



Informatica®
10.5.4

Developer Transformation Guide

Informatica Developer Transformation Guide
10.5.4
May 2023

© Copyright Informatica LLC 2009, 2023

This software and documentation are provided only under a separate license agreement containing restrictions on use and disclosure. 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 LLC.

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation is subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License.

Informatica, the Informatica logo, and PowerCenter are trademarks or registered trademarks of Informatica LLC in the United States and many jurisdictions throughout the world. A current list of Informatica trademarks is available on the web at <https://www.informatica.com/trademarks.html>. Other company and product names may be trade names or trademarks of their respective owners.

Subject to your opt-out rights, the software will automatically transmit to Informatica in the USA information about the computing and network environment in which the Software is deployed and the data usage and system statistics of the deployment. This transmission is deemed part of the Services under the Informatica privacy policy and Informatica will use and otherwise process this information in accordance with the Informatica privacy policy available at <https://www.informatica.com/in/privacy-policy.html>. You may disable usage collection in Administrator tool.

Portions of this software and/or documentation are subject to copyright held by third parties. Required third party notices are included with the product.

See patents at <https://www.informatica.com/legal/patents.html>.

The information in this documentation is subject to change without notice. If you find any problems in this documentation, report them to us at infa_documentation@informatica.com.

Informatica products are warranted according to the terms and conditions of the agreements under which they are provided. INFORMATICA PROVIDES THE INFORMATION IN THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT.

Publication Date: 2023-05-11

Table of Contents

Preface	30
Informatica Resources.	30
Informatica Network.	30
Informatica Knowledge Base.	30
Informatica Documentation.	30
Informatica Product Availability Matrices.	31
Informatica Velocity.	31
Informatica Marketplace.	31
Informatica Global Customer Support.	31
Chapter 1: Introduction to Transformations	32
Introduction to Transformations Overview.	32
Active Transformations.	32
Passive Transformations.	33
Unconnected Transformations.	33
Multi-Strategy Transformations.	33
Transformation Descriptions.	34
Transformations in the Native and Non-native Environments.	36
Handling Transformation Data Types.	39
Decimal Data Type.	39
Timestamp with Time Zone.	40
Timestamp with Local Time Zone.	41
Developing a Transformation.	41
Multi-Group Transformations.	41
Rules and Guidelines for Multi-Group Transformations.	42
Expressions in Transformations.	42
The Expression Editor.	43
Port Names in an Expression.	44
Adding an Expression to a Port.	44
Comments in an Expression.	44
Expression Validation.	44
Test Expressions.	44
Data Type Conversion.	45
Local Variables.	46
Temporarily Store Data and Simplify Complex Expressions.	46
Store Values Across Rows.	47
Capture Values from Stored Procedures.	48
Guidelines for Configuring Variable Ports.	48
Variable Initialization.	49
Default Values for Ports.	49

User-Defined Default Values.	50
User-Defined Default Input Values.	51
Default Value Validation.	53
User-Defined Default Output Values	53
General Rules for Default Values.	55
Default Value Validation.	55
Tracing Levels.	55
Reusable Transformations.	56
Reusable Transformation Instances and Inherited Changes.	56
Editing a Reusable Transformation.	57
Editor Views for a Reusable Transformation.	57
Non-Reusable Transformations.	57
Editor Views for a Non-Reusable Transformation.	57
Creating a Transformation.	58
Chapter 2: Transformation Ports.....	59
Transformation Ports Overview.	59
Create Ports.	59
Configure Ports.	60
Linking Ports.	60
One to One Links.	61
One to Many Links.	61
Manually Linking Ports.	61
Automatically Linking Ports.	61
Rules and Guidelines for Linking Ports.	62
Propagating Port Attributes.	63
Dependency Types.	63
Link Path Dependencies.	63
Implicit Dependencies.	63
Propagated Port Attributes by Transformation.	64
Copying Ports from Excel.	66
Editing Transformations in Excel.	67
Copying Metadata to the Developer Tool.	67
Example: Editing a Transformation in Excel.	68
Rules and Guidelines for Copying from Excel.	68
Chapter 3: Transformation Caches.....	69
Transformation Caches Overview.	69
Cache Types.	70
Cache Files.	70
Cache File Directory.	71
Cache Size.	71
Auto Cache Size.	71

Specific Cache Size.	72
Cache Size Increase by the Data Integration Service.	73
Cache Size for Partitioned Caches.	73
Cache Size Optimization.	74
Step 1. Set the Tracing Level to Verbose Initialization.	74
Step 2. Run the Mapping in Auto Cache Mode.	74
Step 3. Analyze Caching Performance.	74
Step 4. Configure Specific Cache Sizes.	75
Chapter 4: Address Validator Transformation.	76
Address Validator Transformation Overview.	76
Address Reference Data.	77
Types of Address Reference Data.	77
Modes and Templates.	79
Port Groups and Port Selection.	79
Address Validator Transformation Input Port Groups.	80
Address Validator Transformation Output Port Groups.	80
Multiple-Instance Ports.	83
Address Validation Projects.	84
Formatted Addresses and Mail Carrier Standards.	85
Partial Address Completion	86
Address Validator Status Ports.	87
Element Status Code Definitions.	88
Address Resolution Code Output Port Values.	90
Element Input Status Output Port Values.	90
Element Relevance Output Port Values.	91
Element Result Status Output Port Values.	92
Extended Element Result Status Output Port Values.	93
Mailability Score Output Port Values.	94
Match Code Output Port Values.	95
Geocoding Status Output Port Values.	97
Address Validator Transformation General Settings.	98
Address Validation Properties in the Preferences Window.	99
Address Validation Data Properties.	100
Address Validation License Properties.	101
Address Validation Engine Properties.	101
Address Validation Advanced Properties.	102
Alias Locality.	103
Alias Street.	103
Casing Style.	103
Country of Origin.	104
Country Type.	104
Default Country.	105

Dual Address Priority.	106
Element Abbreviation.	106
Execution Instances.	106
Flexible Range Expansion.	107
Geocode Data Type.	108
Global Max Field Length.	108
Global Preferred Descriptor.	109
Input Format Type.	109
Input Format With Country	109
Line Separator.	110
Matching Alternatives.	110
Matching Extended Archive.	111
Matching Scope.	111
Max Result Count.	112
Mode.	112
Optimization Level.	112
Output Format Type.	113
Output Format With Country.	113
Preferred Language.	113
Preferred Script.	119
Ranges To Expand.	120
Standardize Invalid Addresses.	121
Tracing Level.	121
Certification Reports.	121
AMAS Report Fields.	122
CASS Report Fields.	123
SendRight Report.	123
SERP Report Fields.	124
Configuring an Address Validator Transformation.	124
Adding Ports to the Address Validator Transformation.	125
Creating User-Defined Templates.	125
Defining Address Validator Models.	126
Defining a Certification Report.	126
Address Validator Transformation in a Non-native Environment.	127
Address Validator Transformation on the Blaze Engine.	127
Address Validator Transformation on the Spark Engine.	127
Address Validator Transformation on the Databricks Spark Engine.	127
Chapter 5: Aggregator Transformation.	128
Aggregator Transformation Overview.	128
Aggregator Transformations in Dynamic Mappings.	129
Developing an Aggregator Transformation.	129
Aggregator Transformation Ports.	129

Aggregate Expressions.	130
Aggregate Functions.	131
Nested Aggregate Functions.	132
Conditional Clauses in Aggregate Expressions.	132
Group By Ports.	132
Configure Group By Ports.	133
Group By Parameters.	134
Default Values of Group By Ports	134
Non-Aggregate Expressions.	134
Aggregator Caches.	135
Sorted Input for an Aggregator Transformation.	135
Sorted Input Conditions.	135
Sorting Data in an Aggregator Transformation.	136
Aggregator Transformation Advanced Properties.	137
Creating a Reusable Aggregator Transformation.	137
Creating a Non-Reusable Aggregator Transformation.	138
Tips for Aggregator Transformations.	138
Troubleshooting Aggregator Transformations.	139
Aggregator Transformation in a Non-native Environment.	139
Aggregator Transformation on the Blaze Engine.	139
Aggregator Transformation on the Spark Engine.	140
Aggregator Transformation on the Databricks Spark Engine.	141
Chapter 6: Association Transformation.	142
Association Transformation Overview.	142
Memory Allocation.	143
Association Transformation Advanced Properties.	144
Chapter 7: Bad Record Exception Transformation.	145
Bad Record Exception Transformation Overview.	145
Bad Record Exception Output Record Types.	146
Bad Record Exception Management Process Flow.	147
Bad Record Exception Mappings.	147
Bad Record Exception Quality Issues.	148
Human Tasks.	149
Bad Record Exception Ports	149
Bad Record Exception Transformation Input Ports.	150
Bad Record Exception Transformation Output.	150
Bad Record Exception Configuration View.	151
Generating the Bad Records Table and the Issues Table.	152
Bad Record Exception Issue Assignment	153
Assigning Ports to Quality Issues.	153
Exception Transformation Advanced Properties.	154

Configuring a Bad Record Exception Transformation.	154
Bad Record Exception Mapping Example.	155
Bad Record Exception Mapplet.	155
Bad Record Exception Example Input Groups.	156
Bad Record Exception Example Configuration	157
Bad Record Exception Example Mapping Output.	158
Chapter 8: Case Converter Transformation.	160
Case Converter Transformation Overview.	160
Case Strategy Properties.	160
Configuring a Case Converter Strategy.	161
Case Converter Transformation Advanced Properties.	161
Case Converter Transformation in a Non-native Environment.	162
Chapter 9: Classifier Transformation.	163
Classifier Transformation Overview.	163
Classifier Models.	163
Classifier Algorithms.	164
Classifier Transformation Options.	164
Classifier Strategies.	165
Classifier Transformation Advanced Properties.	165
Configuring a Classifier Strategy.	165
Classifier Analysis Example.	166
Create the Classifier Mapping.	167
Input Data Sample.	168
Data Source Configuration.	168
Classifier Transformation Configuration.	168
Router Transformation Configuration.	169
Data Target Configuration.	170
Classifier Mapping Outcome.	170
Classifier Transformation in a Non-native Environment.	171
Chapter 10: Comparison Transformation.	172
Comparison Transformation Overview.	172
Field Matching Strategies.	172
Bigram.	173
Hamming Distance.	173
Edit Distance.	173
Jaro Distance.	174
Reverse Hamming Distance.	174
Identity Matching Strategies.	175
Configuring a Comparison Strategy.	175
Comparison Transformation Advanced Properties.	176

Comparison Transformation in a Non-native Environment.	176
--	-----

Chapter 11: Consolidation Transformation..... 177

Consolidation Transformation Overview.	177
Consolidation Mappings.	178
Consolidation Transformation Ports.	178
Consolidation Transformation Views.	178
Consolidation Transformation Strategies View.	179
Consolidation Transformation Advanced Properties.	179
Cache File Size.	180
Simple Strategies.	181
Row-Based Strategies.	182
Advanced Strategies.	183
Simple Consolidation Functions.	183
CONSOL_AVG.	183
CONSOL_LONGEST.	184
CONSOL_MAX.	184
CONSOL_MIN.	185
CONSOL_MOSTFREQ.	185
CONSOL_MOSTFREQ_NB.	186
CONSOL_SHORTEST.	187
Row-Based Consolidation Functions.	187
CONSOL_GETROWFIELD.	187
CONSOL_MODAL EXACT.	188
CONSOL_MOSTDATA.	189
CONSOL_MOSTFILLED.	190
Consolidation Mapping Example.	191
Input Data.	191
Key Generator Transformation.	191
Consolidation Transformation.	192
Consolidation Mapping Output.	192
Configuring a Consolidation Transformation.	192
Consolidation Transformation in a Non-native Environment.	193
Consolidation Transformation on the Blaze Engine.	193
Consolidation Transformation on the Spark Engine.	193
Consolidation Transformation on the Databricks Spark Engine.	193

Chapter 12: Data Masking Transformation..... 194

Data Masking Transformation Overview.	194
Masking Techniques.	194
Random Masking.	195
Expression Masking.	197
Key Masking.	199

Substitution Masking.	200
Dependent Masking.	203
Tokenization Masking.	205
Encryption.	205
Masking Rules.	206
Mask Format.	207
Source String Characters.	208
Result String Replacement Characters.	208
Range.	209
Blurring.	209
Special Mask Formats.	210
Credit Card Number Masking.	211
Email Address Masking.	211
Advanced Email Masking.	211
IP Address Masking.	212
Phone Number Masking.	212
Social Security Number Masking.	213
URL Address Masking.	213
Social Insurance Number Masking.	214
Default Value File.	214
Data Masking Transformation Configuration.	215
Configure the Data Integration Service.	215
Creating a Data Masking Transformation.	215
Defining the Ports.	216
Configuring Data Masking for Each Port.	216
Previewing the Masked Data.	216
Data Masking Transformation Runtime Properties.	217
Data Masking Example.	218
Read_Customer Data.	218
Customer Data Masking Transformation.	219
Customer Test Data Results.	220
Data Masking Transformation Advanced Properties.	220
Data Masking Transformation in a Non-native Environment.	220
Data Masking Transformation on the Blaze Engine.	220
Data Masking Transformation on the Spark Engine.	221
Chapter 13: Data Processor Transformation.	223
Data Processor Transformation Overview.	223
Data Processor Transformation Views.	224
Data Processor Transformation Ports.	225
Data Processor Transformation Input Ports.	225
Data Processor Transformation Output Ports.	226
Pass-Through Ports.	227

Startup Component.	227
References.	227
Data Processor Transformation Settings.	228
Character Encoding.	228
Rules and Guidelines for Character Encoding.	231
Output Settings.	231
Processing Settings.	232
XMap Settings.	233
XML Output Configuration.	233
Events.	235
Event Types.	235
Data Processor Events View.	235
Logs.	236
Design-Time Event Log.	236
Run-Time Event Log.	237
Viewing an Event Log in the Data Processor Events View.	237
User Log.	237
Data Processor Transformation Development.	238
Create the Data Processor Transformation.	238
Select the Schema Objects	239
Create Objects in a Blank Data Processor Transformation.	239
Create the Ports.	241
Testing the Transformation.	242
Data Processor Transformation Export and Import.	242
Exporting the Data Processor Transformation as a Service.	242
Importing Multiple Data Transformation Services.	243
Importing a Data Transformation Service	243
Exporting a Mapping with a Data Processor Transformation to PowerCenter.	244
Data Processor Transformation Validation.	244
Using a Speed-enhanced Data Transformation Engine for VRL Validations.	245
Data Processor Transformation in a Non-native Environment.	245
Chapter 14: Decision Transformation.	246
Decision Transformation Overview.	246
Decision Transformation Functions.	247
Decision Transformation Conditional Statements	249
Decision Transformation Operators.	250
Decision Transformation NULL Handling.	251
Configuring a Decision Strategy	251
Decision Transformation Advanced Properties.	251
Decision Transformation in a Non-native Environment.	252
Decision Transformation on the Spark Engine.	252
Decision Transformation on the Databricks Spark Engine.	252

Chapter 15: Duplicate Record Exception Transformation.....	253
Duplicate Record Exception Transformation Overview.	253
Duplicate Record Exception Process Flow.	254
Duplicate Record Exceptions.	254
Duplicate Record Exception Configuration View	255
Generating a Duplicate Records Table.	256
Ports.	256
Duplicate Record Exception Transformation Input Ports.	257
Duplicate Record Exception Transformation Output Ports.	257
Creating Ports	258
Duplicate Record Exception Transformation Advanced Properties.	259
Duplicate Record Exception Mapping Example.	259
Duplicate Record Exception Mapping.	260
Match Transformation.	260
Duplicate Record Exception Input Groups.	261
Duplicate Record Exception Example Configuration View.	261
Standard Output Table Records.	262
Cluster Output.	263
Creating a Duplicate Record Exception Transformation.	264
Chapter 16: Expression Transformation.....	266
Expression Transformation Overview.	266
Expression Transformation Ports.	267
Test Expressions.	268
Date Format Strings for Sample Data.	269
Testing an Expression.	269
Port Selectors.	269
Port Selector Configuration.	270
Selection Rules.	270
Creating a Port Selector.	271
Windowing.	272
Windowing Configuration.	273
Dynamic Expressions.	276
Output Port Settings.	277
Creating a Dynamic Expression	279
Flatten a Dynamic Struct.	280
Expression Transformation Advanced Properties.	281
Expression Transformation in a Non-native Environment.	281
Expression Transformation on the Blaze Engine.	282
Expression Transformation on the Spark Engine.	282
Expression Transformation on the Databricks Spark Engine.	282

Chapter 17: Filter Transformation.....	283
Filter Transformation Overview.	283
Filter Transformations in Dynamic Mappings.	284
Filter Condition.	285
Parameterize the Filter Condition.	285
Filtering Rows with Null Values.	287
Filter Transformation Advanced Properties.	287
Filter Transformation Performance Tips.	287
Filter Transformation in a Non-native Environment.	287
Filter Transformation on the Blaze Engine.	288
Chapter 18: Hierarchical to Relational Transformation.....	289
Hierarchical to Relational Transformation Overview.	289
Example - Hierarchical to Relational Transformation.	289
Output Relational Ports and the Overview View.	291
Hierarchical to Relational Transformation Ports.	292
Schema References.	293
Port Configuration.	293
Hierarchical to Relational Transformation Development.	293
Creating the Hierarchical to Relational Transformation	294
Configuring the Ports and Mapping.	294
Testing the Transformation.	295
Chapter 19: Java Transformation.....	296
Java Transformation Overview.	296
Reusable and Non-Reusable Java Transformations.	297
Active and Passive Java Transformations.	297
Data Type Conversion.	297
Complex Data Type Conversion on the Spark Engine.	298
Designing a Java Transformation.	300
Java Transformation Ports.	300
Creating Ports.	300
Setting Default Port Values.	300
Java Transformation Advanced Properties.	301
Configuring the Classpath for the Developer Tool Client.	303
Configuring the Classpath for the Data Integration Service.	304
Developing Java Code.	304
Creating Java Code Snippets.	306
Importing Java Packages.	307
Defining Helper Code.	307
Java Transformation Java Properties.	308
Imports Tab.	308

Helpers Tab.	309
On Input Tab.	309
At End Tab.	310
Functions Tab.	310
Full Code Tab.	310
Filter Optimization with the Java Transformation.	310
Early Selection Optimization with the Java Transformation.	311
Push-Into Optimization with the Java Transformation.	312
Creating a Java Transformation.	313
Creating a Reusable Java Transformation.	313
Creating a Non-Reusable Java Transformation.	314
Compiling a Java Transformation.	314
Troubleshooting a Java Transformation.	315
Finding the Source of Compilation Errors.	315
Identifying the Source of Compilation Errors.	315
Converting to Struct Data Example.	316
Java Transformation in a Non-native Environment.	319
Java Transformation on the Blaze Engine.	319
Java Transformation on the Spark Engine.	320
Chapter 20: Java Transformation API Reference.	322
Java Transformation API Methods Overview.	322
defineJExpression.	323
failSession.	324
generateRow.	324
getInRowType.	325
getMetadata.	326
incrementErrorCount.	326
invokeJExpression.	327
isNull.	328
logError.	328
logInfo.	329
resetNotification.	329
setNull.	330
storeMetadata.	331
Chapter 21: Java Expressions.	332
Java Expressions Overview.	332
Expression Function Types.	333
Using the Define Function Dialog Box to Define an Expression.	333
Step 1. Configure the Function.	334
Step 2. Create and Validate the Expression.	334
Step 3. Generate Java Code for the Expression.	334

Creating an Expression and Generating Java Code by Using the Define Function Dialog Box.	334
Java Expression Templates.	335
Working with the Simple Interface.	335
invokeJExpression.	335
Simple Interface Example.	336
Working with the Advanced Interface.	337
Invoking an Expression with the Advanced Interface.	337
Rules and Guidelines for Working with the Advanced Interface.	337
EDataType Class.	338
JExprParamMetadata Class.	338
defineJExpression.	339
JExpression Class.	340
Advanced Interface Example.	340
JExpression Class API Reference.	341
getBytes.	341
getDouble.	341
getInt.	342
getLong.	342
getResultDataType.	342
getResultMetadata.	342
getStringBuffer.	343
invoke.	343
isResultNull.	343
Chapter 22: Joiner Transformation.	344
Joiner Transformation Overview.	344
Joiner Transformation Advanced Properties.	345
Joiner Caches.	346
Joiner Transformation Ports.	347
Joiner Transformations in Dynamic Mappings.	348
Port Selectors in a Joiner Transformation.	348
Selection Rules	349
Creating a Port Selector.	350
Defining a Join Condition.	351
Simple Condition Type.	352
Advanced Condition Type.	352
Port Selectors in Join Conditions.	353
Dynamic Ports in Join Conditions.	354
Expression Parameter.	354
Join Types.	354
Normal Join.	355
Master Outer Join.	356
Detail Outer Join.	356

