work with a Union transformation:

- ♦ You can create multiple input groups, but only one output group.
- ♦ All input groups and the output group must have matching ports. The precision, datatype, and scale must be identical across all groups.
- ♦ The Union transformation does not remove duplicate rows. To remove duplicate rows, you must add another transformation such as a Router or Filter transformation.
- ♦ You cannot use a Sequence Generator or Update Strategy transformation upstream from a Union transformation.
- ♦ The Union transformation does not generate transactions.

Union Transformation Components

When you configure a Union transformation, define the following components:

- ♦ **Transformation tab.** You can rename the transformation and add a description.
- ♦ **Properties tab.** You can specify the tracing level.
- ♦ **Groups tab.** You can create and delete input groups. The Designer displays groups you create on the Ports tab.
- ♦ **Group Ports tab.** You can create and delete ports for the input groups. The Designer displays ports you create on the Ports tab.

You cannot modify the Ports, Initialization Properties, Metadata Extensions, or Port Attribute Definitions tabs in a Union transformation.

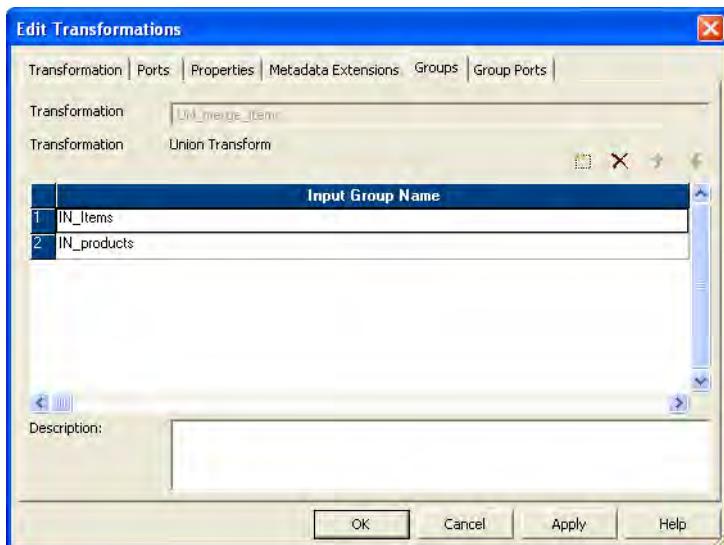
Working with Groups and Ports

A Union transformation has multiple input groups and one output group. Create input groups on the Groups tab, and create ports on the Group Ports tab.

You can create one or more input groups on the Groups tab. The Designer creates one output group by default. You cannot edit or delete the output group.

Figure 25-1 shows the Union transformation Groups tab:

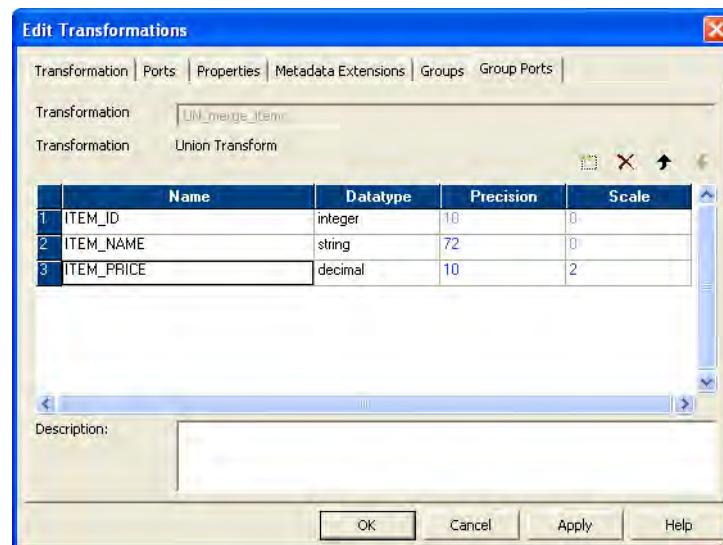
Figure 25-1. Union Transformation Groups Tab



You can create ports by copying ports from a transformation, or you can create ports manually. When you create ports on the Group Ports tab, the Designer creates input ports in each input group and output ports in the output group. The Designer uses the port names you specify on the Group Ports tab for each input and output port, and it appends a number to make each port name in the transformation unique. It also uses the same metadata for each port, such as datatype, precision, and scale.

Figure 25-2 shows the Union transformation Group Ports tab:

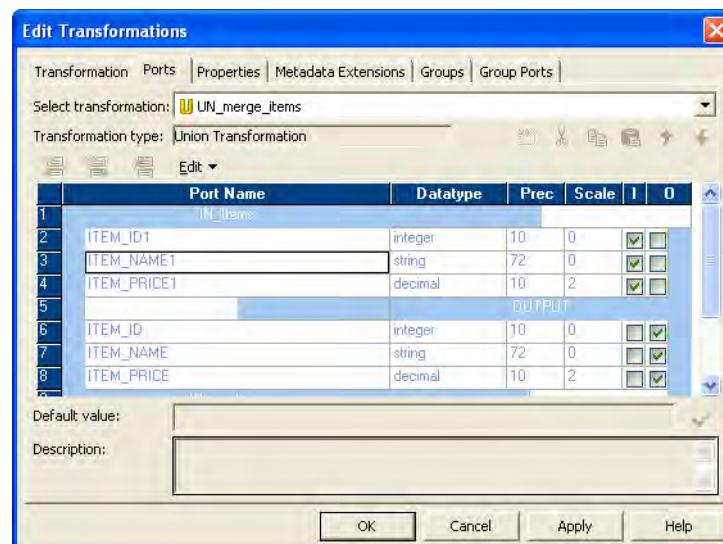
Figure 25-2. Union Transformation Group Ports Tab



The Ports tab displays the groups and ports you create. You cannot edit group and port information on the Ports tab. Use the Groups and Group Ports tab to edit groups and ports.

Figure 25-3 shows the Union transformation Ports tab with the groups and ports defined in Figure 25-1 and Figure 25-2:

Figure 25-3. Union Transformation Ports Tab



Creating a Union Transformation

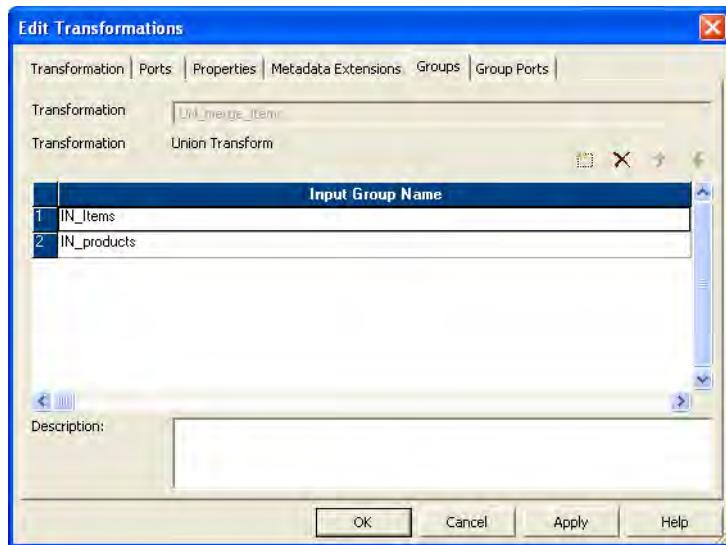
To create a Union transformation, complete the following steps.

To create a Union transformation:

1. In the Mapping Designer, click Transformations > Create.

Select Union Transformation, and enter the name of the transformation. The naming convention for Union transformations is UN_TransformationName. Enter a description for the transformation. Click Create, and then click Done.

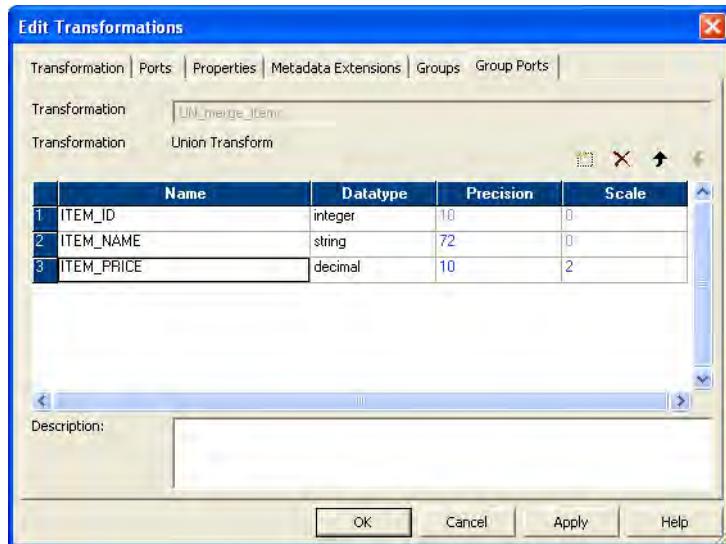
2. Click the Groups tab.