Full Outer Join.	356
Sorted Input for a Joiner Transformation.	357
Configuring the Sort Order.	357
Adding Transformations to the Mapping.	357
Rules and Guidelines for Join Conditions.	358
Example of a Join Condition and Sort Order.	359
Joining Data from the Same Source.	360
Joining Two Branches of the Same Pipeline.	360
Joining Two Instances of the Same Source.	361
Guidelines for Joining Data from the Same Source.	362
Blocking the Source Pipelines.	362
Unsorted Joiner Transformation.	362
Sorted Joiner Transformation.	362
Joiner Transformation Performance Tips.	363
Rules and Guidelines for a Joiner Transformation.	364
Joiner Transformation in a Non-native Environment	364
Joiner Transformation on the Blaze Engine.	364
Joiner Transformation on the Spark Engine.	364
Joiner Transformation on the Databricks Spark Engine.	365
Chapter 23: Key Generator Transformation.	366
Key Generator Transformation Overview.	366
Soundex Strategy.	367
Soundex Strategy Properties.	367
String Strategy.	367
String Strategy Properties.	368
NYSIS Strategy.	368
Key Generator Output Ports.	368
Configuring a Grouping Strategy.	369
Key Creation Properties.	369
Key Generator Transformation Advanced Properties.	370
Key Generator Transformation in a Non-native Environment.	370
Key Generator Transformation on the Blaze Engine.	371
Key Generator Transformation on the Spark Engine.	371
Key Generator Transformation on the Databricks Spark Engine.	371
Chapter 24: Labeler Transformation.	372
Labeler Transformation Overview.	372
When to Use a Labeler Transformation.	373
Reference Data Use in the Labeler Transformation.	374
Character Sets.	374
Probabilistic Models.	375
Reference Tables.	375

Regular Expressions.	375
Token Sets.	375
Labeler Transformation Strategies.	376
Character Labeling Operations.	376
Token Labeling Operations.	376
Labeler Transformation Ports.	377
Character Labeling Properties.	377
General Properties.	377
Reference Table Properties.	378
Character Set Properties.	378
Filter Properties.	379
Token Labeling Properties.	379
General Properties.	379
Token Set Properties.	380
Custom Label Properties.	380
Probabilistic Matching Properties.	381
Reference Table Properties.	381
Configuring a Character Labeling Strategy.	381
Configuring a Token Labeling Strategy.	382
Labeler Transformation Advanced Properties.	383
Labeler Transformation in a Non-native Environment.	383
Chapter 25: Lookup Transformation.....	384
Lookup Transformation Overview.	384
Connected and Unconnected Lookups.	385
Connected Lookups.	386
Unconnected Lookups.	387
Developing a Lookup Transformation.	387
Lookup Query.	388
Default Lookup Query.	388
SQL Override for a Lookup Query.	388
Parameters in an SQL Override Query.	389
Reserved Words.	389
Guidelines for Overriding the Lookup Query.	389
Overriding the Lookup Query.	390
Lookup Source Filter.	390
Filtering Source Rows in a Lookup.	391
Lookup Condition.	391
Configure the Lookup Condition.	392
Rules and Guidelines for Lookup Transformation Conditions.	393
Lookup Cache.	394
Query Properties.	394
Lookup Transformations in Dynamic Mappings.	395

Define Dynamic Ports.	395
Change the Lookup Source.	396
Parameterize the Lookup Source.	397
Lookup Sources that Contain Parameters	398
Configure Parameters in a Duplicate Data Object.	398
Port Selectors.	400
Port Selector Configuration.	401
Selection Rules.	402
Parameterize the Lookup Condition.	403
Creating a Port Selector.	404
Run-time Properties.	406
Advanced Properties.	407
Creating a Reusable Lookup Transformation.	408
Creating a Non-Reusable Lookup Transformation.	409
Creating an Unconnected Lookup Transformation.	410
Unconnected Lookup Example.	411
Lookup Transformation in a Non-native Environment.	413
Lookup Transformation on the Blaze Engine.	413
Lookup Transformation on the Spark Engine.	414
Lookup Transformation on the Databricks Spark Engine.	415
Lookup Transformation in a Streaming Mapping.	416
Chapter 26: Lookup Caches.	417
Lookup Caches Overview.	417
Lookup Cache Types.	418
Uncached Lookup.	419
Static Lookup Cache.	419
Persistent Lookup Cache.	419
Rebuilding a Persistent Lookup Cache.	420
Dynamic Lookup Cache.	420
Shared Lookup Cache.	421
Rules and Guidelines for Sharing a Lookup Cache.	421
Cache Comparison.	422
Cache Partitioning for Lookups.	422
Chapter 27: Dynamic Lookup Cache.	424
Dynamic Lookup Cache Overview.	424
Uses for a Dynamic Lookup Cache.	425
Dynamic Lookup Cache Properties.	425
Dynamic Lookup Cache and Output Values.	427
Lookup Transformation Values.	427
Lookup Transformation Values Example.	428
SQL Override and Dynamic Lookup Cache.	430

Mapping Configuration for a Dynamic Lookup Cache.	430
Insert Else Update.	431
Update Else Insert.	431
Dynamic Lookup Cache and Target Synchronization.	432
Conditional Dynamic Lookup Cache Updates.	433
Conditional Dynamic Lookup Cache Processing.	433
Configuring a Conditional Dynamic Lookup Cache.	433
Dynamic Cache Update with Expression Results.	434
Null Expression Values.	434
Expression Processing.	434
Configuring an Expression for Dynamic Cache Updates.	434
Dynamic Lookup Cache Example.	435
Rules and Guidelines for Dynamic Lookup Caches.	436
Chapter 28: Macro Transformation.	437
Macro Transformation Overview.	437
Macro Instruction.	437
Macro Transformation Parameters.	438
Macro Transformations in Dynamic Mappings.	438
Macro Transformation Example.	438
Macro Transformation in a Non-native Environment.	440
Chapter 29: Match Transformation.	441
Match Transformation Overview.	441
Match Analysis.	442
Column Analysis.	442
Single-Source Analysis and Dual-Source Analysis.	443
Field Match Analysis and Identity Match Analysis.	443
Groups in Match Analysis.	444
Match Pairs and Clusters.	445
Match Score Calculations.	445
Weighted Scores.	446
Null Match Scores.	446
Cluster Output Options.	446
Driver Scores and Link Scores in Cluster Analysis.	448
Master Data Analysis.	449
Mapping Reuse.	450
Identity Match Analysis and Persistent Index Data.	450
Rules and Guidelines for Persistent Index Data.	450
Match Mapping Performance.	451
Viewing Match Cluster Analysis Data.	452
Viewing Match Performance Analysis Data.	452
Match Performance in Identity Analysis.	453

Creating a Data Store for Identity Index Data.	454
Using the Index Data Store in Single-Source Analysis.	454
Match Transformation Views.	455
Match Transformation Ports.	457
Match Transformation Input Ports.	457
Match Transformation Output Ports.	458
Persistence Status Codes and Persistence Status Descriptions.	459
Output Ports and Match Output Selection.	461
Match Mapplets.	462
Creating a Match Mapplet.	462
Using a Match Mapplet.	462
Configuring a Match Analysis Operation.	463
Match Transformation in a Non-native Environment.	464
Match Transformation on the Blaze Engine.	464
Match Transformation on the Spark Engine.	464
Match Transformation on the Databricks Spark Engine.	464
Chapter 30: Match Transformations in Field Analysis.	465
Field Match Analysis.	465
Process Flow for Field Match Analysis.	465
Field Match Type Options.	466
Field Match Strategies.	466
Field Match Algorithms.	466
Field Match Strategy Properties.	469
Field Match Output Options.	469
Match Output Types.	469
Match Output Properties.	470
Field Match Advanced Properties.	471
Field Match Analysis Example.	471
Create the Mapping.	472
Input Data Sample.	472
Key Generator Transformation Configuration.	473
Match Transformation Configuration.	473
Run the Data Viewer.	475
Conclusion.	476
Chapter 31: Match Transformations in Identity Analysis.	477
Identity Match Analysis.	477
Process Flow for Identity Match Analysis.	478
Identity Match Type Properties.	478
Index Directory and Cache Directory Properties.	480
Persistence Method Parameters.	481
Identity Match Strategies.	482

Identity Match Algorithms.	482
Identity Match Strategy Properties.	483
Identity Match Output Options.	484
Match Output Types.	484
Match Output Properties.	485
Identity Match Advanced Properties.	487
Persistent Index Case Study.	488
Identity Match Analysis Example.	490
Create the Mapping.	490
Input Data Sample.	491
Expression Transformation Configuration.	491
Match Transformation Configuration.	492
Run the Data Viewer.	495
Conclusion.	496
Chapter 32: Normalizer Transformation.	497
Normalizer Transformation Overview.	497
Multiple-Occurring Fields.	498
Generated Column ID.	498
Multiple-Occurring Records.	499
Input Hierarchy Definition.	500
Normalizer Transformation Input Ports.	501
Merge Fields.	502
Flatten Fields.	503
Normalizer Transformation Output Groups and Ports.	508
Create an Output Group.	509
Update an Output Group.	511
Key Generation for Output Groups.	512
Normalizer Transformation Advanced Properties.	512
Generate First Level Output Groups.	513
Creating a Normalizer Transformation.	513
Creating a Normalizer Transformation from an Upstream Source.	514
Normalizer Mapping Example.	514
Normalizer Example Mapping.	514
Normalizer Example Definition.	515
Normalizer Example Input and Output Groups.	516
Normalizer Example Mapping Output.	517
Normalizer Transformation in a Non-native Environment.	518
Chapter 33: Merge Transformation.	519
Merge Transformation Overview.	519
Configuring a Merge Strategy.	519
Merge Transformation Advanced Properties.	520

Merge Transformation in a Non-native Environment.	520
Chapter 34: Parser Transformation.	521
Parser Transformation Overview.	521
Parser Transformation Modes.	522
When to Use a Parser Transformation.	522
Reference Data Use in the Parser Transformation.	523
Pattern Sets.	524
Probabilistic Models.	524
Reference Tables.	525
Regular Expressions.	525
Token Sets.	525
Token Parsing Operations.	525
Token Parsing Ports.	526
Token Parsing Properties.	526
General Properties.	527
Probabilistic Model Properties.	527
Reference Table Properties.	528
Token Set Properties.	528
Pattern-Based Parsing Mode.	529
Pattern-Based Parsing Ports.	529
Configuring a Token Parsing Strategy.	530
Configuring a Pattern Parsing Strategy.	530
Parser Transformation Advanced Properties.	531
Parser Transformation in a Non-native Environment.	531
Chapter 35: Python Transformation.	532
Chapter 36: Rank Transformation.	533
Rank Transformation Overview.	533
Ranking String Values.	534
Rank Transformation Properties.	534
Rank Transformations in Dynamic Mappings.	534
Rank Transformation Ports.	534
Rank Index.	535
Rank Port.	536
Define Group By Ports.	536
Group By Parameters.	537
Rank Caches.	537
Rank Transformation Advanced Properties.	538
Rank Transformation in a Non-native Environment.	539
Rank Transformation on the Blaze Engine.	539
Rank Transformation on the Spark Engine.	539

Rank Transformation on the Databricks Spark Engine.	540
Chapter 37: Read Transformation.	541
Read Transformation Overview.	541
Read Transformation Properties	541
General Properties.	542
Data Object Properties.	543
Query Properties.	543
Run-Time Properties	543
Sources Properties	544
Advanced Properties.	544
Synchronize Relational Data Objects.	545
Change the Source Data Object.	545
Parameterize the Read Transformation.	546
Read Transformation Parameters.	547
Constraints	548
Create a Read Transformation.	548
Creating a Read Transformation in the Mapping Editor.	549
Chapter 38: Relational to Hierarchical Transformation.	551
Relational to Hierarchical Transformation Overview.	551
Example - Relational to Hierarchical Transformation.	552
Input Relational Ports and the Overview View.	554
Relational to Hierarchical Transformation Ports.	554
Schema References.	555
Relational to Hierarchical Transformation Development.	555
Creating the Relational to Hierarchical Transformation.	555
Creating the Ports.	555
Chapter 39: REST Web Service Consumer Transformation.	557
REST Web Service Consumer Transformation Overview.	557
REST Web Service Consumer Transformation Process.	558
REST Web Service Consumer Transformation Configuration.	559
Message Configuration.	559
Resource Identification.	559
HTTP Methods.	560
HTTP Get Method.	561
HTTP Post Method.	561
HTTP Put Method.	562
HTTP Delete Method.	563
REST Web Service Consumer Transformation Ports.	563
Input Ports.	564
Output Ports.	564

Pass-through Ports.	564
Argument Ports.	564
URL Ports.	564
HTTP Header Ports.	565
Cookie Ports.	565
Output XML Ports.	565
Response Code Ports.	566
REST Web Service Consumer Transformation Input Mapping.	566
Rules and Guidelines to Map Input Ports to Elements.	566
Mapping Input Ports to the Method Input.	567
REST Web Service Consumer Transformation Output Mapping.	568
Rules and Guidelines to Map Elements to Output Ports.	569
Customize View Options.	569
Mapping the Method Output to Output Ports.	569
REST Web Service Consumer Transformation Advanced Properties.	570
REST Web Service Consumer Transformation Creation.	571
Creating a REST Web Service Consumer Transformation.	571
Parsing a JSON Response Message that Contains Arrays.	572
Example JSON Response Message.	572
Unnamed Arrays in a Response Message.	573
Chapter 40: Router Transformation.	574
Router Transformation Overview.	574
Router Transformations in Dynamic Mappings.	575
Working with Groups.	575
Input Group.	576
Output Groups.	576
Using Group Filter Conditions.	576
Dynamic Ports in Group Filter Conditions.	578
Parameterize the Group Filter.	578
Adding Groups.	579
Working with Ports.	579
Connecting Router Transformations in a Mapping.	580
Router Transformation Advanced Properties.	580
Router Transformation in a Non-native Environment.	580
Chapter 41: Sequence Generator Transformation.	581
Sequence Generator Transformation Overview.	581
Sequence Generator Ports.	581
Pass-Through Ports.	582
NEXTVAL Port.	582
Sequence Generator Transformation Properties.	584
Start Value.	584

End Value.	584
Increment Value.	585
Cycle Through a Range of Values.	585
Sequence Generator Advanced Properties.	585
Reset.	585
Maintain Row Order.	586
Sequence Data Object.	586
Creating a Sequence Data Object.	586
Creating a Sequence Generator Transformation.	589
Frequently Asked Questions.	590
Sequence Generator Transformation in a Non-native Environment.	591
Sequence Generator Transformation on the Blaze Engine.	591
Sequence Generator Transformation on the Spark Engine.	591
Chapter 42: Sorter Transformation.	592
Sorter Transformation Overview.	592
Sorter Transformations in Dynamic Mappings.	593
Developing a Sorter Transformation.	593
Sorter Transformation Ports.	593
Sort Tab.	594
Configure Sort Keys.	594
Parameterize the Sort Keys.	595
Sorter Transformation Advanced Properties.	596
Sorter Cache.	597
Optimizing the Sorter Cache.	597
Creating a Sorter Transformation.	597
Creating a Reusable Sorter Transformation.	598
Creating a Non-Reusable Sorter Transformation.	598
Sorter Transformation Example.	598
Sorter Transformation in a Non-native Environment.	600
Sorter Transformation on the Blaze Engine.	600
Sorter Transformation on the Spark Engine.	600
Sorter Transformation on the Databricks Spark Engine.	601
Chapter 43: SQL Transformation.	603
SQL Transformation Overview.	603
SQL Transformation Ports.	604
Input Ports.	604
Output Ports.	605
Pass-through Ports	606
SQLException Port.	607
Number of Rows Affected.	608
SQL Transformation Advanced Properties.	608

SQL Transformation Query.	610
Define the SQL Query.	611
Input Row to Output Row Cardinality.	612
Query Statement Processing.	613
Port Configuration.	613
Maximum Output Row Count.	614
Error Rows.	614
Continue on SQL Error.	615
Filter Optimization with the SQL Transformation.	616
Early Selection Optimization with the SQL Transformation.	616
Push-Into Optimization with the SQL Transformation.	616
SQL Transformation Example with an SQL Query.	617
Logical Data Object Mapping.	617
Salary Table.	618
Employee Table	618
SQL Transformation.	619
Output.	620
Stored Procedures.	621
SQL Transformation Ports for Stored Procedures.	622
Stored Procedure Result Sets.	623
Stored Procedure Example.	625
SQL Transformation Connection.	626
Creating a Connection Name Parameter.	626
Manually Creating an SQL Transformation.	627
Creating an SQL Transformation from a Stored Procedure.	627
Chapter 44: Standardizer Transformation.	629
Standardizer Transformation Overview.	629
Standardization Strategies.	629
Standardization Properties.	630
Configuring a Standardization Strategy.	631
Standardizer Transformation Advanced Properties.	631
Standardizer Transformation in a Non-native Environment.	632
Chapter 45: Union Transformation.	633
Union Transformation Overview.	633
Groups and Ports.	634
Union Transformation Advanced Properties.	634
Union Transformation Processing.	635
Creating a Union Transformation.	635
Creating a Reusable Union Transformation.	635
Creating a Non-Reusable Union Transformation.	635
Union Transformation in a Non-native Environment.	636

Union Transformation in a Streaming Mapping.	636
Union Transformation on the Databricks Spark Engine.	636
Chapter 46: Update Strategy Transformation.	637
Update Strategy Transformation Overview.	637
Setting the Update Strategy.	637
Update Strategy Transformations in Dynamic Mappings.	638
Flagging Rows Within a Mapping.	638
Update Strategy Expressions.	638
Update Strategy Transformation Advanced Properties.	639
Aggregator and Update Strategy Transformations.	639
Specifying Update Options for Individual Targets.	640
Update Strategy Transformation in a Non-native Environment.	640
Update Strategy Transformation on the Blaze Engine.	641
Update Strategy Transformation on the Spark Engine.	642
Update Strategy Transformation on the Databricks Spark Engine.	643
Chapter 47: Web Service Consumer Transformation.	644
Web Service Consumer Transformation Overview.	644
SOAP Messages.	645
WSDL Files.	645
Operations.	645
Web Service Security.	646
WSDL Selection.	647
Web Service Consumer Transformation Ports.	647
HTTP Header Input Ports.	648
Other Input Ports.	648
Web Service Consumer Transformation Input Mapping.	649
Rules and Guidelines to Map Input Ports to Nodes.	650
Customize View Options.	650
Mapping Input Ports to the Operation Input.	650
Web Service Consumer Transformation Output Mapping.	652
Rules and Guidelines to Map Nodes to Output Ports.	653
Mapping the SOAP Message as XML.	653
Customize View Options.	653
Mapping the Operation Output to Output Ports.	654
Web Service Consumer Transformation Advanced Properties.	655
Web Service Error Handling.	657
Message Compression.	657
Concurrency.	658
Filter Optimizations.	659
Enabling Early Selection Optimization with the Web Service Consumer Transformation.	659
Push-Into Optimization with the Web Service Consumer Transformation.	659

Creating a Web Service Consumer Transformation.	661
Web Service Consumer Transformation Example.	663
Input File.	663
Logical Data Object Model.	663
Logical Data Object Mapping.	663
Web Service Consumer Transformation.	664

Chapter 48: Parsing Web Service SOAP Messages..... 666

Parsing Web Service SOAP Message Overview.	666
Transformation User Interface.	667
Multiple-Occurring Output Configuration.	668
Normalized Relational Output.	668
Generated Keys.	668
Denormalized Relational Output.	669
Pivoted Relational Output.	670
Parsing anyType Elements.	670
Parsing Derived Types.	671
Parsing QName Elements.	672
Parsing Substitution Groups.	672
Parsing XML Constructs in SOAP Messages.	672
Choice Element.	673
List Element.	673
Union Element.	673

Chapter 49: Generating Web Service SOAP Messages..... 674

Generating Web Service SOAP Messages Overview.	674
Transformation User Interface.	675
Input Ports Area.	675
Operation Area.	676
Port and Hierarchy Level Relationships	676
Keys.	677
Map Ports.	678
Map a Port.	679
Map a Group.	680
Map Multiple Ports.	680
Pivoting Multiple-Occurring Ports	680
Map Denormalized Data.	682
Derived Types and Element Substitution.	683
Generating Derived Types.	683
Generating anyType Elements and Attributes.	684
Generating Substitution Groups.	684
Generating XML Constructs in SOAP Messages.	684
Choice Element.	685

List Element	685
Union Element.	686
Chapter 50: Weighted Average Transformation.	687
Weighted Average Transformation Overview.	687
Configuring a Weighted Average Transformation.	687
Weighted Match Scores Example.	688
Weighted Average Transformation Advanced Properties.	688
Weighted Average Transformation in a Non-native Environment.	689
Chapter 51: Window Transformation.	690
Chapter 52: Write Transformation.	691
Write Transformation Overview.	691
Write Transformation Properties	691
General Properties.	692
Data Object Properties.	693
Ports Properties.	693
Run-Time Properties.	693
Run-time Linking Properties.	694
Advanced Properties.	695
Create a Write Transformation.	697
Creating a Write Transformation from a Data Object.	697
Creating a Write Transformation from Mapping Flow.	698
Creating a Write Transformation from a Parameter.	699
Creating a Write Transformation from an Existing Transformation	699
Appendix A: Transformation Delimiters.	702
Transformation Delimiters Overview.	702
Index.	703

Preface

Use the *Informatica® Developer Transformation Guide* to learn about the configuration, guidelines, usage, and run-time behavior of Informatica transformations. Understand how to use transformations in appropriate use cases depending on the environment and run-time engine. View the support for each transformation depending on where and how you plan to run the mapping.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit <https://network.informatica.com>.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit <https://search.informatica.com>. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit <https://docs.informatica.com>.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at <https://network.informatica.com/community/informatica-network/product-availability-matrices>.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at <http://velocity.informatica.com>. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at <https://marketplace.informatica.com>.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:

<https://www.informatica.com/services-and-training/customer-success-services/contact-us.html>.

To find online support resources on the Informatica Network, visit <https://network.informatica.com> and select the eSupport option.

CHAPTER 1

Introduction to Transformations

This chapter includes the following topics:

- [Introduction to Transformations Overview, 32](#)
- [Transformations in the Native and Non-native Environments, 36](#)
- [Handling Transformation Data Types, 39](#)
- [Developing a Transformation, 41](#)
- [Multi-Group Transformations, 41](#)
- [Expressions in Transformations, 42](#)
- [Local Variables, 46](#)
- [Default Values for Ports, 49](#)
- [Tracing Levels, 55](#)
- [Reusable Transformations, 56](#)
- [Non-Reusable Transformations, 57](#)
- [Creating a Transformation, 58](#)

Introduction to Transformations Overview

A transformation is an object that generates, modifies, or passes data.

Informatica Developer 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 that the Data Integration Service performs on the data. Data passes through transformation ports that you link in a mapping or maplet.

Transformations can be active or passive. Transformations can be connected to the data flow, or they can be unconnected.

Active Transformations

An active transformation changes the number of rows that pass through a transformation. Or, it changes the row type.

For example, the Filter transformation is active, because it removes rows that do not meet the filter condition. The Update Strategy transformation is active, because it flags rows for insert, delete, update, or reject.

You cannot connect multiple active transformations or an active and a passive transformation to the same downstream transformation or transformation input group, because the Data Integration Service might not be able to concatenate the rows passed by active transformations.

For example, one branch in a mapping contains an Update Strategy transformation that flags a row for delete. Another branch contains an Update Strategy transformation that flags a row for insert. If you connect these transformations to a single transformation input group, the Data Integration Service cannot combine the delete and insert operations for the row.

Passive Transformations

A passive transformation does not change the number of rows that pass through the transformation, maintains the transaction boundary, and maintains the row type.

You can connect multiple transformations to the same downstream transformation or to the same transformation input group when all transformations in the upstream branches are passive. The transformation that originates the branch can be active or passive.

Unconnected Transformations

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.

Multi-Strategy Transformations

A strategy is a set of one or more operations that a transformation can perform on data. You can assign a different set of input ports and output ports to each strategy in a transformation. The transformation stores the strategies that you define in a single transformation object.

Use the **Dependencies** view to view the ports that each strategy uses.

You can define multiple transformation strategies in the following transformations:

- Case Converter
- Classifier
- Decision
- Key Generator
- Labeler
- Match
- Merge
- Parser
 - To use multiple strategies in a Parser transformation, configure the transformation to parse tokens.
- Standardizer

Transformation Descriptions

The Developer tool contains common and data quality transformations. Data quality transformations are available in Data Engineering Quality.

The following table describes each transformation:

Transformation	Type	Description
Address Validator	Active or Passive	Verifies and enhances the accuracy of postal address records, and adds information that helps users to select the mail recipients and to deliver the mail.
Association	Active	Creates links between the duplicate records that a Match transformation assigns to different clusters.
Aggregator	Active	Performs aggregate calculations.
Bad Record Exception	Active	Identifies records that might contain data errors, and loads the records to tables that an Analyst tool user can review and update.
Case Converter	Passive	Standardizes the case of strings.
Classifier	Passive	Writes labels that summarize the information in input port fields. Use when the fields contain significant amounts of text.
Comparison	Passive	Generates numeric scores that indicate the degree of similarity between pairs of input strings.
Consolidation	Active	Creates a consolidated record from records identified as duplicates by the Match transformation.
Data Masking	Passive	Replaces sensitive production data with realistic test data for non-production environments. This transformation can be connected or unconnected within the pipeline.
Data Processor	Active	Processes unstructured and semi-structured file formats in a mapping.
Decision	Passive	Evaluates conditions in input data and creates output based on the results of those conditions.
Duplicate Record Exception	Active	Identifies records that might contain duplicate information, and loads the records to tables that an Analyst tool user can review and update.
Expression	Passive	Calculates a value.
Filter	Active	Filters data.
Hierarchical to Relational	Active	Processes hierarchical input and transforms it into relational output.
Java	Active or Passive	Executes user logic coded in Java. The repository stores the byte code for the user logic.
Joiner	Active	Joins data from different databases or flat file systems.
Key Generator	Active	Assigns records to groups based on data values in a column that you select.

Transformation	Type	Description
Labeler	Passive	Writes labels that describe the characters or strings in an input port field.
Lookup	Active or Passive	Look up and return data from a flat file, logical data object, reference table, relational table, view, or synonym. This transformation can be connected or unconnected within the pipeline.
Macro	Passive	Enables dynamic functionality for a mapplet.
Match	Active	Generates scores that indicate the degrees of similarity between input records.
Merge	Passive	Reads the data values from multiple input columns and creates a single output column.
Normalizer	Active	Processes source rows that contain multiple-occurring data and returns a target row for each instance of the multiple-occurring data.
Output	Passive	Defines mapplet output rows.
Parser	Passive	Parses the values on an input port into separate output ports based on the types of information that the values contain.
Rank	Active	Limits records to a top or bottom range.
Read	Passive	Reads data from a source.
Relational to Hierarchical	Active	Processes relational input and transforms it into hierarchical output.
REST Web Service Consumer	Active	Connects to a REST web service as a web service client to access or transform data
Router	Active	Routes data into multiple transformations based on group conditions.
Sequence Generator	Passive	Generates a numeric sequence of values.
Sorter	Active	Sorts data based on a sort key.
SQL	Active or Passive	Executes SQL queries against a database.
Standardizer	Passive	Generates standardized versions of input strings.
Union	Active	Merges data from different databases or flat file systems.
Update Strategy	Active	Determines whether to insert, delete, update, or reject rows.
Web Service Consumer	Active	Connects to a web service as a web service client to access or transform data.

Transformation	Type	Description
Weighted Average	Passive	Reads the match scores that a Match transformation generates for the records in a data set, and calculates an average score for each pair of records. You can apply different weights to the scores that the transformation generates for each pair of records.
Write	Passive	Represents a target that the mapping writes data to.

Transformations in the Native and Non-native Environments

Mappings that run in the non-native environment can return different results than mappings that run in the native environment.

Consider the following processing differences:

- The non-native environment uses distributed processing and processes data on different nodes. Each node does not have access to the data that is being processed on other nodes. As a result, the runtime engine might not be able to determine the order in which the data originated. So, when you run a mapping in a non-native environment and you run the same mapping in the native environment, both mappings return correct results, but the results might not be identical.
- Each of the run-time engines in the non-native environment can process mapping logic differently. In the non-native environment, Informatica transformations might be fully supported, supported with restrictions, or not supported. Similarly, in the native environment, some Informatica transformations and transformation behavior might not be supported.

The following table lists transformations and support for different engines in a non-native environment:

Transformation	Supported Engines
<i>Transformations not listed in this table are supported only in the native environment.</i>	
Address Validator	- Blaze - Spark* - Databricks Spark
Aggregator	- Blaze - Spark* - Databricks Spark
Case Converter	- Blaze - Spark** - Databricks Spark
Classifier	- Blaze - Spark* - Databricks Spark

Transformation	Supported Engines
Comparison	- Blaze - Spark** - Databricks Spark
Consolidation	- Blaze - Spark** - Databricks Spark
Data Masking	- Blaze - Spark*
Data Processor	- Blaze - Spark**
Decision	- Blaze - Spark** - Databricks Spark
Expression	- Blaze - Spark* - Databricks Spark
Filter	- Blaze - Spark* - Databricks Spark
Java	- Blaze - Spark*
Joiner	- Blaze - Spark* - Databricks Spark
Key Generator	- Blaze - Spark** - Databricks Spark
Labeler	- Blaze - Spark** - Databricks Spark
Lookup	- Blaze - Spark* - Databricks Spark
Macro	- Blaze - Spark* - Databricks Spark
Match	- Blaze - Spark** - Databricks Spark
Merge	- Blaze - Spark** - Databricks Spark

Transformation	Supported Engines
Normalizer	- Blaze - Spark* - Databricks Spark
Parser	- Blaze - Spark* - Databricks Spark
Python	- Spark* - Databricks Spark
Rank	- Blaze - Spark* - Databricks Spark
Router	- Blaze - Spark* - Databricks Spark
Rule Specification	- Databricks Spark
Sequence Generator	- Blaze - Spark
Sorter	- Blaze - Spark* - Databricks Spark
Standardizer	- Blaze - Spark* - Databricks Spark
Union	- Blaze - Spark* - Databricks Spark
Update Strategy	- Blaze - Spark - Databricks Spark
Weighted Average	- Blaze - Spark** - Databricks Spark
Window	- Spark***
<p>* Supported for both batch and streaming mappings.</p> <p>** Supported with restrictions in batch mappings. Not supported in streaming mappings. For information about the Data Processor transformation support on the Spark engine, see the KB article.</p> <p>*** Supported for streaming mappings only. For more information, see the Data Engineering Streaming User Guide.</p>	

Handling Transformation Data Types

Transformations can process data type specific functions or allow pass-through of data without processing the data. The Data Integration Service processes some data types, such as Decimal, Timestamp with Timezone, and Timestamp with Local Timezone based on transformations.

Decimal Data Type

You can use the Decimal data type to read and write data to flat files and supported databases, such as Oracle, Microsoft SQL Server, IBM DB2, and ODBC.

For transformations that support precision up to 38 digits, the precision is 1 to 38 digits, and the scale is 0 to 38.

Transformations that Support Decimal Data Type

The following transformations support the Decimal data type with precision up to 38 digits and can perform calculations on the data:

- Aggregator
- Data Masking
- Expression
- Filter
- Java
- Joiner
- Lookup
- Normalizer
- Rank
- Router
- Sequence Generator
- Sorter
- Union
- Update Strategy

Transformations with Pass-Through Support for Decimal Data Type

Some transformations can only pass decimal data with precision up to 38 digits through the transformation. The transformation cannot perform any calculations on the data. When you use a Decimal data type with precision up to 38 digits for performing calculations on the transformations, the Data Integration Service processes the data type as Double.

The following transformations have pass-through support for Decimal data type with precision up to 38 digits:

- Data Processor
- Hierarchical to Relational
- Macro
- REST Web Service Consumer Transformation
- SQL

- Web Services Consumer

Transformations that do not Support Decimal Data Type

Some transformations do not support Decimal data type, such as Data Quality transformations.

The following applicable list of Data Quality transformations do not support Decimal data type with precision up to 38 digits:

- Address Validator
- Association
- Case Converter
- Classifier
- Comparison
- Consolidation
- Decision
- Key Generator
- Labeler
- Match
- Merge
- Parser
- Standardizer
- Weighted Average

For transformations that do not support the Decimal 38 data type, when a Decimal data type has precision greater than 28 digits, the Data Integration Service converts decimal values to double in high precision mode.

Timestamp with Time Zone

Timestamp with Time Zone is a variant of the Timestamp data type that includes a time zone offset or a time zone region.

The following transformations support the Timestamp with Time Zone data type:

- Aggregator
- Expression
- Filter
- Java
- Joiner
- Lookup
- Normalizer
- Rank
- Router
- Sequence Generator
- Sorter
- Union
- Update Strategy

Pass-through support implies that you can pass the data through the transformations but you cannot perform any functions on the Timestamp with Time Zone data type.

The following transformations have pass-through support for the Timestamp with Time Zone data type:

- Data Masking
- Data Processor
- Hierarchical to Relational
- Macro
- SQL

Timestamp with Local Time Zone

Timestamp with Local Time Zone is implicitly supported by transformations as the functionality is equivalent to Timestamp.

Developing a Transformation

When you build a mapping, you add transformations and configure them to handle data according to a business purpose.

Complete the following tasks to develop a transformation and incorporate it into a mapping:

1. Add a non-reusable transformation to a mapping or mapplet. Or, create a reusable transformation that you can add to multiple mappings or mapplets.
2. Configure the transformation. Each type of transformation has a unique set of options that you can configure.
3. If the transformation is reusable, add it to the mapping or mapplet.
4. Link the transformation to other objects in the mapping or mapplet.

You drag ports from upstream objects to the transformation input ports. You drag output ports from the transformation to ports on downstream objects. Some transformations use predefined ports that you can select.

Note: If you create a reusable transformation, you add the input and output ports you need before you link the transformation to other objects. You cannot add ports to the transformation instance on the mapplet or mapping canvas. To update the ports on a reusable transformation, open the transformation object from the repository project and add the ports.

Multi-Group Transformations

A transformation can have multiple 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.

All multi-group transformations are active transformations. You cannot connect multiple active transformations or an active and a passive transformation to the same downstream transformation or transformation input group.

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

When you save or validate a mapping, some mappings that contain active or blocking transformations might not be valid.

Rules and Guidelines for Multi-Group Transformations

When you connect transformations in a mapping, you must consider some rules and guidelines for connecting multi-group transformations.

Consider the following rules and guidelines for multi-group transformations:

- You can connect one group to one transformation or target.
- You can connect one or more output ports in a group to multiple transformations or targets.
- You cannot connect fields from multiple output groups in a transformation to the same input group of another transformation.
- You cannot connect fields from multiple output groups in different transformations to the same input group of another transformation unless each transformation between the source and the transformation are passive transformations.
- You cannot connect fields from multiple output groups in a transformation to the same input group of another transformation, unless the other transformation is a blocking transformation.
- You cannot connect an output field to multiple input fields in the same input group, unless the group is in a Normalizer transformation.

Expressions in Transformations

You can enter expressions in the **Expression Editor** in some transformations. Expressions modify data or test whether data matches conditions.

Create expressions that use transformation language functions. Transformation language functions are SQL-like functions that transform data.

Enter an expression in a 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 use the values from the IN_SALARY column later in the mapping. You might also use the transformation to calculate the total and average salaries. The Developer tool requires you to create a separate output port for each calculated value.

The following table lists the transformations in which you can enter 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, based on whether a row meets the specified condition. The Data Integration Service passes rows that return TRUE through this transformation. The transformation applies this value to each row that passes through it.
Joiner	Specifies an advanced condition used to join unsorted source data. For example, you can concatenate first name and last name master ports and then match them with the full name detail port.	TRUE or FALSE, based on whether the row meets the specified condition. Depending on the type of join selected, the Data Integration Service either adds the row to the result set or discards the row.
Rank	Sets the conditions for rows included in a rank. For example, you can rank the top 10 salespeople who are employed with the organization.	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, based on whether a row meets the specified group expression. The Data Integration Service passes rows that return TRUE 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, you might flag all employee rows for reject for people who no longer work for the organization.	Numeric code for update, insert, delete, or reject. The transformation applies this value to each row passed through it.

The Expression Editor

Use the **Expression Editor** to build SQL-like statements.

You can enter an expression manually or 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. The maximum number of characters you can include in an expression is 32,767.

Port Names in an Expression

You can enter transformation port names in an expression.

For connected transformations, if you use port names in an expression, the Developer tool updates that expression when you change port names in the transformation. For example, you write an expression that determines the difference between two dates, `Date_Promised` and `Date_Delivered`. If you change the `Date_Promised` port name to `Due_Date`, the Developer tool 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.

Adding an Expression to a Port

You can add an expression to an output port.

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.
Note: You cannot use an escape character in an expression. If you include an escape character in the expression, the Developer tool might display a parse error.
3. Optionally, add comments to the expression.
Use comment indicators `--` or `//`.
4. Click the Validate button to validate the expression.
5. Click **OK**.
6. If the expression is not valid, fix the validation errors and validate the expression again.
7. When the expression is valid, click **OK** to close the **Expression Editor**.

Comments in an Expression

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

To add comments within the expression, use `--` or `//` comment indicators.

Expression Validation

You need to validate an expression to run a mapping or preview mapplet output.

Use the Validate button in the **Expression Editor** to validate an expression. If you do not validate an expression, the Developer tool validates it when you close the **Expression Editor**. If the expression is invalid, the Developer tool displays a warning. You can save the invalid expression or modify it.

Test Expressions

You can test some expressions that you configure in the Expression Editor. When you test an expression, you enter sample data and then evaluate the expression.

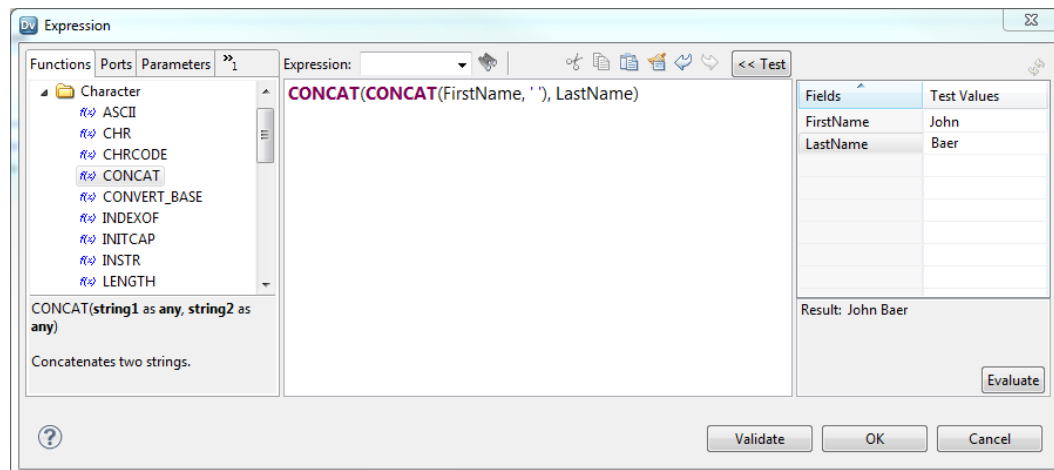
You can test expressions when you configure expressions in the following ways:

- In an output or variable port in the Expression transformation

- In the Mapping Outputs view of an Expression transformation after adding the transformation to a mapping

For example, after configuring an expression that concatenates the first name, a space, and the last name, you can enter sample data for the ports, and then evaluate the expression to verify the result.

The following image shows the results of an expression that concatenates a sample first name and last name:



Date Format Strings for Sample Data

When you test an expression that uses a port with the Date/Time or Timestamp with Time Zone data type, you must enter sample data for the port using the required date format string.

To enter sample data for a port with the Date/Time data type, use the format MM/DD/YYYY HH24:MI:SS. When you evaluate the expression, the Expression Editor displays the result using the format that you specify in the expression. If you omit the format string in the expression, the Expression Editor displays the result using the same format MM/DD/YYYY HH24:MI:SS.