3. Add an input group for each pipeline or pipeline branch you want to merge.

The Designer assigns a default name for each group but they can be renamed.

- 4.** Click the Group Ports tab.



- 5.** Add a new port for each row of data you want to merge.
- 6.** Enter port properties, such as name and datatype.
- 7.** Click the Properties tab to configure the tracing level.
- 8.** Click OK.
- 9.** Click Repository > Save to save changes.

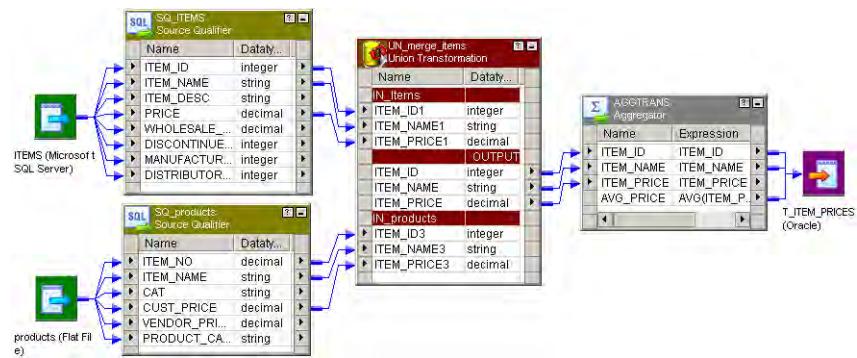
Using a Union Transformation in Mappings

The Union transformation is a non-blocking multiple input group transformation. You can connect the input groups to different branches in a single pipeline or to different source pipelines.

When you add a Union transformation to a mapping, you must verify that you connect the same ports in all input groups. If you connect all ports in one input group, but do not connect a port in another input group, the Integration Service passes NULLs to the unconnected port.

Figure 25-4 shows a mapping with a Union transformation:

Figure 25-4. Mapping with a Union Transformation



When a Union transformation in a mapping receives data from a single transaction generator, the Integration Service propagates transaction boundaries. When the transformation receives data from multiple transaction generators, the Integration Service drops all incoming transaction boundaries and outputs rows in an open transaction. For more information about working with transactions, see “Understanding Commit Points” in the *Workflow Administration Guide*.

Chapter 26

Update Strategy Transformation

This chapter includes the following topics:

- ◆ Overview, 540
- ◆ Flagging Rows Within a Mapping, 541
- ◆ Setting the Update Strategy for a Session, 544
- ◆ Update Strategy Checklist, 547

Overview

Transformation type:

Active
Connected

When you design a data warehouse, you need to decide what type of information to store in targets. As part of the target table design, you need to determine whether to maintain all the historic data or just the most recent changes.

For example, you might have a target table, T_CUSTOMERS, that contains customer data. When a customer address changes, you may want to save the original address in the table instead of updating that portion of the customer row. In this case, you would create a new row containing the updated address, and preserve the original row with the old customer address. This shows how you might store historical information in a target table. However, if you want the T_CUSTOMERS table to be a snapshot of current customer data, you would update the existing customer row and lose the original address.

The model you choose determines how you handle changes to existing rows. In PowerCenter, you set the update strategy at two different levels:

- ♦ **Within a session.** When you configure a session, you can instruct the Integration Service to either treat all rows in the same way (for example, treat all rows as inserts), or use instructions coded into the session mapping to flag rows for different database operations.
- ♦ **Within a mapping.** Within a mapping, you use the Update Strategy transformation to flag rows for insert, delete, update, or reject.

Note: You can also use the Custom transformation to flag rows for insert, delete, update, or reject. For more information about using the Custom transformation to set the update strategy, see “Setting the Update Strategy” on page 66.

Setting the Update Strategy

Use the following steps to define an update strategy:

1. To control how rows are flagged for insert, update, delete, or reject within a mapping, add an Update Strategy transformation to the mapping. Update Strategy transformations are essential if you want to flag rows destined for the same target for different database operations, or if you want to reject rows.
2. Define how to flag rows when you configure a session. You can flag all rows for insert, delete, or update, or you can select the data driven option, where the Integration Service follows instructions coded into Update Strategy transformations within the session mapping.
3. Define insert, update, and delete options for each target when you configure a session. On a target-by-target basis, you can allow or disallow inserts and deletes, and you can choose three different ways to handle updates, as explained in “Setting the Update Strategy for a Session” on page 544.

Flagging Rows Within a Mapping

For the greatest degree of control over the update strategy, you add Update Strategy transformations to a mapping. The most important feature of this transformation is its update strategy expression, used to flag individual rows for insert, delete, update, or reject.

Table 26-1 lists the constants for each database operation and their numeric equivalent:

Table 26-1. Constants for Each Database Operation

Operation	Constant	Numeric Value
Insert	DD_INSERT	0
Update	DD_UPDATE	1
Delete	DD_DELETE	2
Reject	DD_REJECT	3

The Integration Service treats any other value as an insert. For information about these constants and their use, see “Constants” in the *Transformation Language Reference*.

Forwarding Rejected Rows

You can configure the Update Strategy transformation to either pass rejected rows to the next transformation or drop them. By default, the Integration Service forwards rejected rows to the next transformation. The Integration Service flags the rows for reject and writes them to the session reject file. If you do not select Forward Rejected Rows, the Integration Service drops rejected rows and writes them to the session log file.

If you enable row error handling, the Integration Service writes the rejected rows and the dropped rows to the row error logs. It does not generate a reject file. If you want to write the dropped rows to the session log in addition to the row error logs, you can enable verbose data tracing. For more information about error logging, see “Row Error Logging” in the *Workflow Administration Guide*.

Update Strategy Expressions

Frequently, the update strategy expression uses the IIF or DECODE function from the transformation language to test each row to see if it meets a particular condition. If it does, you can then assign each row a numeric code to flag it for a particular database operation. For example, the following IIF statement flags a row for reject if the entry date is after the apply date. Otherwise, it flags the row for update:

```
IIF( ( ENTRY_DATE > APPLY_DATE), DD_REJECT, DD_UPDATE )
```

For more information about the IIF and DECODE functions, see “Functions” in the *Transformation Language Reference*.

To create an Update Strategy transformation:

1. In the Mapping Designer, add an Update Strategy transformation to a mapping.
2. Click Layout > Link Columns.
3. Drag all ports from another transformation representing data you want to pass through the Update Strategy transformation.

In the Update Strategy transformation, the Designer creates a copy of each port you drag. The Designer also connects the new port to the original port. Each port in the Update Strategy transformation is a combination input/output port.

Normally, you would select all of the columns destined for a particular target. After they pass through the Update Strategy transformation, this information is flagged for update, insert, delete, or reject.

4. Open the Update Strategy transformation and rename it.

The naming convention for Update Strategy transformations is *UPD_TransformationName*.

5. Click the Properties tab.

6. Click the button in the Update Strategy Expression field.

The Expression Editor appears.

7. Enter an update strategy expression to flag rows as inserts, deletes, updates, or rejects.

8. Validate the expression and click OK.

9. Click OK to save the changes.

10. Connect the ports in the Update Strategy transformation to another transformation or a target instance.

11. Click Repository > Save.

Aggregator and Update Strategy Transformations

When you connect Aggregator and Update Strategy transformations as part of the same pipeline, you have the following options:

- ◆ **Position the Aggregator before the Update Strategy transformation.** In this case, you perform the aggregate calculation, and then use the Update Strategy transformation to flag rows that contain the results of this calculation for insert, delete, or update.
- ◆ **Position the Aggregator after the Update Strategy transformation.** Here, you flag rows for insert, delete, update, or reject before you perform the aggregate calculation. How you flag a particular row determines how the Aggregator transformation treats any values in that row used in the calculation. For example, if you flag a row for delete and then later use the row to calculate the sum, the Integration Service subtracts the value appearing in this row. If the row had been flagged for insert, the Integration Service would add its value to the sum.

Lookup and Update Strategy Transformations

When you create a mapping with a Lookup transformation that uses a dynamic lookup cache, you must use Update Strategy transformations to flag the rows for the target tables. When you configure a session using Update Strategy transformations and a dynamic lookup cache, you must define certain session properties.

You must define the Treat Source Rows As option as Data Driven. Specify this option on the Properties tab in the session properties.

You must also define the following update strategy target table options:

- ◆ Select Insert
- ◆ Select Update as Update
- ◆ Do not select Delete

These update strategy target table options ensure that the Integration Service updates rows marked for update and inserts rows marked for insert.

If you do not choose Data Driven, the Integration Service flags all rows for the database operation you specify in the Treat Source Rows As option and does not use the Update Strategy transformations in the mapping to flag the rows. The Integration Service does not insert and update the correct rows. If you do not choose Update as Update, the Integration Service does not correctly update the rows flagged for update in the target table. As a result, the lookup cache and target table might become unsynchronized. For more information, see “Setting the Update Strategy for a Session” on page 544.

For more information about using Update Strategy transformations with the Lookup transformation, see “Using Update Strategy Transformations with a Dynamic Cache” on page 354.

For more information about configuring target session properties, see “Working with Targets” in the *Workflow Administration Guide*.

Setting the Update Strategy for a Session

When you configure a session, you have several options for handling specific database operations, including updates.

Specifying an Operation for All Rows

When you configure a session, you can select a single database operation for all rows using the Treat Source Rows As setting.

Configure the Treat Source Rows As session property:

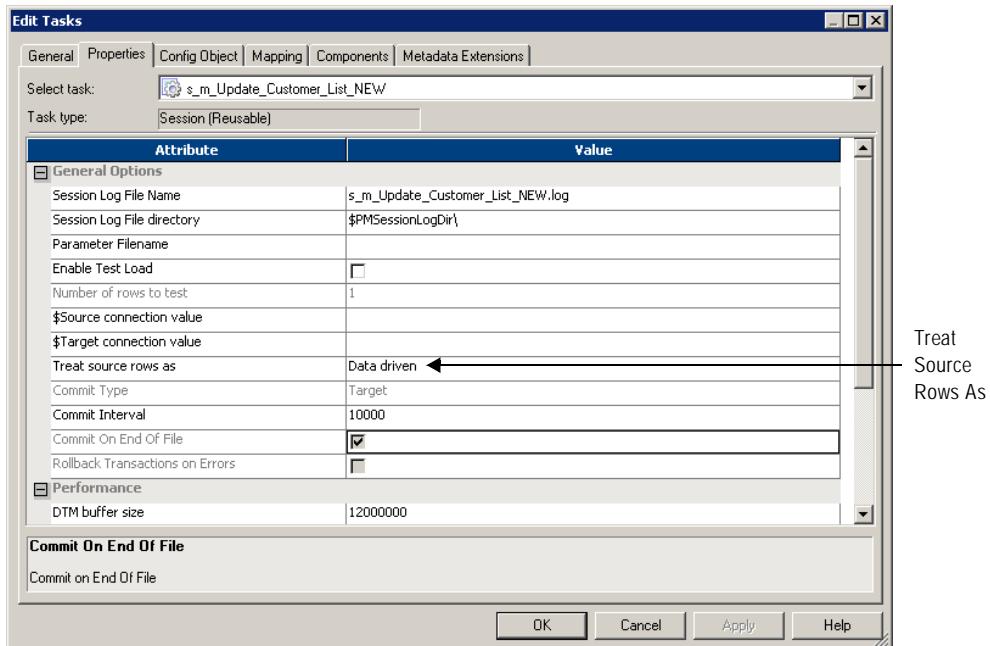


Table 26-2 displays the options for the Treat Source Rows As setting:

Table 26-2. Specifying an Operation for All Rows

Setting	Description
Insert	Treat all rows as inserts. If inserting the row violates a primary or foreign key constraint in the database, the Integration Service rejects the row.
Delete	Treat all rows as deletes. For each row, if the Integration Service finds a corresponding row in the target table (based on the primary key value), the Integration Service deletes it. Note that the primary key constraint must exist in the target definition in the repository.

Table 26-2. Specifying an Operation for All Rows

Setting	Description
Update	Treat all rows as updates. For each row, the Integration Service looks for a matching primary key value in the target table. If it exists, the Integration Service updates the row. The primary key constraint must exist in the target definition.
Data Driven	Integration Service follows instructions coded into Update Strategy and Custom transformations within the session mapping to determine how to flag rows for insert, delete, update, or reject. If the mapping for the session contains an Update Strategy transformation, this field is marked Data Driven by default. If you do not choose Data Driven when a mapping contains an Update Strategy or Custom transformation, the Workflow Manager displays a warning. When you run the session, the Integration Service does not follow instructions in the Update Strategy or Custom transformation in the mapping to determine how to flag rows.

Table 26-3 describes the update strategy for each setting:

Table 26-3. Update Strategy Settings

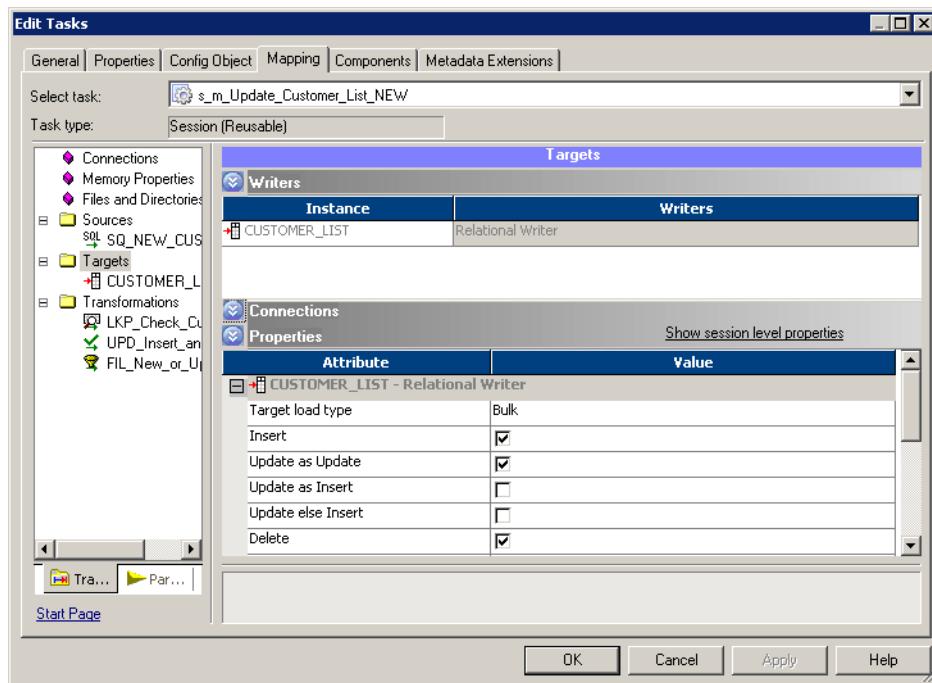
Setting	Use To
Insert	Populate the target tables for the first time, or maintain a historical data warehouse. In the latter case, you must set this strategy for the entire data warehouse, not just a select group of target tables.
Delete	Clear target tables.
Update	Update target tables. You might choose this setting whether the data warehouse contains historical data or a snapshot. Later, when you configure how to update individual target tables, you can determine whether to insert updated rows as new rows or use the updated information to modify existing rows in the target.
Data Driven	Exert finer control over how you flag rows for insert, delete, update, or reject. Choose this setting if rows destined for the same table need to be flagged on occasion for one operation (for example, update), or for a different operation (for example, reject). In addition, this setting provides the only way you can flag rows for reject.

Specifying Operations for Individual Target Tables

Once you determine how to treat all rows in the session, you also need to set update strategy options for individual targets. Define the update strategy options in the Transformations view on Mapping tab of the session properties.

Figure 26-1 displays the update strategy options in the Transformations view on Mapping tab of the session properties:

Figure 26-1. Specifying Operations for Individual Target Tables



You can set the following update strategy options:

- ◆ **Insert.** Select this option to insert a row into a target table.
- ◆ **Delete.** Select this option to delete a row from a table.
- ◆ **Update.** You have the following options in this situation:
 - **Update as Update.** Update each row flagged for update if it exists in the target table.
 - **Update as Insert.** Insert each row flagged for update.
 - **Update else Insert.** Update the row if it exists. Otherwise, insert it.
- ◆ **Truncate table.** Select this option to truncate the target table before loading data.

Update Strategy Checklist

Choosing an update strategy requires setting the right options within a session and possibly adding Update Strategy transformations to a mapping. This section summarizes what you need to implement different versions of an update strategy.

Only perform inserts into a target table.

When you configure the session, select Insert for the Treat Source Rows As session property. Also, make sure that you select the Insert option for all target instances in the session.

Delete all rows in a target table.

When you configure the session, select Delete for the Treat Source Rows As session property. Also, make sure that you select the Delete option for all target instances in the session.

Only perform updates on the contents of a target table.

When you configure the session, select Update for the Treat Source Rows As session property. When you configure the update options for each target table instance, make sure you select the Update option for each target instance.

Perform different database operations with different rows destined for the same target table.

Add an Update Strategy transformation to the mapping. When you write the transformation update strategy expression, use either the DECODE or IIF function to flag rows for different operations (insert, delete, update, or reject). When you configure a session that uses this mapping, select Data Driven for the Treat Source Rows As session property. Make sure that you select the Insert, Delete, or one of the Update options for each target table instance.