To enter sample data for a port with the Timestamp with Time Zone data type, use the format MM/DD/YYYY HH24:MI:SS TZR. When you evaluate the expression, the Expression Editor displays the result using the format YYYY-MM-DD HH24:MI:SS.NS TZR.

Data Type Conversion

Multiple expression and aggregation functions might generate data of a data type that is different from the input data.

For example, when you multiply two decimal numbers with a precision of 18 digits, the resulting data type could be a decimal with a precision of 28 digits.

For input data type of Decimal with a precision of 38 digits, the result of certain operations might produce data that may not fit into resultant data type. So the user might get an overflow exception.

The following functions might require a data type conversion to accommodate an increase in the size of the data when compared to the input data types:

- avg
- cume
- divide

- median
- movingavg
- movingsum
- multiply
- Percentile
- Sum

For example, when the input data is of the Integer data type and you use multiplication operation, the resulting data type could be of the bigint data type. Similarly, when the input data type is of the Decimal data type of precision as 18 digits, the result of the multiplication operation might be large and within Decimal data type of the precision as 28 digits.

Local Variables

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

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 increase performance when you enter multiple related expressions in the same transformation. You can define components as variables instead of parsing and validating the same expression components multiple times in the transformation.

For example, if an Aggregator transformation uses the same filter condition before calculating sums and averages, you can define this condition 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 increase 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 with the same data:

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

Instead of entering the same arguments for both calculations, you might create a variable port for each condition in this calculation, and then change the expression to use the variables.

The following table shows how to use variables to simplify complex expressions and temporarily store data:

Port	Value
V_CONDITION1	JOB_STATUS = 'Full-time'
V_CONDITION2	OFFICE_ID = 1000
AVG_SALARY	AVG(SALARY, (V_CONDITION1 AND V_CONDITION2))
SUM_SALARY	SUM(SALARY, (V_CONDITION1 AND V_CONDITION2))

Store Values Across Rows

You can configure variables in transformations to store data from source rows. You can use the variables in transformation expressions.

For example, a source file contains the following rows:

```
California
California
California
Hawaii
Hawaii
New Mexico
New Mexico
New Mexico
```

Each row contains a state. You need to count the number of rows and return the row count for each state:

```
California,3
Hawaii    ,2
New Mexico,3
```

You can configure an Aggregator transformation to group the source rows by state and count the number of rows in each group. Configure a variable in the Aggregator transformation to store the row count. Define another variable to store the state name from the previous row.

The Aggregator transformation has the following ports:

Port	Port Type	Expression	Description
State	Pass-through	n/a	The name of a state. The source rows are grouped by the state name. The Aggregator transformation returns one row for each state.
State_Count	Variable	IIF (PREVIOUS_STATE = STATE, STATE_COUNT +1, 1)	The row count for the current State. When the value of the current State column is the same as the Previous_State column, the Integration Service increments State_Count. Otherwise, it resets the State_Count to 1.

Port	Port Type	Expression	Description
Previous_State	Variable	State	The value of the State column in the previous row. When the Integration Service processes a row, it moves the State value to Previous_State.
State_Counter	Output	State_Count	The number of rows the Aggregator transformation processed for a state. The Integration Service returns State_Counter once for each state.

Capture Values from Stored Procedures

Variables provide a way to capture multiple columns of return values from stored procedures.

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 the input ports first. The Integration Service evaluates the variable ports next, and the output ports last.

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. The Integration Service does not order input ports because input ports do not reference other 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. Variables can reference other variables, so 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.** The Integration Service evaluates output ports last, because output ports can reference input ports and variable ports. The display order for output ports does not matter because output ports cannot reference other output ports. Be sure output ports display at the bottom of the list of ports.

Data Type

When you configure a port as a variable, you can enter any expression or condition in it. The data type 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 data type 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.

The Integration Service uses the following guidelines to set initial values for variables:

- Zero for numeric ports
- Empty strings for string ports
- 01/01/0001 for Date/Time ports

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.

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 have 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. The default value appears 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 that the ERROR function skips in the log file.

The following errors are 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.
- **Pass-through 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.

Note: The Java Transformation converts PowerCenter® datatypes to Java datatypes, based on the Java Transformation port type. Default values for null input differ based on the Java datatype.

The following table shows the system default values for ports in connected transformations:

Port Type	Default Value	Integration Service Behavior	User-Defined Default Value Supported
Input, Pass-through	NULL	Integration Service passes all input null values as NULL.	Input, Input/Output
Output, Pass-through	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 the error message in the log file.	Output

Variable ports do not support default values. The Integration Service initializes variable ports according to the datatype.

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

User-Defined Default Values

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

Use the following rules and guidelines to override the system default values for ports:

- **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. If NULL is passed to the input port, the Integration Service replaces NULL with the default value.
- **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 log. If you define a default value in the output port, the Integration Service replaces the row with the default value when the output port has a transformation error.
- **Pass-through ports.** You can enter user-defined default values for pass-through ports if you do not want the Integration Service to treat null values as NULL. You cannot enter user-defined default values for output transformation errors in a pass-through 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 it applies the system default value.

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 mapping log or row error log.
- **ABORT.** Abort the mapping.

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'
```

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', 'MONTH DD, YYYY, HH:MI AM')
ERROR ('Null not allowed')
ABORT ('Null not allowed')
SESSSTARTTIME
```

You cannot use values from ports within the expression because the Integration Service assigns default values for the entire mapping when it initializes the mapping.

The following examples are not valid because they use values 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.

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 mapping when it encounters the ABORT function.

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.

To override null values, complete one of the following tasks:

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

The following table summarizes how the Integration Service handles null input 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.

Default Value	Default Value Type	Description
ERROR	User-Defined	Integration Service treats this as a transformation error: <ul style="list-style-type: none"> - Increases the transformation error count by 1. - Skips the row, and writes the error message to the log file or row error log. The Integration Service does not write rows to the reject file.
ABORT	User-Defined	The mapping 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.

Replace Null Values

Use a constant value or expression to substitute a specified value for null values in a port.

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.

For example, the Integration Service replaces all null values in a port with the string 'UNKNOWN DEPT.'

DEPT_NAME	REPLACED VALUE
Housewares	Housewares
NULL	UNKNOWN DEPT
Produce	Produce

Skip 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')
```

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 log file. It does not write these rows to the reject file.

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

The following 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 EXPTRANS: : 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"
)
```

Abort the Mapping

Use the ABORT function to abort a mapping when the Integration Service encounters null input values.

Default Value Validation

The Developer tool validates default values as you enter them.

The Developer tool validates default values when you save a mapping. If you enter a default value that is not valid, the Developer tool marks the mapping as not valid.

User-Defined Default Output Values

You can create user-defined default values to override the system default values for output ports.

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 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 mapping with the ABORT function.
- Write specific messages in the log for transformation errors.

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

The following table summarizes how the Integration Service handles output port transformation errors and default values in transformations:

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: <ul style="list-style-type: none">- 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 log.
ABORT	User-Defined	Mapping aborts and the Integration Service writes a message to the log. The Integration Service does not increase the error count or write rows to the reject file.

Replace 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.

Abort the Mapping

Use the ABORT function to abort a session when the Integration Service encounters null input values.

Write Messages in the Mapping Log

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

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 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 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 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 values.
- If a transformation is not connected to the mapping data flow, 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 mapping. Use the ABORT function as a default value to restrict null input values. The first null value in an input port stops the mapping.
- If an output port default value contains the ABORT function and any transformation error occurs for that port, the mapping 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 mapping.
- The ABORT function, constant values, and constant expressions override ERROR functions configured in output port expressions.

Default Value Validation

The Developer tool validates default values as you enter them.

The Developer tool validates default values when you save a mapping. If you enter a default value that is not valid, the Developer tool marks the mapping as not valid.

Tracing Levels

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

By default, the tracing level for every transformation is Normal. Change the tracing level to a Verbose setting when you need to troubleshoot a transformation that is not behaving as expected. Set the tracing level to Terse when you want the minimum amount of detail to appear in the log.

Configure the following property on the Advanced tab:

Tracing Level

Amount of detail that appears in the log for a transformation.

The following table describes the tracing levels:

Tracing Level	Description
Terse	Logs initialization information and error messages and notification of rejected data.
Normal	Logs initialization and status information, errors encountered, and skipped rows due to transformation row errors. Summarizes mapping results, but not at the level of individual rows. Default is normal.
Verbose Initialization	In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.
Verbose Data	In addition to verbose initialization tracing, logs each row that passes into the mapping. Also notes where string data was truncated to fit the precision of a column and provides detailed transformation statistics. When you configure this tracing level, row data for all rows in a block are written to the log when a transformation is processed.

Reusable Transformations

Reusable transformations are transformations that you can use in multiple mappings or mapplets.

For example, you might create an Expression transformation that calculates value-added tax for sales in Canada to 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. If you change the definition of the transformation, all instances of it inherit the changes.

The Developer tool stores each reusable transformation as metadata separate from any mapping or mapplet that uses the transformation. It stores reusable transformations in a project or folder.

When you add instances of a reusable transformation to mappings, changes you make to the transformation might invalidate the mapping or generate unexpected data.

Reusable Transformation Instances and Inherited Changes

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

When you change the transformation, instances of the transformation reflect these changes. Instead of updating the same transformation in every mapping that uses it, you can update the reusable transformation one time, and all instances of the transformation inherit the change. Instances inherit changes to ports, expressions, properties, and the name of the transformation.

Editing a Reusable Transformation

When you edit a reusable transformation, all instances of that transformation inherit the changes. Some changes might invalidate the mappings that use the reusable transformation.

You can open the transformation in the editor to edit a reusable transformation. You cannot edit an instance of the transformation in a mapping. However, you can edit the transformation runtime properties.

If you make any of the following changes to a reusable transformation, mappings that use instances of it might not be valid:

- When you delete one or more 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 that uses an incompatible datatype.
- When you change a port name, expressions that refer to the port are no longer valid.
- When you enter an expression that is not valid in the reusable transformation, mappings that use the transformation are no longer valid. The Data Integration Service cannot run mappings that are not valid.

Editor Views for a Reusable Transformation

You define properties and create Java code for a reusable Java transformation in views in the editor.

For reusable transformations, the following views are available:

Overview

Enter the name and description of the transformation, and create and configure input and output ports.

Advanced

Set advanced properties for the transformation.

Non-Reusable Transformations

A non-reusable transformation is a transformation that you create in a specific mapping. You cannot use the transformation in any other mapping.

For example, you might create a mapping that contains multiple transformations. Each transformation performs calculations on the source data. You create a non-reusable Aggregator transformation at the end of the mapping to process the results. When you create a non-reusable transformation, you can drag the ports from one transformation to another transformation in the mapping and create the input ports.

The Developer tool stores the non-reusable Aggregator transformation as metadata that it keeps with the mapping.

Editor Views for a Non-Reusable Transformation

Define properties for a non-reusable transformation in the editor views.

The following views appear for non-reusable transformations:

General

Enter the name and description of the transformation.

Ports

Create and configure input and output ports.

Advanced

Set advanced properties for the transformation.

Creating a Transformation

You can create a reusable transformation to reuse in multiple mappings or mapplets. Or, you can create a non-reusable transformation to use one time in a mapping or mapplet.

To create a reusable transformation, click **File > New > Transformation** and complete the wizard.

To create a non-reusable transformation in a mapping or mapplet, select a transformation from the Transformation palette and drag the transformation to the editor.

Certain transformations require you to choose a mode or perform additional configuration when you create the transformation. For example, when you create a Lookup transformation, you must choose a data object to use as a lookup source.

After you create a transformation, it appears in the editor. Some transformations contain predefined ports and groups. Other transformations are empty.

CHAPTER 2

Transformation Ports

This chapter includes the following topics:

- [Transformation Ports Overview, 59](#)
- [Create Ports, 59](#)
- [Configure Ports, 60](#)
- [Linking Ports, 60](#)
- [Propagating Port Attributes, 63](#)
- [Copying Ports from Excel, 66](#)

Transformation Ports Overview

Transformation ports are the individual columns of data that you connect in a mapping or maplet. Transformations receive the data from input ports and send the data out using output ports. Input/output ports receive data and pass it unchanged.

Every input object, output object, maplet, and transformation contains a collection of ports. Input objects provide data, so they contain only output ports. Output objects receive data, so they contain only input ports. Maplets contain only input ports and output ports. Transformations contain a combination of input, output, and input/output ports, depending on the transformation and its application.

A dynamic port can receive one or more columns from an upstream transformation. Dynamic ports can receive new or changed columns based on the data flow.

After you create a transformation in a mapping, create the ports and define the port properties. Complete the mapping by linking it to targets and other transformations through the ports. Propagate port attributes to pass changed attributes to a port throughout a mapping.

Create Ports

When you create some transformations, you do not have to create all of the ports manually. For example, you might create a Lookup transformation and reference a lookup table. If you view the transformation ports, you can see that the transformation has an output port for each column in the table that you referenced. You do not need to define those ports.

Create a 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 the Add button on the Ports tab.** The Designer creates an empty port you can configure.

Configure Ports

When you define the transformation ports, you define port properties. Port properties include the port name, the data type, the port type, and the default value.

Configure the following port properties:

- **Port name.** The name of the port. Use the following conventions while naming ports:
 - Begin with a single- or double-byte letter or single- or double-byte underscore (_).
 - Port names can contain any of the following single- or double-byte characters: a letter, number, underscore (_), \$, #, or @.
- **Datatype, precision, and scale.** If you plan to enter an expression or condition, verify that the datatype matches the return value of the expression.
- **Port type.** Transformations can contain a combination of input, output, input/output, and variable port types.
- **Default value.** Assign a default value for a port that contains null values or an output transformation error. 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.

Linking Ports

After you add and configure input, output, transformation, and maplet objects in a mapping, complete the mapping by linking ports between mapping objects.

The Developer tool creates the connection only when the connection meets link validation and concatenation requirements.

You can leave ports unconnected. The Data Integration Service ignores unconnected ports.

When you link ports between input objects, transformations, maplets, and output objects, you can create the following types of link:

- One to one
- One to many

You can manually link ports or link ports automatically.

One to One Links

Link one port in an input object or transformation to one port in an output object or transformation.

One to Many Links

When you want to use the same data for different purposes, you can link the port providing that data to multiple ports in the mapping.

You can create a one to many link in the following ways:

- Link one port to multiple transformations or output objects.
- Link multiple ports in one transformation to multiple transformations or output objects.

For example, you want to use salary information to calculate the average salary in a bank branch through the Aggregator transformation. You can use the same information in an Expression transformation configured to calculate the monthly pay of each employee.

Manually Linking Ports

You can manually link one port or multiple ports.

Drag a port from an input object or transformation to the port of an output object or transformation.

Use the Ctrl or Shift key to select multiple ports to link to another transformation or output object. The Developer tool links the ports, beginning with the top pair. It links all ports that meet the validation requirements.

When you drag a port into an empty port, the Developer tool copies the port and creates a link.

Automatically Linking Ports

When you link ports automatically, you can link by position or by name.

When you link ports automatically by name, you can specify a prefix or suffix by which to link the ports. Use prefixes or suffixes to indicate where ports occur in a mapping.

Linking Ports by Name

When you link ports by name, the Developer tool adds links between input and output ports that have the same name. Link by name when you use the same port names across transformations.

You can link ports based on prefixes and suffixes that you define. Use prefixes or suffixes to indicate where ports occur in a mapping. Link by name and prefix or suffix when you use prefixes or suffixes in port names to distinguish where they occur in the mapping or maplet.

Linking by name is not case sensitive.

1. Click **Mapping > Auto Link**.
The **Auto Link** dialog box appears.
2. Select an object in the **From** window to link from.
3. Select an object in the **To** window to link to.
4. Select **Name**.
5. Optionally, click **Show Advanced** to link ports based on prefixes or suffixes.

6. Click **OK**.

Linking Ports by Position

When you link by position, the Developer tool links each output port to the corresponding input port. For example, the first output port is linked to the first input port, the second output port to the second input port. Link by position when you create transformations with related ports in the same order.

1. Click **Mapping > Auto Link**.

The **Auto Link** dialog box appears.

2. Select an object in the **From** window to link from.
3. Select an object in the **To** window to link to.
4. Select **Position** and click **OK**.

The Developer tool links each output port to the corresponding input port. For example, the first output port is linked to the first input port, the second output port to the second input port.

Rules and Guidelines for Linking Ports

Certain rules and guidelines apply when you link ports.

Consider the following rules and guidelines when you connect mapping objects:

- If the Developer tool detects an error when you try to link ports between two mapping objects, it displays a symbol indicating that you cannot link the ports.
- Follow the logic of data flow in the mapping. You can link the following types of port:
 - The receiving port must be an input or input/output port.
 - The originating port must be an output or input/output port.
 - You cannot link input ports to input ports or output ports to output ports.
- You must link at least one port of an input group to an upstream transformation.
- You must link at least one port of an output group to a downstream transformation.
- You can link ports from one active transformation or one output group of an active transformation to an input group of another transformation.
- You cannot connect an active transformation and a passive transformation to the same downstream transformation or transformation input group.
- You cannot connect more than one active transformation to the same downstream transformation or transformation input group.
- You can connect any number of passive transformations to the same downstream transformation, transformation input group, or target.
- You can link ports from two output groups in the same transformation to one Joiner transformation configured for sorted data if the data from both output groups is sorted.
- You can only link ports with compatible datatypes. The Developer tool verifies that it can map between the two datatypes before linking them. The Data Integration Service cannot transform data between ports with incompatible datatypes.
- The Developer tool marks some mappings as not valid if the mapping violates data flow validation.

Propagating Port Attributes

Propagate port attributes to pass changed attributes to a port throughout a mapping.

1. In the editor, select a port in a transformation.
2. Click **Mapping > Propagate Attributes**.
The **Propagate Attributes** dialog box appears.
3. Select a direction to propagate attributes.
4. Select the attributes you want to propagate.
5. Optionally, preview the results.
6. Click **Apply**.

The Developer tool propagates the port attributes.

Dependency Types

When you propagate port attributes, the Developer tool updates dependencies.

The Developer tool can update the following dependencies:

- Link path dependencies
- Implicit dependencies

Link Path Dependencies

A link path dependency is a dependency between a propagated port and the ports in its link path.

When you propagate dependencies in a link path, the Developer tool updates all the input and input/output ports in its forward link path and all the output and input/output ports in its backward link path. The Developer tool performs the following updates:

- Updates the port name, datatype, precision, scale, and description for all ports in the link path of the propagated port.
- Updates all expressions or conditions that reference the propagated port with the changed port name.
- Updates the associated port property in a dynamic Lookup transformation if the associated port name changes.

Implicit Dependencies

An implicit dependency is a dependency within a transformation between two ports based on an expression or condition.

You can propagate datatype, precision, scale, and description to ports with implicit dependencies. You can also parse conditions and expressions to identify the implicit dependencies of the propagated port. All ports with implicit dependencies are output or input/output ports.

When you include conditions, the Developer tool updates the following dependencies:

- Link path dependencies
- Output ports used in the same lookup condition as the propagated port
- Associated ports in dynamic Lookup transformations that are associated with the propagated port
- Master ports used in the same join condition as the detail port

When you include expressions, the Developer tool updates the following dependencies:

- Link path dependencies
- Output ports containing an expression that uses the propagated port

The Developer tool does not propagate to implicit dependencies within the same transformation. You must propagate the changed attributes from another transformation. For example, when you change the datatype of a port that is used in a lookup condition and propagate that change from the Lookup transformation, the Developer tool does not propagate the change to the other port dependent on the condition in the same Lookup transformation.

Propagated Port Attributes by Transformation

The Developer tool propagates dependencies and attributes for each transformation.