Reject data.

Add an Update Strategy transformation to the mapping. When you write the transformation update strategy expression, use DECODE or IIF to specify the criteria for rejecting the row. When you configure a session that uses this mapping, select Data Driven for the Treat Source Rows As session property.

Chapter 27

XML Transformations

This chapter includes the following topics:

- ◆ XML Source Qualifier Transformation, 550
- ◆ XML Parser Transformation, 551
- ◆ XML Generator Transformation, 552

XML Source Qualifier Transformation

Transformation type:

Active
Connected

You can add an XML Source Qualifier transformation to a mapping by dragging an XML source definition to the Mapping Designer workspace or by manually creating one. When you add an XML source definition to a mapping, you need to connect it to an XML Source Qualifier transformation. The XML Source Qualifier transformation defines the data elements that the Integration Service reads when it executes a session. It determines how the PowerCenter reads the source data.

An XML Source Qualifier transformation always has one input or output port for every column in the XML source. When you create an XML Source Qualifier transformation for a source definition, the Designer links each port in the XML source definition to a port in the XML Source Qualifier transformation. You cannot remove or edit any of the links. If you remove an XML source definition from a mapping, the Designer also removes the corresponding XML Source Qualifier transformation. You can link one XML source definition to one XML Source Qualifier transformation.

You can link ports of one XML Source Qualifier group to ports of different transformations to form separate data flows. However, you cannot link ports from more than one group in an XML Source Qualifier transformation to ports in the same target transformation.

You can edit some of the properties and add metadata extensions to an XML Source Qualifier transformation. For more information about using an XML Source Qualifier transformation, see the *XML Guide*.

XML Parser Transformation

Transformation type:

Active
Connected

Use an XML Parser transformation to extract XML inside a pipeline. The XML Parser transformation lets you extract XML data from messaging systems, such as TIBCO or MQ Series, and from other sources, such as files or databases. The XML Parser transformation functionality is similar to the XML source functionality, except it parses the XML in the pipeline. For example, you might want to extract XML data from a TIBCO source and pass the data to relational targets.

The XML Parser transformation reads XML data from a single input port and writes data to one or more output ports.

For more information about the XML Parser transformation, see the *XML Guide*.

XML Generator Transformation

Transformation type:

Active
Connected

Use an XML Generator transformation to create XML inside a pipeline. The XML Generator transformation lets you read data from messaging systems, such as TIBCO and MQ Series, or from other sources, such as files or databases. The XML Generator transformation functionality is similar to the XML target functionality, except it generates the XML in the pipeline. For example, you might want to extract data from relational sources and pass XML data to targets.

The XML Generator transformation accepts data from multiple ports and writes XML through a single output port.

For more information about the XML Generator transformation, see the *XML Guide*.

Index

A

ABORT function

See also Transformation Language Reference

using 22

active transformations

See also transformations

Aggregator 38

Custom 54

Filter 190

Java 215

Joiner 284

Midstream XML Generator 552

Midstream XML Parser 551

Normalizer 372

overview 2

Rank 382

Router 390

Sorter 416

Source Qualifier 426, 460

Transaction Control 521

Union 532

Update Strategy 540

XML Source Qualifier 550

add statistic output port

SQL transformation option 485

adding

comments to expressions 12

groups 394

advanced interface

EDataType class 274

example 277

invoking Java expressions 273

Java expressions 273

JExpression API Reference 279

JExpression class 276

JExprParaMetadata class 274

advanced options

SQL transformation 472

aggregate functions

See also Transformation Language Reference

list of 40

null values 41

overview 40

using in expressions 40

Aggregator transformation

compared to Expression transformation 38

components 38

conditional clause example 41

creating 47

functions list 42

group by ports 42

nested aggregation 40

non-aggregate function example 41

null values 41

optimizing performance 50

overview 38

ports 38

sorted ports 45
STDDEV (standard deviation) function 40
tracing levels 48
troubleshooting 51
Update Strategy combination 542
using variables 14
using with the Joiner transformation 294
VARIANCE function 40
All Input transformation scope
 behavior in Joiner transformation 300
API functions
 Custom transformation 104
API methods
 Java transformation 237
array-based functions
 data handling 132
 is row valid 132
 maximum number of rows 130
 number of rows 131
 overview 130
 row strategy 135
 set input error row 136
ASCII
 Custom transformation 55
 External Procedure transformation 146
ASCII mode
 configuring sort order for Joiner transformation 292
associated ports
 Lookup transformation 348
 sequence ID 349
averages
 See Aggregator transformation

B

BankSoft example
 Informatica external procedure 159
 overview 148
BigDecimal datatype
 Java transformation 215, 230
binary datatypes
 Java transformation 215
blocking
 detail rows in Joiner transformation 299
blocking data
 Custom transformation 70
 Custom transformation functions 125
 Joiner transformation 299

C

C/C++
 See also Visual C++
 linking to Integration Service 173
Cache directory
 Joiner transformation property 286
cache file name prefix
 overview 363
caches
 concurrent 340, 341
 dynamic lookup cache 345
 Joiner transformation 299
 Lookup transformation 338
 named persistent lookup 363
 sequential 340
 sharing lookup 363
 static lookup cache 344
caching
 master rows in Joiner transformation 299
calculations
 aggregate 38
 multiple calculations 142
 using the Expression transformation 142
 using variables with 15
case-sensitive string comparison
 Joiner transformation property 286
Char datatypes
 Java transformation 215
Class Name property
 Java transformation 221
CLASSPATH
 Java transformation, configuring 229
COBOL source definitions
 adding Normalizer transformation automatically 374
 normalizing 372
code pages
 See also Administrator Guide
 access functions 185
 Custom transformation 55
 External Procedure transformation 146
code snippets
 creating for Java transformation 225
COM external procedures
 adding to repository 153
 compared to Informatica external procedures 148
 creating 149
 creating a source 155
 creating a target 155
 datatypes 171
 debugging 173

developing in Visual Basic 156
developing in Visual C++ 149, 154
development notes 171
distributing 169
exception handling 172
initializing 175
memory management 173
overview 149
registering with repositories 153
return values 172
row-level procedures 172
server type 149
unconnected 175

COM servers
 type for COM external procedures 149

comments
 adding to expressions 12

commit
 example for Java transformation 239
 Java transformation API method 239
 syntax for Java transformation 239

compilation errors
 identifying in Java transformation 234

compiling
 Custom transformation procedures 85
 DLLs on Windows systems 165
 Java transformation 231

composite key
 creating with Sequence Generator transformation 403

concurrent caches
 See caches

conditions
 Filter transformation 192
 Joiner transformation 288
 Lookup transformation 328, 332
 Router transformation 393

configuring
 ports 6, 7

connect string
 syntax 471

connected lookups
 See also Lookup transformation
 creating 335
 description 309
 overview 309

connected transformations
 Aggregator 38
 Custom 54
 Expression 142
 External Procedure 146
 Filter 190

Joiner 284
Lookup 308
Midstream XML Generator 552
Midstream XML Parser 551
Normalizer 372
Rank 382
Router 390
Sequence Generator 402
Source Qualifier 426
SQL 460
Stored Procedure 490
Update Strategy 540
XML Source Qualifier 550

connecting to databases
 SQL transformation 470

connection settings
 SQL transformation 470

connection variables
 connection property 502

connections
 SQL transformation 470

connectivity
 connect string examples 471

constants
 replacing null values with 21

continue on SQL error
 SQL transformation 485

creating
 Aggregator transformation 47
 COM external procedures 149
 connected Lookup transformation 335
 Custom transformation 57, 73
 Expression transformation 143
 Filter transformation 193
 Informatica external procedures 159
 Joiner transformation 303
 non-reusable instance of reusable transformation 33
 ports 7
 Rank transformation 386
 reusable transformations 32
 Router transformation 400
 Sequence Generator transformation 412
 Stored Procedure transformation 498
 transformations 5
 Union transformation 536
 Update Strategy transformation 541

Current Value property
 Sequence Generator transformation 407, 409

CURRVAL port
 Sequence Generator transformation 406

custom functions

using with Java expressions 264
Custom transformation
 blocking data 70
 building the module 85
 code pages 55
 compiling procedures 85
 components 58
 creating 57, 73
 creating groups 59
 creating ports 59
 creating procedures 73
 defining port relationships 60
 distributing 56
 functions 90
 Generate Transaction property 68
 generating code files 57, 75
 initialization properties 72
 Inputs May Block property 70
 Is Partitionable property 65
 metadata extensions 72
 overview 54
 passing rows to procedure 96
 port attributes 62
 procedure properties 72
 properties 64
 property IDs 109
 Requires Single Thread property 66
 rules and guidelines 57
 setting the update strategy 66
 threads 66
 thread-specific code 66
 transaction boundaries 69
 transaction control 68
 Transformation Scope property 68
 Update Strategy property 66
Custom transformation functions
 API 104
 array-based 130
 blocking logic 125
 change default row strategy 129
 change string mode 126
 data boundary output 121
 data handling (array-based) 132
 data handling (row-based) 117
 deinitialization 102
 error 122
 generated 98
 increment error count 124
 initialization 98
 is row valid 132
 is terminated 124
 maximum number of rows 130
 navigation 105
 notification 100
 number of rows 131
 output notification 121
 pointer 126
 property 108
 rebind datatype 115
 row strategy (array-based) 135
 row strategy (row-based) 128
 session log 123
 set data access mode 104
 set data code page 127
 set input error row 136
 set pass-through port 120
 working with handles 90, 105
Custom transformation procedures
 creating 73
 example 76
 generating code files 75
 thread-specific 66
 working with rows 96
Cycle
 Sequence Generator transformation property 407, 408

D

data
 joining 284
 pre-sorting 45
 rejecting through Update Strategy transformation 547
 selecting distinct 454
 storing temporary 14
data driven
 overview 545
data handling functions
 array-based 132
 row-based 117
database resilience
 SQL transformation 478
databases
See also Installation and Configuration Guide
See also specific database vendors, such as Oracle
 joining data from different 284
 options supported 514
datatypes
 COM 171
 Java transformation 215
 Source Qualifier 426
 transformation 171
Date/Time datatypes

Java transformation 215
DB2
See IBM DB2
 debugging
 external procedures 173
 Java transformation 232
 default groups
 Router transformation 392
 default join
 Source Qualifier 434
 default query
 methods for overriding 432
 overriding using Source Qualifier 438
 overview 431
 viewing 431
 default values
 Aggregator group by ports 43
 entering 28
 Filter conditions 193
 input ports 18
 input/output ports 18
 output ports 18, 20
 overview 18
 rules for 28
 validating 28
defineJExpression
 Java expression API method 275
defining
 port dependencies in Custom transformation 60
deinitialization functions
 Custom transformation 102
dependencies
 ports in Custom transformations 60
detail outer join
 description 290
detail rows
 blocking in Joiner transformation 299
 processing in sorted Joiner transformation 299
 processing in unsorted Joiner transformation 299
developing
 COM external procedures 149
 Informatica external procedures 159
dispatch function
 description 181
distributing
 Custom transformation procedures 56
 external procedures 169
DLLs (dynamic linked libraries)
 compiling external procedures 165
double datatype
 Java transformation 215, 230
dynamic connections
 performance considerations 473
 SQL transformation 470
dynamic linked libraries
See DLLs
dynamic lookup cache
 error threshold 359
 filtering rows 347
 overview 345
 reject loading 359
 synchronizing with target 359
 using flat file sources 345
dynamic Lookup transformation
 output ports 347
dynamic SQL queries
 SQL transformation 466

E

EDataType class
 Java expressions 274
effective Transaction Control transformation
 definition 524
End Value property
 Sequence Generator transformation 407, 409
entering
 expressions 10, 12
 source filters 450
 SQL query override 438
 user-defined joins 440
environment variables
 setting for Java packages 229
error count
 incrementing for Java transformation 243
ERROR function
See also Transformation Language Reference
 using 22
error handling
 for stored procedures 512
 with dynamic lookup cache 359
error messages
See also Troubleshooting Guide
 for external procedures 173
 tracing for external procedures 173
errors
 COBOL sources 379
 handling 25
 increasing threshold in Java transformation 243
 validating in Expression Editor 13
 with dynamic lookup cache 359
exceptions

- from external procedures 172
- Expression Editor**
 - overview 12
 - syntax colors 13
 - using with Java expressions 267
 - validating expressions using 13
- Expression transformation**
 - creating 143
 - multiple calculations 142
 - overview 142
 - routing data 143
 - using variables 14
- expressions**

See also Transformation Language Reference

 - Aggregator transformation 40
 - calling lookups 333
 - calling stored procedure from 506
 - entering 10, 12
 - Filter condition 192
 - non-aggregate 43
 - return values 11
 - rules for Stored Procedure transformation 516
 - simplifying 14
 - update strategy 541
 - using with Java transformation 264
 - validating 13
- External Procedure transformation**

See also COM external procedures

See also Informatica external procedures

 - ATL objects 150
 - BankSoft example 148
 - building libraries for C++ external procedures 152
 - building libraries for Informatica external procedures 165
 - building libraries for Visual Basic external procedures 158
 - code page access function 185
 - COM datatypes 171
 - COM external procedures 149
 - COM vs. Informatica types 148
 - creating in Designer 159
 - debugging 173
 - description 147
 - development notes 171
 - dispatch function 181
 - exception handling 172
 - external procedure function 181
 - files needed 179
 - IDispatch interface 149
 - Informatica external procedure using BankSoft example 159
 - Informatica external procedures 159
 - initializing 175
 - interface functions 181
 - member variables 185
 - memory management 173
 - MFC AppWizard 165
 - multi-threaded code 146
 - overview 146
 - parameter access functions 183
 - partition related functions 186
 - pipeline partitioning 147
 - process variable support 180
 - properties 147
 - property access function 182
 - return values 172
 - row-level procedure 172
 - session 156
 - 64-bit 184
 - tracing level function 187
 - unconnected 175
 - using in a mapping 155
 - Visual Basic 156
 - Visual C++ 149
 - wrapper classes 173
- external procedures**

See also External Procedure transformation

 - debugging 173
 - development notes 171
 - distributing 169
 - distributing Informatica external procedures 170
 - interface functions 181
 - linking to 146

F

- failing sessions**
 - Java transformation 240
- failSession**
 - example for Java transformation 240
 - Java transformation API method 240
 - syntax for Java transformation 240
- Filter transformation**
 - condition 192
 - creating 193
 - example 190
 - overview 190
 - performance tips 195
 - tips for developing 195
 - troubleshooting 196
- filtering rows**
 - Source Qualifier as filter 195

transformation for 190, 416
flat file lookups
 description 311
 sorted input 311
flat files
 joining data 284
 lookups 311
foreign key
 creating with Sequence Generator transformation 403
Forwarding Rejected Rows
 configuring 541
 option 541
full database connection
 passing to SQL transformation 470
full outer join
 definition 291
functions
 See also Transformation Language Reference
 aggregate 40
 non-aggregate 41

G

generate Java code
 Java expressions 267
generate output row
 Java transformation 241
generate rollback row
 Java transformation 247
generate transaction
 Java transformation 223
Generate Transaction property
 Java transformation 223
generated functions
 Custom transformation 98
generateRow
 example for Java transformation 241
 Java transformation API methods 241
 syntax for Java transformation 241
generating transactions
 Custom transformation 68
 Java transformation 224, 239
getBytes
 Java expression API method 282
getDouble
 Java expression API method 281
getInRowType
 example for Java transformation 242
 Java transformation API method 242
 syntax for Java transformation 242
getInt

 Java expression API method 281
getLong
 Java expression API method 281
getResultSetMetadata
 Java expression API method 280
getStringBuffer
 Java expression API method 282
group by ports
 Aggregator transformation 42
 non-aggregate expression 43
 using default values 43
group filter condition
 Router transformation 393
groups
 adding 394
 Custom transformation 59
 Custom transformation rules 60
 HTTP transformation 205
 Java transformation 219
 Router transformation 392
 Union transformation 534
 user-defined 392

H

handles
 Custom transformation 90
Helper Code tab
 example 257
 Java transformation 226
heterogeneous joins
 See Joiner transformation
high precision
 enabling for Java transformation 230
HTTP transformation
 authentication 199
 configuring groups and ports 205
 configuring HTTP tab 204
 configuring properties 204
 creating 200
 examples 209
 groups 205
 Is Partitionable property 203
 Requires Single Thread per Partition property 203
 thread-specific code 203

I

IBM DB2
 connect string syntax 471

IDispatch interface
 defining a class 149

IIF function
 replacing missing keys with Sequence Generator transformation 403

Import Packages tab
 example 256
 Java transformation 226

Increment by
 Sequence Generator transformation property 407, 408

incrementErrorCount
 example for Java transformation 243
 Java transformation API method 243
 syntax for Java transformation 243

incrementing
 setting sequence interval 408

indexes
 lookup conditions 336
 lookup table 313, 336

ineffective Transaction Control transformation
 definition 524

Informatica external procedures
 compared to COM 148
 debugging 173
 developing 159
 development notes 171
 distributing 170
 exception handling 172
 generating C++ code 162
 initializing 175
 memory management 173
 return values 172
 row-level procedures 172
 unconnected 175