The following table describes the dependencies and attributes the Developer tool propagates for each transformation:

Transformation	Dependency	Propagated Attributes
Address Validator	None.	None. This transform has predefined port names and datatypes.
Aggregator	<ul style="list-style-type: none"> - Ports in link path - Expression - Implicit dependencies 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description - Port name - Datatype, precision, scale
Association	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Case Converter	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Classifier	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Comparison	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Consolidator	None.	None. This transform has predefined port names and datatypes.
Data Masking	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Data Processor	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Decision	<ul style="list-style-type: none"> - Downstream ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Expression	<ul style="list-style-type: none"> - Ports in link path - Expression - Implicit dependencies 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description - Port name - Datatype, precision, scale

Transformation	Dependency	Propagated Attributes
Filter	<ul style="list-style-type: none"> - Ports in link path - Condition 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description - Port name
Hierarchical to Relational	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Joiner	<ul style="list-style-type: none"> - Ports in link path - Condition - Implicit Dependencies 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description - Port name - Datatype, precision, scale
Key Generator	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Labeler	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Lookup	<ul style="list-style-type: none"> - Ports in link path - Condition - Associated ports (dynamic lookup) - Implicit Dependencies 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description - Port name - Port name - Datatype, precision, scale
Match	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Merge	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Normalizer	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name
Parser	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Rank	<ul style="list-style-type: none"> - Ports in link path - Expression - Implicit dependencies 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description - Port name - Datatype, precision, scale
Read		
REST Web Service Consumer	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Router	<ul style="list-style-type: none"> - Ports in link path - Condition 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description - Port name
Sequence Generator	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description
Sorter	<ul style="list-style-type: none"> - Ports in link path 	<ul style="list-style-type: none"> - Port name, datatype, precision, scale, description

Transformation	Dependency	Propagated Attributes
SQL	- Ports in link path	- Port name, datatype, precision, scale, description
Standardizer	- Ports in link path	- Port name, datatype, precision, scale, description
Union	- Ports in link path - Implicit dependencies	- Port name, datatype, precision, scale, description - Datatype, precision, scale
Update Strategy	- Ports in link path - Expression - Implicit dependencies	- Port name, datatype, precision, scale, description - Port name - Datatype, precision, scale
Weighted Average	- Ports in link path	- Port name, datatype, precision, scale, description
Write		

Copying Ports from Excel

You can configure ports and port properties in Excel and copy them into transformation ports in the Developer tool. Port properties include the column name, data type, precision, and scale. You might want to do this when you need to develop or edit a transformation with many ports.

You can copy metadata to the following transformation types:

- Aggregator
- Expression
- Filter
- Java
- Joiner
- Lookup
- Normalizer
- Rank
- Read
- Router
- Sequence
- Sorter
- SQL
- Union
- Update Strategy
- Web Service Consumer

- Window
- Write

Editing Transformations in Excel

When you need to edit a large portion of a transformation, you do not need to change every value in the Developer tool. Instead, you can copy the transformation ports to Excel, change all the values simultaneously using Auto Fill, and then **Paste (Replace)** the transformation ports back into the Developer tool.

1. In the mapping editor of the Developer tool, select the transformation that you want to copy ports from.
2. To copy the original transformation from the Developer tool, right-click within Ports and click **Select All**.
3. Copy the ports to an Excel spreadsheet.
4. Make changes within the Excel spreadsheet. If you are changing large portions of metadata, you can use the Auto Fill feature in Excel. This allows you to fill in data based on adjacent cells by dragging the fill handle. See [“Example: Editing a Transformation in Excel” on page 68](#) for more information.
5. Copy the metadata from Excel.
6. To update the transformation with the changes, right-click within Ports and click **Paste (Replace)**.

Copying Metadata to the Developer Tool

You can create transformation ports in Excel and then copy them to the Developer tool.

1. Create a mapping in the Developer tool with the required transformations.
2. Define metadata for a transformation in Excel.
3. Copy the metadata from Excel.
4. To move the metadata to the transformation in the Developer tool, right-click within Ports and click **Paste (Replace)**.

The following image shows a sample Excel table and the resulting transformation after you copy the metadata to the Developer tool:

	A	B	C	D
1	Name	Type	Precision	Scale
2	EMPNO	decimal	10	0
3	ENAME	string	6	0
4	JOB	string	9	0
5	MGR	decimal	4	0
6	HIREDATE	string	19	0
7	SAL	decimal	7	0
8	COMM	string	7	0
9	DEPTNO	decimal	2	0

Sample Excel table

	Name	Type	Precision	Scale	Input	Output	Variable	Expression
1	EMPNO	decimal	10	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	EMPNO
2	ENAME	string	6	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	ENAME
3	JOB	string	9	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	JOB
4	MGR	decimal	4	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	MGR
5	HIREDATE	string	19	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	HIREDATE
6	SAL	decimal	7	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	SAL
7	COMM	string	7	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	COMM
8	DEPTNO	decimal	2	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	DEPTNO

After metadata is copied to the Developer tool

Note: You must confirm that the values in each cell are valid before copying the values to a transformation. For example, a string type cannot have a scale value other than "0." Precision values cannot be words and type values cannot be numbers. If the metadata is incorrect, an error message appears.

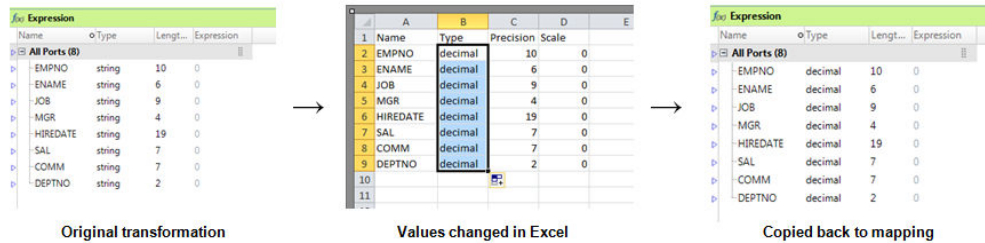
Example: Editing a Transformation in Excel

You are developing a transformation and you need to change all the string data types to decimal. Instead of changing each field individually, you make global changes in Excel and copy them to the Developer tool.

1. You right-click within Ports, click **Select All**, and paste the metadata into Excel.
2. You change the first data type value from "string" to "decimal" and then use the fill handle to automatically change the remaining cells in the column.
3. To update the transformation with the changes, you copy the metadata from Excel, right-click within Ports, and click **Paste (Replace)**.

By using Excel, you avoid having to change each field individually.

The following image shows the process of moving a transformation to Excel, using Auto Fill to change certain values, and then copying the transformation back to the Developer tool:



Rules and Guidelines for Copying from Excel

Consider the following rules and guidelines when you copy metadata from Excel to the Developer tool:

- You must confirm that the values in each cell are valid before copying the values to a transformation. For example, a string type cannot have a scale value other than "0." Precision values cannot be words and type values cannot be numbers. If the metadata is incorrect, an error message appears.
- The location that you can copy metadata to depends on the type of transformation that you are updating. For example, the Paste (Replace) option might be unavailable when you right-click in the Properties view of a transformation. However, the option will still be available when you right-click directly within the transformation ports in the editor.

CHAPTER 3

Transformation Caches

This chapter includes the following topics:

- [Transformation Caches Overview, 69](#)
- [Cache Types, 70](#)
- [Cache Files, 70](#)
- [Cache Size, 71](#)
- [Cache Size Increase by the Data Integration Service, 73](#)
- [Cache Size for Partitioned Caches, 73](#)
- [Cache Size Optimization, 74](#)

Transformation Caches Overview

The Data Integration Service allocates cache memory for Aggregator, Joiner, Lookup, Rank, and Sorter transformations in a mapping. The Data Integration Service creates index and data caches for the Aggregator, Joiner, Lookup, and Rank transformations. The Data Integration Service creates one cache for the Sorter transformation.

You can configure the cache sizes for these transformations. The cache size determines how much memory the Data Integration Service allocates for each transformation cache at the start of a mapping run.

If the cache size is larger than the available memory on the machine, the Data Integration Service cannot allocate enough memory and the mapping run fails.

If the cache size is smaller than the amount of memory required to run the transformation, the Data Integration Service processes some of the transformation in memory and stores overflow data in cache files. When the Data Integration Service pages cache files to the disk, processing time increases. For optimal performance, configure the cache size so that the Data Integration Service can process the complete transformation in memory.

By default, the Data Integration Service automatically calculates the memory requirements at run time, based on the maximum amount of memory that the service can allocate. After you run a mapping in auto cache mode, you can tune the cache sizes for the transformations. You analyze the transformation statistics in the mapping log to determine the cache sizes required for optimal performance, and then configure specific cache sizes for the transformations.

Cache Types

Aggregator, Joiner, Lookup, and Rank transformations require an index cache and a data cache. The Data Integration Service stores key values in the index cache and output values in the data cache. Sorter transformations require a single cache. The Data Integration Service stores sort keys and the data to be sorted in the Sorter cache.

The following table describes the type of information that the Data Integration Service stores in each cache:

Mapping Object	Cache Types and Descriptions
Aggregator	<ul style="list-style-type: none">- Index. Stores group values as configured in the group by ports.- Data. Stores calculations based on the group by ports.
Joiner	<ul style="list-style-type: none">- Index. Stores all master rows in the join condition that have unique keys.- Data. Stores master source rows.
Lookup	<ul style="list-style-type: none">- Index. Stores lookup condition information.- Data. Stores lookup data that is not stored in the index cache.
Rank	<ul style="list-style-type: none">- Index. Stores group values as configured in the group by ports.- Data. Stores ranking information based on the group by ports.
Sorter	<ul style="list-style-type: none">- Sorter. Stores sort keys and data.

Cache Files

When you run a mapping, the Data Integration Service creates at least one cache file for each Aggregator, Joiner, Lookup, Rank, and Sorter transformation. If the Data Integration Service cannot run a transformation in memory, it writes the overflow data to the cache files.

The following table describes the types of cache files that the Data Integration Service creates for different mapping objects:

Mapping Object	Cache File
Aggregator, Joiner, Lookup, and Rank transformations	The Data Integration Service creates the following types of cache files: <ul style="list-style-type: none">- One header file for each index cache and data cache- One data file for each index cache and data cache
Sorter transformation	The Data Integration Service creates one sorter cache file.

When you run a mapping, the Data Integration Service writes a message in the mapping log indicating the cache file name and the transformation name. When a mapping completes, the Data Integration Service releases cache memory and usually deletes the cache files. You might find index and data cache files in the cache directory under the following circumstances:

- You configure the Lookup transformation to use a persistent cache.
- The mapping does not complete successfully. The next time you run the mapping, the Data Integration Service deletes the existing cache files and creates new ones.

Because writing to cache files can slow mapping performance, configure the cache sizes to run the transformation in memory.

Cache File Directory

For Aggregator, Joiner, Lookup, and Rank transformations, the Data Integration Service creates the cache files in the directory specified in the Cache Directory property. For Sorter transformations, the service creates the cache files in the directory specified in the Work Directory property.

If the Data Integration Service process does not find the directory, it fails the mapping and writes a message to the mapping log indicating that it could not create or open the cache file. The Data Integration Service may create multiple cache files. The number of cache files is limited by the amount of disk space available in the cache directory.

Cache Size

Cache size determines how much memory the Data Integration Service allocates for each transformation cache at the start of a mapping run. You can configure a transformation cache size to use auto cache mode or to use a specific value.

Auto Cache Size

By default, a transformation cache size is set to Auto. The Data Integration Service automatically calculates the cache memory requirements at run time. You define the maximum amount of memory that the service can allocate.

The Data Integration Service uses the following guidelines to automatically allocate the memory:

Allocates more memory to transformations with higher processing times.

The Data Integration Service allocates more memory to transformations that typically have higher processing times. For example, the Data Integration Service allocates more memory to the Sorter transformation because the Sorter transformation typically takes longer to run.

Allocates more memory to the data cache than to the index cache.

Aggregator, Joiner, Lookup, and Rank transformations require an index cache and a data cache. When the Data Integration Service divides the memory allocated for the transformation across the index and data cache, it allocates more memory to the data cache.

Sorter transformations require a single cache. The service allocates all of the memory allocated for the transformation to the Sorter cache.

Maximum Memory for Auto Cache Size

You define the maximum amount of memory that the Data Integration Service can allocate to transformation caches in the Maximum Memory Per Request property for Data Integration Service modules in the Administrator tool.

Each module runs different types of requests that have different memory requirements. For example, mapping and profile requests typically require more cache memory than SQL service or web service requests.

You can configure the Maximum Memory Per Request property for the following Data Integration Service modules:

- Mapping Service Module
- Profiling Service Module
- SQL Service Module
- Web Service Module

Note: Mapping Service Module requests include mappings and mappings run from Mapping tasks within a workflow.

For the Profiling Service Module, Maximum Memory Per Request defines the maximum amount of memory that the Data Integration Service can allocate for each mapping run for a single profile request.

For the remaining modules, the behavior of Maximum Memory Per Request depends on the Data Integration Service configuration. The behavior depends on the Launch Job Options property and the Maximum Memory Size property on the Data Integration Service.

The following table describes the behavior of Maximum Memory Per Request for the mapping, SQL service, and web service modules based on the Data Integration Service configuration:

Data Integration Service Configuration	Maximum Memory Per Request Behavior
Runs jobs in separate local or remote system processes, or Maximum Memory Size is 0 (default)	<p>Maximum amount of memory, in bytes, that the Data Integration Service can allocate for all transformations that use auto cache mode in a single request.</p> <p>The value that you define for Maximum Memory Per Request affects only transformations that use auto cache mode. The Data Integration Service allocates memory separately to transformations for which you configure a specific cache size. The total memory used by the request can exceed the value of Maximum Memory Per Request.</p> <p>For example, Maximum Memory Per Request is set to 800 MB. A mapping has three transformations that require caching. You configure two transformations to use auto cache mode and configure the third transformation to use a total of 500 MB for the cache sizes. The Data Integration Service allocates a total of 1,300 MB of memory for all of the transformation caches.</p>
Runs jobs in the Data Integration Service process, and Maximum Memory Size is greater than 0	<p>Maximum amount of memory, in bytes, that the Data Integration Service can allocate for a single request.</p> <p>The value that you define for the Maximum Memory Per Request property affects all transformations. The total memory used by the request cannot exceed the value of Maximum Memory Per Request.</p>

When you increase the maximum amount of memory used for auto cache mode, you increase the maximum cache size that can be used for all requests to the module. You can increase the maximum amount of memory to ensure that no cache files are paged to the disk. However, because this value is used for all requests, the Data Integration Service might allocate more memory than is needed for some requests.

Specific Cache Size

You can configure a specific cache size for a transformation. The Data Integration Service allocates the specified amount of memory to the transformation cache at the start of the mapping run. Configure a specific value in bytes when you tune the cache size.

The first time that you configure a cache size, use auto cache mode. After you run the mapping, analyze transformation statistics in the mapping log to determine the cache sizes required to run the transformations in memory. When you configure the cache size to use the value specified in the mapping log, you can ensure

that no allocated memory is wasted. However, the optimal cache size varies based on the size of the source data. Review the mapping logs after subsequent mapping runs to monitor changes to the cache size. If you configure a specific cache size for a reusable transformation, verify that the cache size is optimal for each use of the transformation in a mapping.

To define specific cache sizes, configure the cache size values in the transformation properties in the Developer tool.

Cache Size Increase by the Data Integration Service

The Data Integration Service creates each memory cache based on the configured cache size. In some situations, the Data Integration Service might increase the configured cache size because it requires more cache memory.

The Data Integration Service might increase the configured cache size for one of the following reasons:

Configured cache size is less than the minimum cache size required to process the operation.

The Data Integration Service requires a minimum amount of memory to initialize each mapping. If the configured cache size is less than the minimum required cache size, then the Data Integration Service increases the configured cache size to meet the minimum requirement. If the Data Integration Service cannot allocate the minimum required memory, the mapping fails.

Configured cache size is not a multiple of the cache page size.

The Data Integration Service stores cached data in cache pages. The cached pages must fit evenly into the cache. For example, if you configure 10 MB (1,048,576 bytes) for the cache size and the cache page size is 10,000 bytes, then the Data Integration Service increases the configured cache size to 1,050,000 bytes to make it a multiple of the 10,000-byte page size.

When the Data Integration Service increases the configured cache size, it continues to run the mapping and writes the following messages in the mapping log:

```
INFO: MAPPING,    TE_7212,    Increasing [Index Cache] size for transformation
<transformation name> from <configured cache size> to <new cache size>.
INFO: MAPPING,    TE_7212,    Increasing [Data Cache] size for transformation
<transformation name> from <configured cache size> to <new cache size>.
```

Cache Size for Partitioned Caches

If you have the Partitioning option, cache partitioning creates a separate cache for each partition that runs an Aggregator, Joiner, Rank, Lookup, or Sorter transformation. During cache partitioning, each partition stores different data in a separate cache. When the Data Integration Service uses cache partitioning for these transformations, the service divides the allocated cache size across the partitions.

For example, you configure the transformation cache size to be 100 MB. The Data Integration Service uses four partitions to run the transformation. The service divides the cache size value so that each partition uses a maximum of 25 MB for the cache size.

Cache Size Optimization

For optimal mapping performance, configure the cache sizes so that the Data Integration Service can run the complete transformation in memory.

To configure optimal cache sizes, perform the following tasks:

1. Set the tracing level to verbose initialization.
2. Run the mapping in auto cache mode.
3. Analyze caching performance in the mapping log.
4. Configure specific values for the cache sizes.

Step 1. Set the Tracing Level to Verbose Initialization

In the Developer tool, set the tracing level to verbose initialization to enable the Data Integration Service to write transformation statistics to the mapping log. The transformation statistics list the cache sizes required for optimal performance. By default, the tracing level is set to normal.

Set the tracing level to verbose initialization in one of the following ways:

- Modify the advanced properties for each transformation that uses a cache.
- Modify the default mapping configuration properties if you plan to run the mapping for the first time from the Developer tool. For more information, see the *Informatica Developer Tool Guide*.
- Modify the advanced properties for an application that contains the mapping if you plan to run the deployed mapping for the first time from the command line. For more information, see the *Informatica Developer Tool Guide*.

Step 2. Run the Mapping in Auto Cache Mode

The first time that you run the mapping, use auto cache mode for the transformation cache sizes.

You can run the mapping from the Developer tool. Or, you can add the mapping to an application and then deploy the application to the Data Integration Service so that you can run the mapping from the command line.

Step 3. Analyze Caching Performance

After you run the mapping in auto cache mode, analyze the transformation statistics in the mapping log to determine the cache sizes required for optimal mapping performance.

When an Aggregator, Joiner, Lookup, or Rank transformation pages to the disk, the mapping log specifies the index and data cache sizes required to run the transformation in memory. For example, you run an Aggregator transformation called AGG_TRANS. The mapping log contains the following text:

```
CMN 1791, The index cache size that would hold [1098] aggregate groups of input rows for
[AGG_TRANS], in memory, is [286720] bytes
CMN 1790, The data cache size that would hold [1098] aggregate groups of input rows for
[AGG_TRANS], in memory, is [1774368] bytes
```

The log shows that the index cache requires 286,720 bytes and the data cache requires 1,774,368 bytes to run the transformation in memory without paging to the disk.

When a Sorter transformation pages to the disk, the mapping log states that the Data Integration Service made multiple passes on the source data. The Data Integration Service makes multiple passes on the data when it has to page to the disk to complete the sort. The message specifies the number of bytes required for

a single pass, which is when the Data Integration Service reads the data once and performs the sort in memory without paging to the disk.

For example, you run a Sorter transformation called SRT_TRANS. The mapping log contains the following text:

```
SORT_40427, Sorter Transformation [SRT_TRANS] required 2-pass sort (1-pass temp I/O: 13126221824 bytes). You may try to set the cache size to 14128 MB or higher for 1-pass in-memory sort.
```

The log shows that the Sorter cache requires 14,128 MB so that the Data Integration Service makes one pass on the data.

Step 4. Configure Specific Cache Sizes

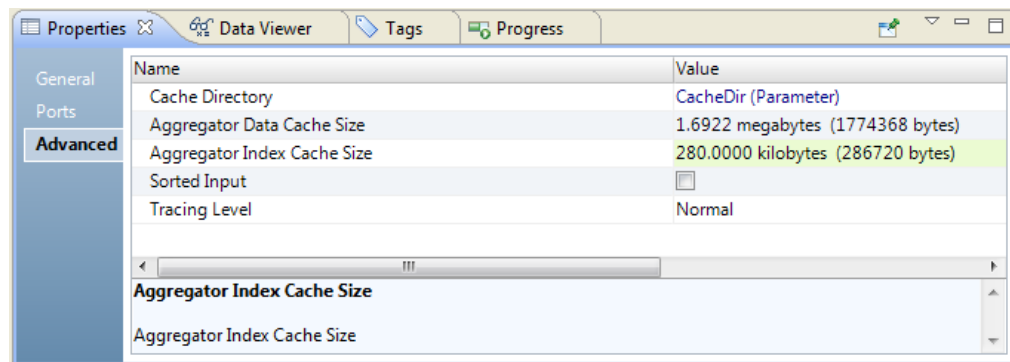
For optimal performance, configure the transformation cache sizes to use the values specified in the mapping log. Update the index and data cache size transformation properties in the Developer tool.