Informix
See also Installation and Configuration Guide
 connect string syntax 471
 stored procedure notes 495

initialization functions
 Custom transformation 98

initializing
 Custom transformation procedures 72
 external procedures 175
 Integration Service variable support for 180
 variables 16

input parameters
 stored procedures 491

input ports
 default values 18
 overview 8
 using as variables 227

input row
 getting row type 242

input/output ports
 default values 18
 overview 8

Inputs Must Block property
 Java transformation 222

instance variable
 Java transformation 226, 227, 228

instances
 definition 31

Integration Service
 aggregating data 42
 datatypes 171
 error handling of stored procedures 512
 running in debug mode 173
 transaction boundaries 69
 variable support 180

invoke
 Java expression API method 279

invokeJExpression
 API method 271

Is Active property
 Java transformation 222

Is Partitionable property
 Custom transformation 65
 HTTP transformation 203
 Java transformation 222

isNull
 example for Java transformation 244
 Java transformation API method 244
 syntax for Java transformation 244

isResultNull
 Java expression API method 280

J

Java Classpath
 session property 229

Java code snippets
 creating for Java transformation 225
 example 256

Java Code tab
 using 217

Java expression API methods
 defineJExpression 275
 getBytes 282
 getDouble 281
 getInt 281
 getLong 281
 getResultMetadata 280

getStringBuffer 282
invoke 279
isResultNull 280

Java expressions
advanced interface 273
advanced interface example 277
configuring 266
configuring functions 266
creating 267
EDataType class 274
expression function types 264
generate Java code 267
generating 266
invokeJExpression API method 271
invoking with advanced interface 273
invoking with simple interface 271
Java Expressions tab 268
JExpression API reference 279
JExpression class 276
JExprParaMetadata class 274
rules and guidelines 271, 273
simple interface 271
simple interface example 272
steps to create 268
using custom functions 264
using transformation language functions 264
using user-defined functions 264
 using with Java transformation 264

Java Expressions tab
 Java expressions 268

Java packages
 importing 226

Java transformation
 active 215
 API methods 237
 checking null values 244
 Class Name property 221
 compilation errors 232
 compiling 231
 creating code 225
 creating groups 219
 creating Java code 225
 creating ports 219
 datatype mapping 215
 debugging 232
 default port values 220
 example 252
 failing session 240
 Generate Transaction property 223, 224
 getting input row type 242
 Helper Code tab 226

Import Package tab 226
Is Partitionable property 222
Java Code tab 217
Language property 221
locating errors 232
On End of Data tab 228
On Input Row tab 227
On Receiving Transaction tab 228
Output is Deterministic property 223
Output is Ordered property 223
passive 215
primitive datatypes 215
properties 221
Requires Single Thread Per Partition property 223
session level classpath 229
session log 245, 246
setting CLASSPATH 229
setting null values 248
setting output row type 249
setting the update strategy 224
Tracing Level property 222
transaction control 223
transformation level classpath 229
Transformation Scope property 222, 223
Update Strategy property 224

Java transformation API methods
 commit 239
 failSession 240
 generateRow 241
 getInRowType 242
 incrementErrorCount 243
 isNull 244
 logError 246
 logInfo 245
 rollback 247
 setNull 248
 setOutRowType 249

JDK
 Java transformation 214

JExpression API reference
 Java expressions 279

JExpression class
 Java expressions 276

JExprParaMetadata class
 Java expressions 274

join condition
 defining 294
 overview 288
 using sort origin ports 292

join override
 left outer join syntax 445

normal join syntax 443
right outer join syntax 447
join syntax
 left outer join 445
 normal join 443
 right outer join 447
join type
 detail outer join 290
 full outer join 291
Joiner properties 289
 left outer join 442
 master outer join 290
 normal join 289
 right outer join 442
 Source Qualifier transformation 442
Join Type property
 Joiner transformation 286
joiner cache
 Joiner transformation 299
Joiner data cache size
 Joiner transformation property 287
Joiner index cache size
 Joiner transformation property 287
Joiner transformation
 All Input transformation scope 300
 behavior with All Input transformation scope 300
 behavior with Row transformation scope 300
 behavior with Transaction transformation scope 300
 blocking source data 299
 caches 299
 conditions 288
 configuring join condition to use sort origin ports 292
 configuring sort order 292
 configuring sort order in ASCII mode 292
 configuring sort order in Unicode mode 292
 creating 303
 detail pipeline 284
 dropping transaction boundaries 302
 join types 289
 joining data from the same source 296
 joining multiple databases 284
 master pipeline 284
 overview 284
 performance tips 306
 preserving transaction boundaries 301
 processing real-time data 301
 properties 286
 real-time data 300
Row transformation scope 300
rules for input 284
Transaction transformation scope 300
transactions 300
transformation scope 300
using with Sorter transformation 292
joining sorted data
 configuring to optimize join performance 292
 using sorted flat files 292
 using sorted relational data 292
 using Sorter transformation 292
joins
 creating key relationships for 436
 custom 435
 default for Source Qualifier 434
 Informatica syntax 442
 user-defined 440
JRE
 Java transformation 214

K

keys
 creating for joins 436
 creating with Sequence Generator transformation 403
 creating with sequence IDs 349
 source definitions 436

L

Language property
 Java transformation 221
left outer join
 creating 445
 syntax 445
libraries
 for C++ external procedures 152
 for Informatica external procedures 165
 for VB external procedures 158
load order
 Source Qualifier 426
load types
 stored procedures 511
local variables
 See variables
log files
 See session logs
logError
 example for Java transformation 246
 Java transformation API method 246
 syntax for Java transformation 246
logical database connection
 passing to SQL transformation 470

performance considerations 473
logInfo
 example for Java transformation 245
 Java transformation API method 245
 syntax for Java transformation 245
long datatype
 Java transformation 215
lookup caches
 definition 338
 dynamic 345
 dynamic, error threshold 359
 dynamic, synchronizing with target 359
 dynamic, WHERE clause 358
 handling first and last values 329
 named persistent caches 363
 overriding ORDER BY 325
 overview 338
 partitioning guidelines with unnamed caches 363
 persistent 342
 recache from database 340
 reject loading 359
 sharing 363
 sharing unnamed lookups 363
 static 344
lookup condition
 definition 315
 overview 328
lookup ports
 definition 313
 NewLookupRow 347
 overview 313
lookup properties
 configuring 320
lookup query
 description
 dynamic cache 358
 ORDER BY 324
 overriding 324
 overview 324
 reserved words 326
 Sybase ORDER BY limitation 326
 WHERE clause 358
Lookup SQL Override option
 dynamic caches, using with 358
 mapping parameters and variables 324
 reducing cache size 325
lookup table
 indexes 313, 336
Lookup transformation
See also Workflow Administration Guide
 associated input port 348
 cache sharing 363
 caches 338
 components of 313
 condition 328, 332
 connected 309
 creating connected lookup 335
 default query 324
 dynamic cache 345
 entering custom queries 327
 error threshold 359
 expressions 333
 filtering rows 347
 flat file lookups 308, 311
 lookup sources 308
 mapping parameters and variables 324
 multiple matches 329
 named persistent cache 363
 NewLookupRow port 347
 overriding the default query 324
 overview 308
 performance tips 336
 persistent cache 342
 ports 313
 properties 316
 recache from database 340
 reject loading 359
 return values 332
 sequence ID 349
 synchronizing dynamic cache with target 359
 unconnected 309, 331
 Update Strategy combination 543
lookups
See lookup query 324

M

Mapping Designer
 adding reusable transformation 33
 creating ports 7
mapping parameters
 in lookup SQL override 324
 in Source Qualifier transformations 427
mapping variables
 in lookup SQL override 324
 in Source Qualifier transformations 427
 reusable transformations 31
mappings
 adding COBOL sources 373
 adding reusable transformations 33
 adding transformations 5
 affected by stored procedures 491

configuring connected Stored Procedure
 transformation 504
configuring unconnected Stored Procedure
 transformation 506
flagging rows for update 541
lookup components 313
modifying reusable transformations 34
using an External Procedure transformation 155
 using Router transformations 398
master outer join
 description 290
 preserving transaction boundaries 301
master rows
 caching 299
 processing in sorted Joiner transformation 299
 processing in unsorted Joiner transformation 299
max output row count
 SQL transformation 485
memory management
 for external procedures 173
metadata extensions
 in Custom transformations 72
methods
 Java transformation 227, 228
 Java transformation API 237
MFC AppWizard
 overview 165
Microsoft SQL Server
 connect string syntax 471
 stored procedure notes 496
Midstream XML Generator
 overview 552
Midstream XML Parser
 overview 551
missing values
 replacing with Sequence Generator 403
multi-group
 transformations 9
multiple matches
 Lookup transformation 329

N

named cache
 persistent 342
 recache from database 342
 sharing 365
named persistent lookup cache
 overview 363
 sharing 365
native datatypes

O

object datatype
 Java transformation 215
On End of Data tab
 Java transformation 228
On Input Row tab

SQL transformation 465
NewLookupRow output port
 overview 347
NEXTVAL port
 Sequence Generator 404
non-aggregate expressions
 overview 43
non-aggregate functions
 example 41
normal join
 creating 443
 definition 289
 preserving transaction boundaries 301
 syntax 443
Normal tracing levels
 overview 30
normalization
 definition 372
Normalizer transformation
 adding 376
 COBOL source automatic configuration 374
 differences (VSAM v. relational) 378
 overview 372
 troubleshooting 379
notification functions
 Custom transformation 100
Null ordering in detail
 Joiner transformation property 287
Null ordering in master
 Joiner transformation property 286
null values
 aggregate functions 41
 checking in Java transformation 244
 filtering 196
 replacing using aggregate functions 43
 replacing with a constant 21
 setting for Java transformation 248
 skipping 24
number of cached values
 Sequence Generator property value 407
 Sequence Generator transformation property 410
NumRowsAffected port
 SQL transformation 475

example 258
 Java transformation 227
 On Receiving Transaction tab
 Java transformation 228
 operators
 See also Transformation Language Reference
 lookup condition 328
 Oracle
 connect string syntax 471
 stored procedure notes 496
 ORDER BY
 lookup query 324
 outer join
 See also join type
 creating 447
 creating as a join override 448
 creating as an extract override 448
 Integration Service supported types 442
 Output is Deterministic property
 Java transformation 223
 Output is Ordered property
 Java transformation 223
 output parameters
 stored procedures 491
 output ports
 default values 18
 dynamic Lookup transformation 347
 error handling 18
 NewLookupRow in Lookup transformation 347
 overview 8
 required for Expression transformation 142
 using as variables 227
 output row
 setting row type in Java transformation 249
 overriding
 default Source Qualifier SQL query 438
 lookup query

P

parameter access functions
 64-bit 184
 description 183
 parameter binding
 SQL transformation queries 465
 partition related functions
 description 186
 partitioned pipelines
 joining sorted data 292
 passive transformations
 Expression 142

External Procedure transformation 146
 Java 215
 Lookup 308
 overview 2
 Sequence Generator 402
 Stored Procedure 490
 pass-through ports
 SQL transformation 469
 percentile
 See Aggregator transformation
 See also Transformation Language Reference
 performance
 Aggregator transformation 45
 improving filter 195
 Joiner transformation 306
 logical database connections 473
 Lookup transformation 336
 static database connections 473
 using variables to improve 14
 persistent lookup cache
 named and unnamed 342
 named files 363
 overview 342
 recache from database 340
 sharing 363
 pipeline partitioning
 Custom transformation 65
 External Procedure transformation 147
 HTTP transformation 203
 pipelines
 Joiner transformation 284
 merging with Union transformation 532
 port attributes
 editing 63
 overview 62
 port dependencies
 Custom transformation 60
 port values
 Java transformation 220
 ports
 Aggregator transformation 38
 configuring 6
 creating 6, 7
 Custom transformation 59
 default values overview 18
 evaluation order 16
 group by 42
 HTTP transformation 205
 Java transformation 219
 Lookup transformation 313
 NewLookup Row in Lookup transformation 347

Rank transformation 384
 Router transformation 396
 Sequence Generator transformation 404
 sorted 45, 452
 sorted ports option 452
 Source Qualifier 452
 Union transformation 534
 variables ports 14
post-session
 errors 513
 stored procedures 509
pre- and post-session SQL
 Source Qualifier transformation 455
pre-session
 errors 512
 stored procedures 509
primary key
 creating with Sequence Generator transformation 403
primitive datatypes
 Java transformation 215
promoting
 non-reusable transformations 32
property access function
 description 182
property functions
 Custom transformation 108
property IDs
 Custom transformation 109

Q

queries
 Lookup transformation 324
 overriding lookup 324
 Source Qualifier transformation 431, 438
query
See lookup query
query mode
 rules and guidelines 469
 SQL transformation 464
quoted identifiers
 reserved words 431

R

Rank transformation
 creating 386
 defining groups for 385
 options 383
 overview 382
 ports 384
 RANKINDEX port 384
 using variables 14
ranking
 groups of data 385
 string values 383
real-time data
 processing with Joiner transformation 301
 with Joiner transformation 300
recache from database
 named cache 342
 overview 340
 unnamed cache 342
registering
 COM procedures with repositories 153
reinitializing lookup cache
See recache from database
reject file
 update strategies 541
relational databases
 joining 284
repositories
 COM external procedures 153
 registering COM procedures with 153
Requires Single Thread per Partition property
 HTTP transformation 203
 Java transformation 223
Requires Single Thread property
 Custom transformation 66
reserved words
 generating SQL with 431
 lookup query 326
 resword.txt 431
Reset property
 Sequence Generator transformation 408, 411
return port
 Lookup transformation 314, 332
return values
 from external procedures 172
 Lookup transformation 332
 Stored Procedure transformation 491
reusable transformations
 adding to mappings 33
 changing 34
 creating 32
 creating a non-reusable instance 33
 mapping variables 31
 overview 31
 reverting to original 35
Revert button
 in reusable transformations 35

right outer join
 creating 446
 syntax 446
 rollback
 example for Java transformation 247
 Java transformation API method 247
 syntax for Java transformation 247
 rolling back data
 Java transformation 247
 Router transformation
 connecting in mappings 398
 creating 400
 example 394
 group filter condition 393
 groups 392
 overview 390
 ports 396
 routing rows
 transformation for 390
 row strategy functions
 array-based 135
 row-based 128
 Row transformation scope
 behavior in Joiner transformation 300
 row-based functions
 data handling 117
 row strategy 128
 rows
 deleting 547
 flagging for update 541
 rules
 default values 28

S

script mode
 rules and guidelines 462
 SQL transformation 461
 ScriptError port
 description 462
 ScriptName port
 SQL transformation 462
 ScriptResults port
 SQL transformation 462
 scripts locale option
 SQL transformation 485
 select distinct
 overriding in sessions 454
 Source Qualifier option 454
 Sequence Generator transformation
 creating 412

creating composite key 403
 creating primary and foreign keys 403
 Current Value 409
 CURRVAL port 406
 cycle 408
 End Value property 409
 Increment By property 408
 NEXTVAL port 404
 non-reusable 410
 number of cached values 410
 overview 402
 ports 404
 properties 407
 replacing missing values 403
 reset 411
 reusable 411
 start value 408
 using IIF function to replace missing keys 403
 sequence ID
 Lookup transformation 349
 sequential caches
 See caches
 session logs
 Java transformation 245, 246
 tracing levels 30
 sessions
 \$\$\$\$SessStartTime 427
 configuring to handle stored procedure errors 512
 External Procedure transformation 156
 improving performance through variables 14
 incremental aggregation 38
 overriding select distinct 454
 running pre- and post-stored procedures 509
 setting update strategy 544
 skipping errors 22
 Stored Procedure transformation 491
 setNull
 example for Java transformation 248
 Java transformation API method 248
 syntax for Java transformation 248
 setOutRowType
 example for Java transformation 249
 Java transformation API method 249
 syntax for Java transformation 249
 sharing
 named lookup caches 365
 unnamed lookup caches 363
 simple interface
 API methods 271
 example 272
 Java expressions 271

sort order
 Aggregator transformation 45
 configuring for Joiner transformation 292
 Source Qualifier transformation 452

sort origin
 configuring the join condition to use 292
 definition 292

sorted data
 joining from partitioned pipelines 292
 using in Joiner transformation 292

sorted flat files
 using in Joiner transformation 292

sorted input
 flat file lookups 311

Sorted Input property
 Aggregator transformation 45
 Joiner transformation 287

sorted ports
 Aggregator transformation 45
 caching requirements 39
 pre-sorting data 45
 reasons not to use 45
 sort order 452
 Source Qualifier 452

sorted relational data
 using in Joiner transformation 292

Sorter transformation
 configuring 419
 configuring Sorter Cache Size 419
 creating 423
 overview 416
 properties 419
 using with Joiner transformation 292

\$Source
 multiple sources 322, 502
 Lookup transformations 317
 Stored Procedure transformations 501

Source Analyzer
 creating key relationships 436

source filters
 adding to Source Qualifier 450

Source Qualifier transformation
 \$\$\$\$SessStartTime 427
 configuring 456
 creating key relationships 436
 custom joins 435
 datatypes 426
 default join 434
 default query 431
 entering source filter 450
 entering user-defined join 440

joining source data 434
joins 436
mapping parameters and variables 427
Number of Sorted Ports option 452
outer join support 442
overriding default query 432, 438
overview 426
pre- and post-session SQL 455
properties 429, 457
Select Distinct option 454
sort order with Aggregator 46
SQL override 438
target load order 426
troubleshooting 458
viewing default query 432
XML Source Qualifier 550

sources
 joining 284
 joining data from the same source 296
 joining multiple 284
 merging 532