1. In the Developer tool, open the reusable or non-reusable transformation.
2. Locate the cache size properties depending on the following transformation types:

Option	Description
Reusable Aggregator, Joiner, Rank, or Sorter transformation	Click the Advanced view.
Non-reusable Aggregator, Joiner, Rank, or Sorter transformation	Click the Advanced tab in the Properties view.
Reusable Lookup transformation	Click the Run-time view.
Non-reusable Lookup transformation	Click the Run-time tab in the Properties view.

3. Enter the values in bytes that the mapping log recommended for the index and data cache sizes.

The following image shows a non-reusable Aggregator transformation that has specific values configured for the index and data cache sizes:



4. Click **File > Save**.

CHAPTER 4

Address Validator Transformation

This chapter includes the following topics:

- [Address Validator Transformation Overview, 76](#)
- [Address Reference Data, 77](#)
- [Modes and Templates, 79](#)
- [Port Groups and Port Selection, 79](#)
- [Address Validator Transformation Input Port Groups, 80](#)
- [Address Validator Transformation Output Port Groups, 80](#)
- [Multiple-Instance Ports, 83](#)
- [Address Validation Projects, 84](#)
- [Formatted Addresses and Mail Carrier Standards, 85](#)
- [Partial Address Completion , 86](#)
- [Address Validator Status Ports, 87](#)
- [Address Validator Transformation General Settings, 98](#)
- [Address Validation Properties in the Preferences Window, 99](#)
- [Address Validation Advanced Properties, 102](#)
- [Certification Reports, 121](#)
- [Configuring an Address Validator Transformation, 124](#)
- [Adding Ports to the Address Validator Transformation, 125](#)
- [Creating User-Defined Templates, 125](#)
- [Defining Address Validator Models, 126](#)
- [Defining a Certification Report, 126](#)
- [Address Validator Transformation in a Non-native Environment, 127](#)

Address Validator Transformation Overview

The Address Validator transformation is a multiple-group transformation that compares input address data with address reference data. It determines the accuracy of the addresses and fixes errors in the addresses.

The transformation creates records that meet mail delivery standards for data content and structure. The transformation also adds status information to each address.

The Address Validator transformation performs the following operations on address data:

- The transformation compares address records in the source data to address definitions in the address reference data.
- It generates detailed status reports on the validity of each input address, its deliverable status, and the nature of any errors or ambiguities it contains.
- It fixes errors and completes partial address records. To fix an address, the transformation must find a positive match with an address in the reference data. The transformation copies the required data elements from the address reference data to the address records.
- It adds information that does not appear in the standard address but that helps postal delivery, such as delivery point information and geocoding information.
- It writes output addresses in the format that the data project and the mail carrier requires. You define the format when you select the output ports on the transformation.

Address Reference Data

An address reference data set describes the addresses that a national mail carrier recognizes in a country. Before you perform address validation with the Address Validator transformation, install the address reference data on the Informatica services machine in the domain. You buy and download the address reference data from Informatica.

Install an address reference data file for each country that the source address data identifies. A country that has a large population might require multiple files. In addition, you can install data files that supplement or enrich the address data. The mail carrier can use the data enrichments to certify the accuracy of the addresses and to accelerate mail delivery.

When you perform address validation, the Address Validator transformation compares each input record to the address reference data. If the transformation finds the input address in the address reference data, the transformation can update the record with the correct and complete address data. If you purchased additional reference data sets, the transformation can also enrich the address data.

Use the **Preferences** window in the Developer tool to view information about the address reference data files on the Informatica services machine in the domain.

Types of Address Reference Data

The validation mode that you select determines how the transformation compares the input address to the address reference data.

The Address Validator transformation can read the following types of address reference data:

Address code lookup data

Install address code lookup data to retrieve a partial address or full address from a code value on an input port. The completeness of the address depends on the level of address code support in the country to which the address belongs. To read the address code from an input address, select the country-specific ports in the Discrete port group.

You can select ports for the following countries:

- Austria. Returns an address to building level.
- Germany. Returns an address to locality, municipality, or street level.
- Japan. Returns an address to the unique mailbox level.
- South Africa. Returns an address to street level.
- South Korea. Returns an address to the unique mailbox level.
- Serbia. Returns an address to street level.
- United Kingdom. Returns an address to the unique mailbox level.

The Address Validator transformation reads address code lookup data when you configure the transformation to run in address code lookup mode.

Batch and interactive data

Install batch and interactive data to perform address validation on a set of address records. Use batch and interactive data to verify that the input addresses are fully deliverable and complete based on the current postal data from the national mail carrier.

When you configure the transformation to run in batch mode, the Address Validator transformation returns a single address for each input address. When you configure the transformation to run in interactive mode, the Address Validator transformation returns one or more addresses for each input address.

CAMEO data

Install CAMEO data to add customer segmentation data to residential address records. Customer segmentation data indicates the likely income level and lifestyle preferences of the residents at each address.

The Address Validator transformation reads CAMEO data when you configure the transformation to run in batch mode or certified mode.

Certified data

Install certified data to verify that address records meet the certification standards that a mail carrier defines. An address meets a certification standard if contains data elements that can identify a unique mailbox, such as delivery point data elements. When an address meets a certification standard, the mail carrier charges a reduced delivery rate.

The following countries define certification standards:

- Australia. Certifies mail according to the Address Matching Approval System (AMAS) standard.
- Canada. Certifies mail according to the Software Evaluation And Recognition Program (SERP) standard.
- France. Certifies mail according to the National Address Management Service (SNA) standard.
- New Zealand. Certifies mail according to the SendRight standard.
- United States. Certifies mail according to the Coding Accuracy Support System (CASS) standard.

The Address Validator transformation reads certified data when you configure the transformation to run in certified mode.

Geocode data

Install geocode data to add geocodes to address records. Geocodes are latitude and longitude coordinates.

The Address Validator transformation reads geocode data when you configure the transformation to run in batch mode or certified mode.

Suggestion list data

Install suggestion list data to find alternative valid versions of a partial address record. Use suggestion list data when you configure an address validation mapping to process address records one by one in real time. The Address Validator transformation uses the data elements in the partial address to perform a duplicate check on the suggestion list data. The transformation returns any valid address that includes the information in the partial address.

The Address Validator transformation reads suggestion list data when you configure the transformation to run in suggestion list mode.

Supplementary data

Install supplementary data to add data to an address record that can assist the mail carrier in mail delivery. Use the supplementary data to add detail about the geographical or postal area that contains the address. In some countries, supplementary data can provide a unique identifier for a mailbox within the postal system.

The Address Validator transformation reads supplementary data when you configure the transformation to run in batch mode or certified mode.

Note: The transformation does not read address reference data in country recognition mode or in parse mode.

RELATED TOPICS:

- [“Address Validator Transformation General Settings” on page 98](#)

Modes and Templates

When you configure the Address Validator transformation, you can select the type of validation that the transformation performs. The transformation defines the validation type as the mode. Select the mode on the **General Settings** tab or as an advanced property of the transformation.

You can create a port template in the transformation. A template is a subset of ports from one or more port groups. Use a template to organize the ports that you expect to use in a project.

Port Groups and Port Selection

The Address Validator transformation contains predefined port groups that contain the input and output ports that you can use. When you configure an Address Validator transformation, you browse the groups and select the ports that you need.

Select input ports that correspond to the structure of the address input data. Select output ports that contain the address data that the project requires.

You can add input and output ports directly to the transformation, or you can create a default model that contains input and output ports. When you add ports directly to the transformation, the ports you select apply to that transformation only. When you add ports to the default model, the ports you select apply to future Address Validator transformations that you create.

You can also add pass-through ports to the transformation for columns that you do not want the Address Validator transformation to process.

Address Validator Transformation Input Port Groups

Before you can connect address data to input ports on the transformation, you browse the input groups and select the ports that correspond to the structure and content of the input data. Browse the output groups and select the ports that match your data requirements.

The Address Validator transformation displays the port groups in a Basic Model and Advanced Model. You can define most addresses using a port group in the Basic Model. If your addresses are highly complex, use the additional ports available in the Advanced Model.

Note: Select ports from one input port group only.

The transformation has the following input port groups:

Discrete

Use Discrete ports to read data columns that contain complete information on a single data element, such as a house number, street, or post code. Find the Discrete group in the Basic and Advanced models.

Hybrid

Use Hybrid ports to read data columns that contain information about one or more data elements. The Hybrid group combines ports from the Discrete and Multiline groups. Use Hybrid ports to create address records that you can submit to a mail carrier. Hybrid ports structure an address to mail carrier standards and identify the type of data on each line. Find the Hybrid group in the Basic and Advanced models.

Multiline

Use Multiline ports to read data columns that contain multiple data elements. Each input column corresponds to a line of an address. For best results, define the input data in the format that the mail carrier requires. Select the Multiline ports to create a printable set of address records.

Each Multiline port represents a line in the printable address, such as the following line of street data:

```
123 Main Street, Apartment 2
```

Multiline ports do not specify the type of data that appears on each address line. Find the Multiline group in the Basic and Advanced models.

Single-Line

Use Single-Line ports to read a single data column that contains all address elements to province level and that does not include a separator between elements. Use the Complete Address port in the port group to submit the address elements. The port group also includes a Country port that you can use to read country information for the address. Find the Single-Line group in the Basic and Advanced models.

Address Validator Transformation Output Port Groups

Before you can connect the Address Validator transformation to other transformations or data objects, you determine the types of information you need and the structure that the output addresses will take.

Browse the output groups and select the ports that match your data requirements.

Note: You can select ports from multiple output groups, and you can select ports that have common functionality.

The transformation has the following predefined output groups:

Address Elements

Writes street data elements such as house number, apartment number, and street name to separate parts. Find the Address Elements group in the Basic and Advanced models.

AT Supplementary

Writes data to addresses in Austria that can help postal delivery, such as building-level post code data. Find the AT Supplementary group in the Basic model.

AU Supplementary

Writes data to addresses in Australia that identifies the geographical regions to which the Australian Bureau of Statistics assigns the addresses. Find the AU Supplementary group in the Basic model.

Australia Specific

Writes data to addresses in Australia that enables the addresses to meet the Address Matching Approval System (AMAS) standards of Australia Post. Find the Australia Specific group in the Basic and Advanced models.

BE Supplementary

Writes data to addresses in Belgium that can help postal delivery. The data includes locality and neighborhood identification codes from the statistics directorate in Belgium. Find the BE Supplementary group in the Basic model.

BR Supplementary

Writes data to addresses in Brazil that can help postal delivery, such as district identification codes from the Institute of Geography and Statistics (IBGE). Find the BR Supplementary group in the Basic model.

CAMEO

Generates demographic and income summary data that you can use in customer segmentation analysis. Find the CAMEO group in the Basic model.

Canada Specific

Writes data to addresses in Canada that enables the addresses to meet the Software Evaluation and Recognition Program (SERP) standards of Canada Post. Find the Canada Specific group in the Basic model.

CH Supplementary

Writes data to addresses in Switzerland that can help postal delivery, such as extended post code data. Find the CH Supplementary group in the Basic model.

CZ Supplementary

Writes data to addresses in the Czech Republic that can help postal delivery, such as extended post code data. Find the CZ Supplementary group in the Basic model.

Contact Elements

Writes person or contact data, such as names, salutations, and job titles. Find the Contact Elements group in the Advanced model.

Country

Writes the country name or country code defined by the International Organization for Standardization (ISO). Find the Country group in the Basic and Advanced models.

DE Supplementary

Writes data to addresses in Germany that can help postal delivery, such as municipality and district code data. Find the DE Supplementary group in the Basic model.

ES Supplementary

Writes data to addresses in Spain that can help postal delivery. Find the ES Supplementary group in the Basic model.

Formatted Address Line

Writes addresses that are formatted for printing and mailing. Find the Formatted Address Line group in the Basic and Advanced models.

FR Supplementary

Writes data to addresses in France that can help postal delivery, such as identification codes from the National Institute of Statistics and Economic Studies (INSEE). Find the FR Supplementary group in the Basic model.

France Specific

Writes data to addresses in France that enables the addresses to meet the National Address Management Service (SNA) standards of La Poste. Find the France Specific group in the Basic model.

Geocoding

Generates geocode data, such as latitude and longitude coordinates, for an address. Find the Geocoding group in the Basic model.

ID Elements

Writes Record ID and Transaction Key data. Find the ID Elements group in the Advanced model.

IT Supplementary

Writes data to addresses in Italy that can help postal delivery. Find the IT Supplementary group in the Basic model.

JP Supplementary

Writes data to addresses in Japan that can help postal delivery, such as Choumei Aza codes. Find the JP Supplementary group in the Basic model.

KR Supplementary

Writes data to addresses in South Korea that can help postal delivery, such as unique identifiers that can specify current and non-current versions of a given address. Find the KR Supplementary group in the Basic model.

Last Line Elements

Writes data that can appear on the last line of a domestic address. Find the Last Line Elements group in the Basic and Advanced models.

New Zealand Specific

Writes data to addresses in New Zealand that enables the addresses to meet the SendRight standards of New Zealand Post. Find the New Zealand Specific group in the Basic model.

PL Supplementary

Writes data to addresses in Poland that can help postal delivery, such as Territorial Division (TERYT) data. Find the PL Supplementary group in the Basic model.

Residue

Writes data elements that the transformation cannot parse to other ports. Find the Residue group in the Basic and Advanced models.

RS Supplementary

Writes data to addresses in Serbia that can help postal delivery, such as post code suffix data. Find the RS Supplementary group in the Basic model.

RU Supplementary

Writes data to addresses in Russia that can help postal delivery, such as the Federal Information Addressing System identifier for the address. Find the RU Supplementary group in the Basic model.

Status Info

Generates detailed data on the quality of each input and output address. Find the Status Info group in the Basic model.

UK Supplementary

Writes data to addresses in the United Kingdom that can help postal delivery, such as delivery point data and ordnance survey data. Find the UK Supplementary group in the Basic model.

US Specific

Writes data to addresses in the United States that enables the addresses to meet the Coding Accuracy Support System (CASS) standards of the United States Postal Service. Find the US Specific group in the Basic model.

US Supplementary

Writes geographic and demographic data such as Federal Information Processing Standard (FIPS) codes for United States addresses. Find the US Supplementary group in the Basic model.

XML

Writes address record data in an XML structure that the Address Verification software library defines. Find the XML group in the Advanced model.

ZA Supplementary

Writes data to addresses in South Africa that can help postal delivery, such as National Address Database data. Find the ZA Supplementary group in the Basic model.

Multiple-Instance Ports

Many types of address data can occur more than once in an address. You can select multiple instances of a port when the address contains multiple cases of a data element.

A multiple-instance port can contain up to six instances. Many addresses use one instance of a port for each data element that they contain. Some addresses use a second instance of a port. A small set of addresses use more than one port instance.

Often, the first instance of a port is the primary name or the largest area that the port identifies. You must verify the relationship between the port instances for any port that you select.

Street Complete Ports Example

A United Kingdom address record can contain two street names when one street is part of a larger street plan.

The following table contains an address that uses two Street Complete ports:

Port	Data
Street Number Complete 1	1A
Street Complete 1	THE PHYGTLE
Street Complete 2	SOUTH STREET
Locality Name 1	NORFOLK
Postcode 1	NR25 7QE

In the example, the street data in Street Complete 1 is dependent on the street data in Street Complete 2. The data in Street Number Complete 1 refers to the data in Street Complete 1.

Note: Although Street Complete 1 specifies the location of the mailbox, Street Complete 2 might be the larger street.

Contact Ports Example

An address record can contain multiple contacts when each contact is a member of the same household.

The following table contains an address that uses two Contact Name ports:

Port	Data
Contact Name 1	MR. JOHN DOE
Contact Name 2	MS. JANE DOE
Formatted Address Line 1	2 MCGRATH PLACE EAST
Formatted Address Line 2	ST. JOHN'S NL A1B 3V4
Formatted Address Line 3	CANADA

In the example, the organization can decide on the precedence to apply to Contact Name 1 or Contact Name 2. The Address Validator transformation does not prioritize the contact data.

If you format addresses for printed output, you might use multiple instances of the Formatted Address Line ports. You can select up to 12 Formatted Address Line ports.

Address Validation Projects

You can use the Address Validator transformation in many types of project. You create an address template with different ports for each project type.

You can define an address validation project with one or more of the following objectives:

Create formatted addresses that conform to mail carrier standards

You can prepare a large address record set for a mail campaign. If you create the addresses in the format that the mail carrier prefers, your mail costs are significantly reduced. When you prepare addresses for mailing, select output ports that write each line of the formatted address to a single port. You can select a different port for the contact name, the street address lines, and the locality and post code lines.

Organize addresses by income and lifestyle indicators

You can add customer segmentation data to residential address records. Customer segmentation data indicates the likely income level and lifestyle preferences of the residents at each address. You select ports from the CAMEO output group to add customer segmentation data to address records. You can use customer segmentation data in mail campaigns that target multiple consumer markets.

Create addresses that are certified by the mail carrier

When you prepare a record set for Australia Post, Canada Post, or the United States Postal Service (USPS), you can add data that confirms the deliverability of each address.

The Address Validator transformation can generate reports to certify that the address records are complete and accurate to the data standards of each mail carrier.

Create addresses that meet regulatory requirements

You can verify that the address records held by your organization are accurate to industry or government regulations. Select output ports that write each address data element to a separate field. Also select the address validation status ports that provide detailed information on the accuracy and completeness of the output data.

Complete partial addresses

You can enter a partial address and retrieve the valid complete addresses that match the partial address in the reference data. To complete partial addresses, configure the transformation to run in suggestion list mode or interactive mode. You can enter the input address as a single line on the Complete Address port.

Improve the data quality of addresses

In parallel with other data projects, you can to improve the structure and general data quality of the address data set. For example, the data set may contain more columns that you need, or it may contain the same type of data across multiple columns. You can reduce the number of columns in the data set, and you can simplify the columns that you use for different types of data.

Formatted Addresses and Mail Carrier Standards

When you prepare address records for a mail campaign, you create a printable address structure that matches the formatting standards of the mail carrier.

For example, the USPS maintains the following address format for domestic United States addresses:

Line 1	Person/Contact Data	JOHN DOE
Line 2	Street Number, Street, Sub-Building	123 MAIN ST NW STE 12
Line 3	Locality, State, ZIP Code	ANYTOWN NY 12345

You can define a printable address format that writes each line of the address to a single port. You can use ports that recognize the types of data on each line, or you can use ports that populate the address structure regardless of the data on each line.

The following table shows different ways you can format a United States address for printing:

For This Address	Use These Ports	Or Use These Ports
JOHN DOE	Recipient Line 1	Formatted Address Line 1
123 MAIN ST NW STE 12	Delivery Address Line 1	Formatted Address Line 2
ANYTOWN NY 12345	Country Specific Last Line 1	Formatted Address Line 3

Use Formatted Address Line ports when the data set contains different types of address, such as business and residential addresses. A business address may need three address lines for contact and organization data. The Address Validator transformation ensures that each business or residential address is correctly formatted by using Formatted Address Line ports only when they are needed. However, Formatted Address Line ports do not identify the type of data that they contain.

Use Recipient Line, Delivery Address Line, and Country Specific Last Line ports when all address follow one format. The Recipient Line, Delivery Address Line, and Country Specific Last Line ports separate the address data elements by information type and make the data set easy to understand.

Note: You can select other ports to process this address. This example focuses on ports that format the addresses for printing and delivery.

Demographic and Geographic Data

When you create a record set for a mail campaign, you can add multiple types of data that may not otherwise appear in the address. Use this data to review the demographic and geographic spread of the mail items.

For example, you can identify the Congressional District that a United States address belongs to. You can also generate latitude and longitude coordinates if the destination country includes the coordinates in its mail system reference data.

Partial Address Completion

When you use suggestion list mode or interactive mode, you can enter an incomplete address and retrieve valid and complete addresses from the reference data.

Select suggestion list mode when you are uncertain of an address and you want to view a list of valid address candidates. Select interactive mode when you are confident of an address and you want to validate the complete form. In each case, the Address Validator transformation searches the address reference data and returns all addresses that contain the input data.

Consider the following rules and guidelines when you configure the transformation to run in suggestion list mode or interactive mode:

- You can define an input address on multiple ports, or you can enter all address elements on the Complete Address input port.
- When you configure the transformation in suggestion list mode, select ports from the Discrete input group. Alternatively, select a Complete Address port and optionally select a Country Name port from the Multiline group.