SQL
 adding custom query 438
 overriding default query 432, 438
 viewing default query 432

SQL Ports tab
 SQL transformation 486

SQL query
 adding custom query 438
 overriding default query 432, 438
 viewing default query 432

SQL settings tab
 SQL transformation 485

SQL statements
 supported by SQL transformation 488

SQL transformation
 advanced options 472
 configuring connections 470
 database resilience 478
 description 460
 dynamic SQL queries 466
 native datatype column 465
 NumRowsAffected 475
 passing full connection information 470
 pass-through ports 469
 properties 482, 483
 query mode 464
 script mode 461
 ScriptError port 462
 ScriptName port 462
 ScriptResults port 462

SELECT queries 465
 setting SQL attributes 485
 setting Verbose Data 483
 SQL ports description 486
 static query ports 466
 static SQL queries 465
 supported SQL statements 488
 transaction control 478
 using string substitution 467
 standard deviation
 See Aggregator transformation
 See also Transformation Language Reference
 Start Value property
 Sequence Generator transformation 407, 408
 static code
 Java transformation 226
 static database connection
 description 470
 performance considerations 473
 static lookup cache
 overview 344
 static SQL queries
 configuring ports 466
 SQL transformation 465
 static variable
 Java transformation 226
 status codes
 Stored Procedure transformation 491
 Stored Procedure transformation
 call text 501
 configuring 494
 configuring connected stored procedure 504
 configuring unconnected stored procedure 506
 connected 491
 creating by importing 498, 499
 creating manually 500, 501
 execution order 502
 expression rules 516
 importing stored procedure 498
 input data 490
 input/output parameters 491
 modifying 503
 output data 490
 overview 490
 performance tips 517
 pre- and post-session 509
 properties 501
 return values 491
 running pre- or post-session 509
 setting options 501
 specifying session runtime 492
 specifying when run 492
 status codes 491
 troubleshooting 518
 unconnected 491, 506
 stored procedures
 See also Stored Procedure transformation
 changing parameters 503
 creating sessions for pre or post-session run 510
 database-specific syntax notes 495
 definition 490
 error handling 512
 IBM DB2 example 497
 importing 499
 Informix example 495
 load types 511
 Microsoft example 496
 Oracle example 496
 post-session errors 513
 pre-session errors 512
 session errors 513
 setting type of 502
 specifying order of processing 492
 supported databases 514
 Sybase example 496
 Teradata example 497
 writing 495
 writing to variables 15
 string substitution
 SQL transformation queries 467
 strings
 ranking 383
 Sybase ASE
 connect string syntax 471
 ORDER BY limitation 326
 stored procedure notes 496
 syntax
 common database restrictions 449
 creating left outer joins 445
 creating normal joins 443
 creating right outer joins 447

T

tables
 creating key relationships 436
 \$Target
 multiple targets 322, 502
 Lookup transformations 317
 Stored Procedure transformations 501
 Target Designer
 automatic COBOL normalization 374

- target load order
 - Source Qualifier 426
- target tables
 - deleting rows 547
 - inserts 547
 - setting update strategy for 545
- targets
 - updating 540
- TC_COMMIT_AFTER constant
 - description 522
- TC_COMMIT_BEFORE constant
 - description 522
- TC_CONTINUE_TRANSACTION constant
 - description 522
- TC_ROLLBACK_AFTER constant
 - description 522
- TC_ROLLBACK_BEFORE constant
 - description 522
- Teradata
 - connect string syntax 471
- Terse tracing level
 - defined 30
- threads
 - Custom transformation 66
- thread-specific operations
 - Custom transformations 66
 - HTTP transformations 203
- TINFPParam parameter type
 - definition 174
- tracing level function
 - description 187
- Tracing Level property
 - Java transformation 222
- tracing levels
 - Joiner transformation property 287
 - Normal 30
 - overriding 30
 - overview 30
 - Sequence Generator transformation property 408
 - session 30
 - session properties 48
 - Terse 30
 - Verbose Data 30
 - Verbose Initialization 30
- tracing messages
 - for external procedures 173
- transaction
 - definition 521
 - generating 68, 224, 239
 - working with in Joiner transformation 300
- transaction boundaries
- Custom transformation 69
- transaction boundary
 - dropping in Joiner transformation 302
 - preserving in Joiner transformation 301
- transaction control
 - Custom transformation 68
 - example 522
 - expression 522
 - Java transformation 223
 - overview 520
 - SQL transformation 478
 - transformation 521
- Transaction Control transformation
 - creating 529
 - effective 524
 - in mappings 524
 - ineffective 524
 - mapping validation 528
 - overview 521
 - properties 521
- Transaction transformation scope
 - behavior in Joiner transformation 300
- Transformation Exchange (TX)
 - definition 146
- transformation language
 - aggregate functions 40
 - using with Java expressions 264
- transformation scope
 - All Input transformation scope with Joiner transformation 300
 - Custom transformation 68
 - defining for Joiner transformation 300
 - Java transformation 223
 - Joiner transformation property 287
 - Row transformation scope with Joiner transformation 300
 - Transaction transformation scope with Joiner transformation 300
- Transformation Scope property
 - Java transformation 222
- transformations
 - See also* active transformations
 - See also* connected transformations
 - See also* passive transformations
 - See also* unconnected transformations
 - active and passive 2
 - adding to mappings 5
 - Aggregator 38
 - connected 2
 - creating 5
 - Custom 54

definition 2
 descriptions 2
 Expression 142
 External Procedure 146
 Filter 190
 handling errors 25
 Joiner 284
 Lookup 308
 making reusable 32
 Midstream XML Generator 552
 Midstream XML Parser 551
 multi-group 9
 Normalizer 372
 overview 2
 promoting to reusable 32
 Rank 382
 reusable transformations 31
 Router 390
 Sequence Generator 402
 Source Qualifier 426
 SQL 460
 Stored Procedure 490
 tracing levels 30
 types that allow for expressions 11
 unconnected 2
 Union 532
 Update Strategy 540
 XML Source Qualifier 550
Treat Source Rows As
 update strategy 544
troubleshooting
 Aggregator transformation 51
 Filter transformation 196
 Java transformation 232
 Normalizer transformation 379
 Source Qualifier transformation 458
 Stored Procedure transformation 518
TX-prefixed files
 external procedures 162

U

unconnected Lookup transformation
 input ports 331
 return port 332
unconnected lookups
See also Lookup transformation
 adding lookup conditions 332
 calling through expressions 333
 description 309
 designating return values 332

overview 331
unconnected transformations
 External Procedure transformation 146, 175
 Lookup 308
 Lookup transformation 331
 Stored Procedure transformation 490
Unicode mode
See also Administrator Guide
 configuring sort order for Joiner transformation 292
 Custom transformation 55
 External Procedure Transformation 146
Union transformation
 components 532
 creating 536
 groups 534
 guidelines 532
 overview 532
 ports 534
unnamed cache
 persistent 342
 recache from database 342
 sharing 363
unsorted Joiner transformation
 processing detail rows 299
 processing master rows 299
update strategy
 setting with a Custom transformation 66
 setting with a Java transformation 224
Update Strategy transformation
 Aggregator combination 542
 checklist 547
 creating 541
 entering expressions 541
 forwarding rejected rows 541
 Lookup combination 543
 overview 540
 setting options for sessions 544, 545
 steps to configure 540
Update Strategy Transformation property
 Java transformation 222
URL
 adding through business documentation links 12
 user-defined functions
 using with Java expressions 264
 user-defined group
 Router transformation 392
 user-defined joins
 entering 440
 user-defined methods
 Java transformation 227, 228

V

Validate button
transformations 28

validating
default values 28
expressions 13

values
calculating with Expression transformation 142

variable ports
overview 14

variables
capturing stored procedure results 15
initializations 16
Java transformation 226, 227, 228
overview 14
port evaluation order 16

Verbose Data tracing level
overview 30
SQL transformation 483

Verbose Initialization tracing level
overview 30

Visual Basic
adding functions to Integration Service 173
Application Setup Wizard 169
code for external procedures 147
COM datatypes 171
developing COM external procedures 156
distributing procedures manually 170
wrapper classes for 173

Visual C++
adding libraries to Integration Service 173
COM datatypes 171
developing COM external procedures 149
distributing procedures manually 170
wrapper classes for 173

X

XML transformations
See also XML Guide
Midstream XML Generator 552
Midstream XML Parser 551
Source Qualifier 550

W

web links
adding to expressions 12

Windows systems
compiling DLLs on 165

wizards
ATL COM AppWizard 149
MFC AppWizard 165
Visual Basic Application Setup Wizard 169

wrapper classes
for pre-existing libraries or functions 173