- Suggestion list mode and interactive mode can return multiple addresses for each input address. The Max Result Count property specifies an upper limit for the number of addresses returned. If the number of matching address is greater than the Max Result Count value, the Count Overflow port returns the number of additional addresses.
- Informatica Address Verification refers to suggestion list mode as fast completion mode.

Address Validator Status Ports

The Address Validator transformation writes status information on the address elements that it reads and writes on input and output ports. Use the Status Info ports to view the status information.

You can select the following status ports:

Address Resolution Code

Describes the non-valid address elements in the address. Select the port from the Status Info port group in the Basic Model.

Address Type

Indicates the address type in cases where the mail carrier recognizes more than one form of an address. Select the port from the Status Info port group in the Basic Model.

Element Input Status

Describes the levels of similarity between the input address elements and the reference data. Select the port from the Status Info port group in the Basic Model.

Element Relevance

Identifies the address elements that the mail carrier requires to identify a mailbox for the address. Select the port from the Status Info port group in the Basic Model.

Element Result Status

Describes any update that address validation makes to the to the input address. Select the port from the Status Info port group in the Basic Model.

Extended Element Result Status

Indicates the presence of additional data for an address in the reference data. The port can contain detailed information about the updates that address validation makes to an address. Select the port from the Status Info port group in the Basic Model.

Geocoding Status

Describes the type of geocoding data that address validation returns for an address. Select the port from the Geocoding port group in the Basic Model.

Mailability Score

Indicates the likelihood that the mail carrier can deliver a mail item to the address. Select the port from the Status Info port group in the Basic Model.

Match Code

Describes the results of address validation and address correction operations on an input address. Select the port from the Status Info port group in the Basic Model.

Result Percentage

Represents the degree of similarity between an input address and the corresponding output address as a percentage value. Select the port from the Status Info port group in the Basic Model.

Element Status Code Definitions

The Element Input Status, Element Relevance, Element Result Status, and Extended Element Result Status ports provides status information about the validity of input and output data elements. Select the element ports to review the outcomes of an address validation operation.

The codes contain the following information:

- Element Input Status codes represent the quality of the match found between the input address data and the reference data.
- Element Relevance codes identify the address elements that are necessary for address delivery in the destination country.
- Element Result Status codes describe any change made to the input data during processing.
- Extended Element Result Status codes indicate that the address reference data contains additional information about the address element.

Each port returns a 20-character code in which each character refers to a different address data element. When you read the output codes on element ports, you must know the element that each character refers to. The 20 characters consist of 10 pairs. The two codes in each pair represent a type of address information. For example, the first position in the return code represents basic postal code information.

Note: The Address Resolution Code port returns a 20-character string based on the same address elements as the Element Status ports.

The following table describes the address elements that the values at each position identify:

Position	Address Element	Description	Address Element Example
1	Postal code level 0	Basic postal code information, such as a five-digit ZIP Code.	The five-digit ZIP Code 10118
2	Postal code level 1	Additional postal code information, such as the final four digits of a ZIP+4 Code.	0110, in the ZIP+4 Code 10118-0110
3	Locality level 0	Primary location, such as city or town.	London, in England
4	Locality level 1	Dependent locality, suburb, village.	Islington, in London
5	Province level 0	Primary region within a country, such as a United States state name, Canadian province name, Swiss canton.	New York State
6	Province level 1	United States county name.	Queens County, in New York State
7	Street level 0	Primary street information.	South Great George's Street
8	Street level 1	Dependent street information.	George's Arcade, on South Great George's Street

Position	Address Element	Description	Address Element Example
9	Number level 0	Building or house number related to the primary street.	460, on South Great George's Street
10	Number level 1	Building or house number related to the dependent street.	81, on George's Arcade
11	Delivery service level 0	Case postale or post box descriptor and number.	PO Box 111
12	Delivery service level 1	Code of the post office that is responsible for delivery.	MAIN STN
13	Building level 0	Building name or number. Does not identify a house number.	Alice Tully Hall
14	Building level 1	Additional building name or number.	Starr Theater, at Alice Tully Hall
15	Sub-building level 0	Apartment, suite, or floor name or number.	80, in 350 5th Avenue, Floor 80
16	Sub-building level 1	Apartment, suite, or floor information, when paired with Sub-building level 0 information.	80-18, where 18 is the suite number and 80 is the floor number
17	Organization level 0	Company name.	AddressDoctor® GmbH
18	Organization level 1	Additional corporate information, such as a parent company.	Informatica Corporation
19	Country level 0	Country name.	United States of America
20	Country level 1	Territory.	United States Virgin Islands

When a port name has a number suffix, level 0 refers to data on port number 1 and level 1 refers to data on port numbers 2 through 6.

Level 0 information can precede or follow level 1 information in a printed address. For example, Postal code level 1 follows Postal code level 0, and Locality level 1 precedes Locality level 0.

Address Resolution Code Output Port Values

Address Resolution Code is a 20-character string in which each character in the string represents a different input address element. The value for a character describes any non-valid address element at the corresponding position in the address.

The following table describes the Address Resolution Code port values:

Code	Description
2	The address element is required for delivery but is not present in the input address. The address reference data contains the missing address element. An output of 2 indicates that address is not valid for delivery without the address element.
3	The address element is a house number or street number that is outside the valid range for the address. For example, the address element contains a house number that does not exist on the specified street. Suggestion list mode returns alternative addresses.
4	Address validation cannot verify or correct the address element because the input address contains more than one instance of the element.
5	The address element is ambiguous in the current address, and the address reference data contains alternatives. Address validation copies the input element to the output address. For example, the address element is a valid post code that does not match a valid locality in the address.
6	The address element contradicts another element in the address. Address validation cannot determine the correct element for the address. The output address copies the input address.
7	The address element cannot be corrected without multiple changes to the address. Address validation can correct the address, but the number of changes indicate that the address is not reliable.
8	The data does not conform to mail carrier validation rules.

Element Input Status Output Port Values

Element Input Status is a 20-character string in which each character represents a different input address element. The value for each character represents the type of processing performed on the address element.

Find the port in the Status Info port group.

The following table describes the codes that Element Input Status can return in each position on the output string in batch, certified, or suggestion list mode:

Code	Description
0	The input address contains no data at the current position.
1	The reference data does not contain the data at the current position.
2	Cannot check the data because the reference data is missing.

Code	Description
3	The data at the current position is incorrect. The reference database suggests that the number or delivery service value is outside the range that the reference data expects. In batch and certified modes, the transformation passes the input data at the current position uncorrected as output.
4	The data at the current position matches the reference data but contains errors.
5	The data at the current position matches the reference data, but the transformation corrected or standardized the data.
6	The data at the current position matches the reference data without any error.

The following table describes the codes that Element Input Status can return in each position on the output string in parse mode:

Code	Description
0	The input address contains no data at the current position.
1	The transformation moved the element at the current position to another position in the output address.
2	The element at the current position matched the reference data value, but the transformation normalized the element in the output address.
3	The data at the current position is correct.

Element Relevance Output Port Values

Element Relevance values indicates if an address element is required for postal delivery. Find the port in the Status Info port group.

The Element Relevance value is a 20-character string in which each character can represent a different type of address data. After you run the address validation mapping, review the port output to identify the address elements that are necessary for each address. Use the results to verify that you selected the right output ports for the address data. If you do not select an output port for a relevant address data element, the output for that address will not be valid.

The following table describes the codes that Element Relevance can return in each position on the output string:

Code	Description
0	Not relevant to delivery to the address.
1	Relevant to delivery to the address. The national postal carrier cannot deliver to the address without the data at this position in the output string.

Note: Element Relevance values are available for addresses with a Match Code value of Cx or Vx in batch mode or Cx, Vx, I3, or I4 in interactive mode. Other assessment codes such as Element Input Status, Element

Result Status, Extended Element Result Status, and Address Resolution Code return values regardless of the Match Code value.

Element Result Status Output Port Values

Element Result Status is a 20-character string in which each character represents a different input address element. The value for each character describes any update that the validation process makes to the address element.

Find the port in the Status Info port group.

The following table describes the Element Result Status port values:

Code	Description
0	The output address contains no data at the current position.
1	The transformation cannot find the data at the current position in the reference data. The transformation copies the input data to the output data.
2	Data at the current position is not checked but is standardized.
3	Data at the current position is checked but does not match the reference data. The reference data suggests that the number data is not in the valid range. The transformation copies the input data to the output port. Applies in batch mode.
4	The transformation copies the input data to the output data because the reference data is missing.
5	Data at the current position is validated but not changed because multiple matches exist in the reference data. Applies in batch mode.
6	Data validation deleted the input value at the current position.
7	Data at the current position is validated but the input data contained a spelling error. Validation corrected the error with a value from the reference data.
8	Data at the current position is validated and updated with a value from the reference data. A value of 8 can also mean that the reference database contains additional data for the input element. For example, validation can add a building number or sub-building number if it finds a perfect match for the street name or building name.
9	Data at the current position is validated but not changed, and the delivery status is not clear. For example, the DPV value is wrong.
C	Data at the current position is validated and verified, but the name data is out of date. Validation changed the name data.
D	Data at the current position is validated and verified but changed from an exonym to an official name.

Code	Description
E	Data at the current position is validated and verified. However, address validation standardized the character case or the language. Address validation can change the language if the value fully matches a language alternative. For example, address validation can change "Brussels" to "Bruxelles" in a Belgian address.
F	Data at the current position is validated, verified, and not changed, due to a perfect match with the reference data.

Positions 19 and 20 in the output string relate to country data.

The following table describes the values that validation may return for positions 19 and 20:

Code	Description
0	The output address contains no data at the current position.
1	Address validation does not recognize the country data.
4	Address validation identifies the country from the Default Country value in the Address Validator transformation.
5	Address validation cannot determine the country because the reference data contains multiple matches.
6	Address validation identifies the country from a script.
7	Address validation identifies the country from the address format.
8	Address validation identifies the country from major town data.
9	Address validation identifies the country from province data.
C	Address validation identifies the country from territory data.
D	Address validation identifies the country from the country name, but the name contains errors.
E	Address validation identifies the country from the address data, for example from an ISO code or a country name.
F	Address validation identifies the country from the Force Country value set in the Address Validator transformation.

Extended Element Result Status Output Port Values

Extended Element Result Status is a 20-character string in which each character represents a different input address element. The port output codes supplement the status data on the Element Input Status port and Element Result Status port. The port output code can also indicate the presence of additional information about an address element in the reference data.

Find the port in the Status Info port group.

The following table describes the Extended Element Result Status port values:

Code	Description
1	Address reference data contains additional information about the address element. Address validation does not require the additional information.
2	Address validation updated the address element to resolve a data error or format error. Address validation did not verify the address element.
3	Address validation updated the address element to resolve a data error or format error. Address validation verified the number data in the address element.
4	Address validation moved the address element to another field to resolve a format error.
5	Address reference data contains an alternative version of the address element, such as a preferred locality name.
6	Address validation did not verify all parts of the address element. The element includes data that address validation cannot validate.
7	Address validation found a valid address element in the wrong position in an address. Address validation moved the address element to the correct position.
8	Address validation found a valid address element in the wrong data field. Address validation moved the address element to the correct field.
9	Address validation generated the output element according to mail carrier validation rules.
A	Address validation found address elements from different address types that are eligible for the current position. Address validation selected the output address element that conforms the mail carrier rules in the destination country.
B	Address validation cannot determine the element relevance. Address validation returns the default value for the country that the address specifies.
C	Suggestion list mode. Address validation can return additional address suggestions for the address element. To return the additional suggestions, update the Max Result Count property for the Address Validator transformation.
D	Address validation interpolated the numeric data in the address element.
E	Address validation cannot return the address element in the preferred language. Address validation returns the element in the default language.
F	Address code lookup mode. The input address is out of date.

Mailability Score Output Port Values

The Mailability Score value represents an estimate of the deliverability of the output address. Use the mailability score as a general indicator of the deliverability of the address. Find the port in the Status Info port group.

The Address Validator transformation considers multiple factors when it calculates the mailability score. The transformation bases the calculations principally on the Match Code value and the Element Result Status

value for the address. Other factors that influence the mailability score include the postal relevance of the address values and the granularity of the reference data for the country.

The Mailability Score port value provides an estimate of the deliverability of the address. The score is not a precise or definitive indicator of the deliverability of the address.

The following table describes the Mailability Score output codes:

Value	Summary	Description
5	Completely Confident	Indicates that address validation checked and verified all relevant elements of the input address.
4	Almost Certain	Indicates either of the following scenarios: <ul style="list-style-type: none"> - One or more relevant address elements cannot be checked due to absent reference data. Other address elements are verified. - Address validation corrected one or more relevant elements with a very high degree of confidence. This occurs when address validation finds a single match between the input address and the reference data and the degree of variation is very low.
3	Should Be Fine	Indicates that address validation corrected one or more relevant elements in the input address. Address validation found a single match between the input address and the reference data, and the degree of variation is acceptable.
2	Fair Chance	Indicates that address validation cannot correct or verify the address for one of the following reasons: <ul style="list-style-type: none"> - Address validation cannot identify a candidate match in the reference data with sufficient confidence. - Address validation found multiple candidate matches with similar confidence levels. <p>The mail carrier might be able to deliver to the address.</p>
1	Risky	Indicates that address validation can find partial reference data matches only for the input address.
0	Undeliverable	Indicates that address validation cannot find a match for the address in the reference data. The input address is missing too many elements, or address validation cannot verify a majority of the elements in the address.

Match Code Output Port Values

The Match Code value summarizes the results of the comparison of the input address to the reference data. The code also summarizes any correction that the transformation made to the address. Find the port in the Status Info port group.

The following table describes the Match Code output port values:

Code	Description
A1	Address code lookup found a partial address or a complete address for the input code.
A0	Address code lookup found no address for the input code.
C4	Corrected. All postally relevant elements are checked.

Code	Description
C3	Corrected. Some elements cannot be checked.
C2	Corrected, but the delivery status is unclear due to absent reference data.
C1	Corrected, but the delivery status is unclear because user standardization introduced errors.
I4	Data cannot be corrected completely, but there is a single match with an address in the reference data.
I3	Data cannot be corrected completely, and there are multiple matches with addresses in the reference data.
I2	Data cannot be corrected. Batch mode returns partial suggested addresses.
I1	Data cannot be corrected. Batch mode cannot suggest an address.
N7	Validation error. Address validation did not take place because single-line validation is not unlocked.
N6	Validation error. Address validation did not take place because single-line validation is not supported for the destination country.
N5	Validation error. Address validation did not take place because the reference database is out of date.
N4	Validation error. Address validation did not take place because the reference data is corrupt or badly formatted.
N3	Validation error. Address validation did not take place because the country data cannot be unlocked.
N2	Validation error. Address validation did not take place because the required reference database is not available.
N1	Validation error. Address validation did not take place because the country is not recognized or not supported.
Q3	Suggestion List mode. Address validation can retrieve one or more complete addresses from the address reference data that correspond to the input address.
Q2	Suggestion List mode. Address validation can combine the input address elements and elements from the address reference data to create a complete address.
Q1	Suggestion List mode. Address validation cannot suggest a complete address. To generate a complete address suggestion, add data to the input address.
Q0	Suggestion List mode. There is insufficient input data to generate a suggestion.
RB	Country recognized from abbreviation. Recognizes ISO two-character and ISO three-character country codes. Can also recognize common abbreviations such as "GER" for Germany.
RA	Country recognized from the Force Country property.

Code	Description
R9	Country recognized from the Default Country property.
R8	Country recognized from the country name.
R7	Country recognized from the country name, but the validation process identified errors in the country data.
R6	Country recognized from territory data.
R5	Country recognized from province data.
R4	Country recognized from major town data.
R3	Country recognized from the address format.
R2	Country recognized from a script.
R1	Country not recognized because multiple matches are available.
R0	Country not recognized.
S4	Parse mode. The address was parsed perfectly.
S3	Parse mode. The address was parsed with multiple results.
S1	Parse mode. There was a parsing error due to an input format mismatch.
V4	Verified. The input data is correct. Address validation checked all postally relevant elements, and inputs matched perfectly.
V3	Verified. The input data is correct, but some or all elements were standardized, or the input contains outdated names or exonyms.
V2	Verified. The input data is correct, but some elements cannot be verified because of incomplete reference data.
V1	Verified. The input data is correct, but user standardization has negatively impacted deliverability. For example, the post code length is too short.

Geocoding Status Output Port Values

The following table describes the Geocoding Status output port values. Find this port in the Geocoding port group.

Select this port if you have installed geocoding reference data for an input address country.

Value	Description
EGC0	Cannot append geocodes to the input address because geocodes are not available for the address.
EGC1-3	Reserved for future use.

Value	Description
EGC4	Geocodes are partially accurate to the postal code level.
EGC5	Geocodes are accurate to the postal code level.
EGC6	Geocodes are accurate to the locality level.
EGC7	Geocodes are accurate to the street level.
EGC8	Geocodes are accurate to the house number level. The geocodes estimate the house number location and include an offset to the side of the street that contains the mailbox.
EGC9	Geocodes are accurate to the arrival point or rooftop.
EGCA	Geocodes are accurate to the center of the parcel of land.
EGCC	The geocode database is corrupted.
EGCN	Cannot find the geocode database.
EGCU	The geocode database is not unlocked.

Note: Informatica no longer issues reference data for parcel centroid and rooftop geocoding.

Address Validator Transformation General Settings

Configure the general settings to set up parameters required for address validation.

You can configure the following properties on the **General Settings** view:

Default Country

Specifies the address reference data set that the transformation uses if it cannot identify a country destination in the input address. Select None if your data includes country information.

You can also set the default country as an advanced property on the transformation.

Force Country

Optional property. Replaces the country name or abbreviation in the input address with the name or abbreviation of the default country. If the input address does not identify a country, the transformation appends the default country data to the address.

Line Separator

Specifies the delimiter symbol that separates data fields within a single-line address.

You can also specify a line separator as an advanced property on the transformation.

Casing Style

Sets the character case style for output data. Select the Mixed option to follow the address reference data standard for initial capital letters. Select the Preserved option to write the address in the casing style that the address reference data uses.

You can also set the casing style as an advanced property on the transformation.

Mode

Determines the type of validation that the transformation performs.

Select one of the following options:

Mode Type	Description
Address code lookup	Returns a partial address or a complete address from the reference data when you provide an address code as an input. Several countries support address codes that represent the locality, street, building, or unique mailbox for an address.
Batch	Performs address validation on the records in a data set. Batch validation focuses on address completeness and deliverability. Batch mode does not return suggestions for poor-quality addresses. Batch is the default mode.
Certified	Performs address validation on the records in a data set to the certification standards of the specified country. The certification standards require that each address identifies a unique mailbox. You can perform certified address validation on addresses in Australia, France, New Zealand, the United Kingdom, and the United States.
Country recognition	Determines a destination country for the postal address. The transformation does not perform address validation in country recognition mode.
Interactive	Completes an incomplete valid address. When an incomplete input address matches more than one address in the reference data, the transformation returns all valid addresses up to the limit that the Max Result Count specifies.
Parse	Parses data into address fields. The transformation does not perform address validation in parse mode.
Suggestion list	Returns a list of valid addresses from the reference data when an input address contains fragmentary information. When an address fragment matches more than one address in the reference data, the transformation returns all valid addresses up to the limit that the Max Result Count specifies.

You can also set the mode as an advanced property on the transformation.

Address Validation Properties in the Preferences Window

You can view the properties of the address validation engine and the address reference data files that the engine reads in the Developer tool. The Developer tool exposes the properties of the engine that the Data Integration Service uses to run address validation mappings. The Developer tool lists the properties for the Content Management Service that governs the address validation operations.

Use the **Preferences** window in the Developer tool to review the properties. Select the **Content Status** option on the **Preferences** window to identify the Content Management Service that the current Data Integration Service uses. To view the properties, select the local Content Management Service.

You can view the following properties:

Address Validation Data

The address validation data properties list the types of reference data that the current Content Management Service can provide to the Data Integration Service. The properties also indicate the countries to which the reference data applies.

Address Validation Engine

The address validation engine properties include the current engine version, the engine in which the certification components were most recently updated, and the data preloading method.

Address Validation License

The address validation license properties include license information for the reference data that the current Content Management Service can provide to the Data Integration Service.

Address Validation Data Properties

The address validation data properties list the types of reference data that the current Content Management Service can provide to the Data Integration Service. The properties also include the countries to which the reference data applies.

The following table describes the data properties that display when you select the Content Management Service in the **Content Status** view:

Property	Description
Country ISO	The country to which the address reference data file applies. The property shows the ISO three-character code for the country.
Expiry Date	The date on which the current file expires. Informatica releases a newer file on the expiry date. You can use the current address reference data file after the expiry date, but the data in the file may no longer be accurate.
Country Type	The type of address validation that you can perform with the data. You select the processing type in the Mode option on the General Settings tab. If the mode that you select does not correspond to an address data file on the domain, the address validation mapping will fail.
Unlock Expiry Date	The date on which the license expires. You cannot use any version of the file after the unlock expiry date. The Unlock Expiry Date property and Expiration Date property on the Address Validation License Properties view represent the same information.
Unlock Start Date	The date on which the license becomes effective for the mode that the Country Type property identifies and the country that the Country ISO property identifies. You cannot use any version of the file before the unlock start date.

Address Validation License Properties

The address validation license properties include license information for the reference data that the current Content Management Service can provide to the Data Integration Service.

The following table describes the license properties that display when you select the Content Management Service in the **Content Status** view:

Property	Description
Unlock Code	The license code that unlocks the reference data for the mode that the Code Type property identifies. The Developer tool displays the first four characters of the code and masks the other characters.
Code Type	The mode of address validation that you can perform with the data that the license specifies. Informatica issues a single license code for each mode. The license code can apply to one or more countries. You select the processing type in the Mode option on the General Settings tab. If the mode that you select does not correspond to an address data file on the domain, the address validation mapping will fail.
Country List	The countries for which the unlock code unlocks the reference data. The Country List property contains one or more ISO three-character codes for each country.
Status	The status of the license code. The property returns OK when the license file is valid.
Expiration Date	The date on which the license expires. The Expiration Date property and the Unlock Expiry Date property on the Address Validation Data Properties view represent the same information.

Address Validation Engine Properties

The address validation engine properties include the current engine version, the engine in which the certification components were most recently updated, and the data preloading method.

The following table describes the engine properties that display when you select the Content Management Service in the **Content Status** view:

Property	Value
Engine Version	The version of the address validation engine that the Data Integration Service runs.
CASS Version	The version of the address validation engine in which Informatica most recently updated the CASS certification components. Use the property to identify the engine version in a CASS certification report. The property also includes the CASS certification cycle that the engine supports. For example, the engine might support certification cycle N.
AMAS Version	The version of the address validation engine in which Informatica most recently updated the AMAS certification components. Use the property to identify the engine version in a AMAS certification report.
SendRight Version	The version of the address validation engine in which Informatica most recently updated the SendRight certification components. Use the property to identify the engine version in a SendRight certification report.

Property	Value
SERP Version	The version of the address validation engine in which Informatica most recently updated the SERP certification components. Use the property to identify the engine version in a SERP certification report.
SNA Version	The version of the address validation engine in which Informatica most recently updated the SNA certification components. Use the property to identify the engine version in a SNA certification report.
Preloading Method	The method that the Data Integration Service uses to preload reference database into memory. The Content Management Service properties specify the countries for which the Data Integration Service preloads reference data. The possible values are MAP and LOAD. The default value is MAP. The MAP method and the LOAD method both allocate a block of memory and then read the reference data into the block. However, the MAP method can share reference data between multiple processes.
Cache Size	The size of the data cache that the Data Integration Service uses for reference data that the service does not preload. The possible values are NONE, SMALL, and LARGE. The default value is LARGE.
Maximum Memory Usage	The number of megabytes of memory that the address validation engine can allocate. The default value is 4096.
Maximum Address Object Count	The maximum number of address validation instances that the Data Integration Service can run at the same time. The default value is 3.
Maximum Thread Count	The maximum number of threads that address validation can use. The default value is 2.
Maximum Result Count	The maximum number of addresses that address validation can return when you run a mapping in suggestion list mode. The default value is 20. The upper limit on the property is 100.
Current Date	The current date. The Developer tool returns the property values that apply on the current date.
Write XML BOM	Indicates whether the Data Integration Service writes a byte order mark in the GetConfig.xml file. The possible values are ALWAYS, IF_NECESSARY, and NEVER. The default value is IF_NECESSARY.
XML Encoding	Identifies the XML encoding that the address validation engine uses to read and write data.

Address Validation Advanced Properties

Configure the advanced properties to determine how the Data Integration Service processes data for the Address Validator transformation.

Alias Locality

Determines whether address validation replaces a valid locality alias with the official locality name.

A locality alias is an alternative locality name that the USPS recognizes as an element in a deliverable address. You can use the property when you configure the Address Validator transformation to validate United States address records in Certified mode.

The following table describes the Alias Locality options:

Option	Description
Off	Disables the Alias Locality property.
Official	Replaces any alternative locality name or locality alias with the official locality name. Default option.
Preserve	Preserves a valid alternative locality name or locality alias. If the input locality name is not valid, address validation replaces the name with the official name.

Alias Street

Determines whether address validation replaces a street alias with the official street name.

A street alias is an alternative street name that the USPS recognizes as an element in a deliverable address. You can use the property when you configure the Address Validator transformation to validate United States address records in Certified mode.

The following table describes the Alias Street options:

Option	Description
Off	Does not apply the property.
Official	Replaces any alternative street name or street alias with the official street name. Default option.
Preserve	Preserves a valid alternative street name or street alias. If the input street name is not valid, address validation replaces the name with the official name.

Casing Style

Specifies the character case style that the transformation applies to the output address data.

The following table describes the Casing Style options:

Option	Description
Assign Parameter	Uses a parameter that you define to set the casing style.
Lower	Writes the output address in lowercase letters.
Mixed	Uses the casing style in use in the destination country when it is possible to do so.

Option	Description
Preserved	Applies the casing style that the address reference data uses. Default option.
No Change	Does not apply a casing style to the address. Note: The No Change option does not guarantee that the output address will match the case of the input address. If address validation replaces an address element with an element from the reference data, the element follows the case that the reference data uses.
Upper	Writes the output address in uppercase letters.

You can also configure the casing style on the **General Settings** tab.

Parameter Usage

You can use one of the following parameters to specify the casing style:

- LOWER. Writes the output address in lowercase letters.
- MIXED. Uses the casing style in use in the destination country when it is possible to do so.
- NATIVE. Applies the casing style that the address reference data uses. Default option. Matches the **Preserved** option.
- NOCHANGE. Does not apply a casing style to the address.
- UPPER. Writes the output address in uppercase letters.

Enter the parameter value in uppercase.

Country of Origin

Identifies the country in which the address records are mailed.

Select a country from the list. The property is empty by default.

Country Type

Determines the format of the country name or abbreviation in Complete Address or Formatted Address Line port output data. The transformation writes the country name or abbreviation in the standard format of the country you select.

The following table describes the Country Type options:

Option	Country
ISO 2	ISO two-character country code
ISO 3	ISO three-character country code
ISO #	ISO three-digit country code
Abbreviation	(Reserved for future use)
CN	Canada

Option	Country
DA	(Reserved for future use)
DE	Germany
EN	Great Britain (default)
ES	Spain
FI	Finland
FR	France
GR	Greece
IT	Italy
JP	Japan
HU	Hungary
KR	Korea, Republic of
NL	Netherlands
PL	Poland
PT	Portugal
RU	Russia
SA	Saudi Arabia
SE	Sweden

Default Country

Specifies the address reference data set that the transformation uses when an address record does not identify a destination country.

Select a country from the list. Use the default option if the address records include country information. Default is None.

You can also configure the default country on the **General Settings** tab.

Parameter Usage

You can use a parameter to specify the default country. When you create the parameter, enter the ISO 3166-1 alpha-3 code for the country as the parameter value. When you enter a parameter value, use uppercase characters. For example, if all address records include country information, enter NONE.

Dual Address Priority

Determines the type of address to validate. Set the property when input address records contain more than one type of valid address data.

For example, use the property when an address record contains both post office box elements and street elements. Address validation reads the data elements that contain the type of address data that you specify. Address validation ignores any incompatible data in the address.

The following table describes the options on the Dual Address Priority property:

Option	Description
Delivery service	Validates delivery service data elements in an address, such as post office box elements.
Postal admin	Validates the address elements required by the local mail carrier. Default option.
Street	Validates street data elements in an address, such as building number elements and street name elements.

Element Abbreviation

Determines if the transformation returns the abbreviated form of an address element. You can set the transformation to return the abbreviated form if the address reference data contains abbreviations.

For example, the United States Postal Service (USPS) maintains short and long forms of many street and locality names. The short form of HUNTSVILLE BROWNSFERRY RD is HSV BROWNS FRY RD. You can select the Element Abbreviation property when the street or locality values exceed the maximum field length that the USPS specifies.

The option is cleared by default. Set the property to ON to return the abbreviated address values. The property returns the abbreviated locality name and locality code when you use the transformation in batch mode. The property returns the abbreviated street name, locality name, and locality code when you use the transformation in certified mode.

Execution Instances

Specifies the number of threads that the Data Integration Service tries to create for the current transformation at run time. The Data Integration Service considers the Execution Instances value if you override the Maximum Parallelism run-time property on the mapping that contains the transformation. The default Execution Instances value is 1.

The Data Integration Service considers multiple factors to determine the number of threads to assign to the transformation. The principal factors are the Execution Instances value and the values on the mapping and on the associated application services in the domain.

The Data Integration Service reads the following values when it calculates the number of threads to use for the transformation:

- The *Maximum Parallelism* value on the Data Integration Service. Default is 1.
- Any *Maximum Parallelism* value that you set at the mapping level. Default is Auto.
- The *Execution Instances* value on the transformation. Default is 1.

If you override the Maximum Parallelism value at the mapping level, the Data Integration Service attempts to use the lowest value across the properties to determine the number of threads.

If you use the default Maximum Parallelism value at the mapping level, the Data Integration Service ignores the Execution Instances value.

The Data Integration Service also considers the *Max Address Object Count* property on the Content Management Service when it calculates the number of threads to create. The *Max Address Object Count* property determines the maximum number of address validation instances that can run concurrently in a mapping. The *Max Address Object Count* property value must be greater than or equal to the *Maximum Parallelism* value on the Data Integration Service.

Rules and Guidelines for the Execution Instances Property

Consider the following rules and guidelines when you set the number of execution instances:

- Multiple users might run concurrent mappings on a Data Integration Service. To calculate the correct number of threads, divide the number of central processing units that the service can access by the number of concurrent mappings.
- In PowerCenter, the *AD50.cfg* configuration file specifies the maximum number of address validation instances that can run concurrently in a mapping.
- When you use the default Execution Instances value and the default Maximum Parallelism values, the transformation operations are not partitionable.
- When you set an Execution Instances value greater than 1, you change the Address Validator transformation from a passive transformation to an active transformation.

Flexible Range Expansion

Imposes a practical limit on the number of addresses that the Address Validator transformation returns when you set the Ranges to Expand property. You can set the Ranges to Expand property and the Flexible Range Expansion property when you configure the transformation to run in suggestion list mode.

The Ranges to Expand property determines how the transformation returns address suggestions when an input address does not contain house number data. If the input address does not include contextual data, such as a full post code, the Ranges to Expand property can generate a large number of very similar addresses. The Flexible Range Expansion property restricts the number of addresses that the Ranges to Expand property generates for a single address. Set the Flexible Range Expansion property to On when you set the Ranges to Expand property to All.

The following table describes the options on the Flexible Range Expansion property:

Option	Description
On	Address validation limits the number of addresses that the Ranges to Expand property adds to the suggestion list. Default option.
Off	Address validation does not limit the number of addresses that the Ranges to Expand property adds to the suggestion list.

Note: The Address Validator transformation applies the Flexible Range Expansion property in a different way to every address that it returns to the suggestion list. The transformation does not impose a fixed limit on the number of expanded addresses in the list. The transformation also considers the Max Result Count property setting when it calculates the number of expanded addresses to include in the list.

Geocode Data Type

Determines how the Address Validator transformation calculates geocode data for an address. Geocodes are latitude and longitude coordinates.

The geocoding results that the transformation returns depend on the geocoding reference data that you install. For information about geocoding reference data, contact Informatica.

Select one of the following geocode options:

Arrival point

Returns the latitude and longitude coordinates of the entrance to a building or parcel of land. Default option.

You can select the arrival point option for addresses in the following countries:

Australia, Austria, Canada, Croatia, Denmark, Estonia, Finland, France, Germany, Hungary, Italy, Latvia, Liechtenstein, Lithuania, Luxembourg, Mexico, Monaco, Netherlands, Norway, Poland, Slovakia, Slovenia, Sweden, Switzerland, and the United States.

If you specify arrival point geocodes and the Address Validator transformation cannot return the geocodes for an address, the transformation returns interpolated geocodes.

Standard

Returns the estimated latitude and longitude coordinates of the entrance to the building or parcel of land. An estimated geocode is also called an interpolated geocode.

The Address Validator transformation uses the nearest available geocodes in the reference data to estimate the geocodes for the address.

Note: Informatica no longer issues reference data for parcel centroid or rooftop geocoding.

Parameter Usage

You can use a parameter to specify the geocode type. Enter ARRIVAL_POINT or NONE. To return the standard geocodes, enter NONE.

Enter the parameter value in uppercase.

Global Max Field Length

Determines the maximum number of characters on any line in the address. If the Address Validator transformation writes an output address line that contains more characters than you specify, the transformation abbreviates the address elements on the line.

Use the property to control the line length in the address. For example, the SNA standards require that an address contains no more than 38 characters on any line. If you generate addresses to the SNA standard, set the Global Max Field Length to 38.

Default is 1024.

Parameter Usage

You can use a parameter to specify the maximum number of addresses. To set the parameter value, enter an integer from 0 through 1024.

Global Preferred Descriptor

Determines the format of the building descriptor, sub-building descriptor, and street descriptor that the Address Validator transformation writes to the output data. Select a descriptor when the address reference data for the destination country contains a range of descriptors for one or more of the data elements.

The following table describes the options on the property:

Option	Description
Database	Returns the descriptor that the reference database specifies for the element in the address. If the database does not specify a descriptor for the address, the transformation copies the input value to the output address. Database is the default value.
Long	Returns the complete form of the descriptor, for example <i>Street</i> .
Preserve Input	Copies the descriptor from the input address to the output address. If the input descriptor is not a valid version of the descriptor, the transformation returns an equivalent valid descriptor from the reference database.
Short	Returns an abbreviated form of the descriptor, for example <i>St</i> .

Input Format Type

Describes the most common type of information contained in unfielded input data. Use the Input Format Type property when you connect input data to the Complete Address or Formatted Address Line ports. Select the option that best describes the information in the mapping source data.

Select one of the following options:

- All
- Address
- Organization
- Contact
- Organization/Contact
The address includes organization information and contact information.
- Organization/Dept
The address includes organization information and department information.

Default is All.

Input Format With Country

Specifies whether the input contains country data. Select the property if you connect input data to the Complete Address or Formatted Address Line input ports and if the data contains country information.

The option is cleared by default.

Line Separator

Specifies the delimiter symbol that indicates line breaks in a formatted address.

Select one of the following options:

- Assign a parameter to identify the line separator
- Carriage return (CR)
- Comma
- Line Feed (LF)
- None
- Semicolon
- Tab
- Windows New Line (CRLF)

Default is semicolon.

You can also configure the line separator on the **General Settings** tab.

Parameter Usage

You can use a parameter to specify the line separator. The parameter value is case-sensitive. Enter the parameter value in uppercase characters.

Enter one of the following values:

- CR
- CRLF
- COMMA
- LF
- PIPE
- SEMICOLON
- SPACE
- TAB

Matching Alternatives

Determines whether address validation recognizes alternative place names, such as synonyms or historical names, in an input address. The property applies to street, locality, and province data.

Note: The Matching Alternatives property does not preserve alternative names in a validated address.

The following table describes the Matching Alternatives options:

Option	Description
All	Recognizes all known alternative street names and place names. Default option.
Archives only	Recognizes historical names only. For example, address validation validates "Constantinople" as a historical version of "Istanbul."

Option	Description
None	Does not recognize alternative street names or place names.
Synonyms only	Recognizes synonyms and exonyms only. For example, address validation validates "Londres" as an exonym of "London."

Matching Extended Archive

Determines whether address validation returns a unique delivery point code for an out-of-date Japanese address.

The address reference data files for Japan include data for out-of-date or retired addresses alongside the current addresses for the corresponding mailboxes. When you select the Matching Extended Archive property, address validation returns the delivery point code for the current version of each address. Address validation also writes a value to the Extended Element Result Status port to indicate that the input address is out of date.

To retrieve the current address from the address reference data, enter the address code as an input element.

The following table describes the Matching Extended Archive options:

Option	Description
Off	Does not apply the property.
On	Returns the address code for the current version of an out-of-date Japanese address.

The Matching Extended Archive property uses supplementary data and address code lookup data for Japan. To apply the property in address validation, configure the transformation to run in address code lookup mode.

Matching Scope

Determines the amount of data that the transformation matches against the address reference data during address validation.

The following table describes the Matching Scope options:

Option	Description
All	Validates all selected ports. Default option.
Delivery Point	Validates building and sub-building address data in addition to data that the Street option validates.
Locality	Validates province, locality, and postcode data.
Street	Validates street address data in addition to data that the Locality option validates.

Max Result Count

Determines the maximum number of addresses that address validation can return in suggestion list mode.

You can set a maximum number in the range 1 through 100. Default is 20.

Note: Suggestion list mode performs an address check against address reference data and returns a list of addresses that are possible matches for the input address. When you verify an address in suggestion list mode, address validation returns the best matches first.

Parameter Usage

You can use a parameter to specify the maximum number of addresses. To set the parameter value, enter an integer from 0 through 100.

Mode

Determines the type of address analysis that the transformation performs. You can also configure the mode on the **General Settings** tab of the transformation.

RELATED TOPICS:

- [“ Address Validator Transformation General Settings” on page 98](#)

Optimization Level

Determines how the transformation matches input address data and address reference data. The property defines the type of match that the transformation must find between the input data and reference data before it can update the address record.

The following table describes the Optimization Level options:

Option	Description
Narrow	The transformation parses building numbers or house numbers from street information before it performs validation. Otherwise the transformation validates the input address elements strictly according to the input port structure. The narrow option performs the fastest address validation, but it can return less accurate results than other options.
Standard	The transformation parses multiple types of address information from the input data before it performs validation. When you select the standard option, the transformation updates an address if it can match multiple input values with the reference data. Default is Standard.
Wide	The transformation uses the standard parsing settings and performs additional parsing operations across the input data. When you select the wide option, the transformation updates an address if it can match at least one input value with the reference data. The wide option increases mapping run times.

Parameter Usage

You can use a parameter to specify the optimization level. Enter NARROW, STANDARD, or WIDE. Enter the parameter value in uppercase.

Output Format Type

Describes the most common type of information that the transformation writes on the Complete Address or Formatted Address Line output port. Select the option that best describes the data that you expect on the output port.

Select one of the following options:

- All
- Address
- Organization
- Contact
- Organization/Contact
The address includes organization information and contact information.
- Organization/Dept
The address includes organization information and department information.

Default is All.

Output Format With Country

Determines if the transformation writes country identification data to the Complete Address or Formatted Address Line output ports.

The option is cleared by default.

Preferred Language

Determines the languages in which the Address Validator transformation returns address elements when the reference data sets contain data in more than one language. You can set a preferred language for addresses in Belgium, Canada, China, Finland, Hong Kong, Ireland, Israel, Macau, Switzerland, and Taiwan.

The Address Validator transformation can return address data in the following languages:

- The default language for the address in the address reference data. The default language is the main spoken language in the region to which each address belongs.
- Any other language that the address reference data supports for an address. For example, the Belgium reference data contains address elements in Flemish, French, and German.

The address reference data might contain data for a single address element or for a complete address in multiple languages. For example, address validation can return all address elements for Ireland in the English language and can return street, locality, and province information in the Irish language. Additionally, the reference data might specify different default languages for addresses in different parts of a country. For example, in the Switzerland reference data, the default language varies from region to region between French, German, and Italian.

The following table summarizes the options that you can select on the Preferred Language property:

Option	Description
Database	Returns each address in the language that the address reference data specifies. The address reference data might specify different languages for addresses in different regions in a country. Database is the default option.
Alternative 1, Alternative 2, Alternative 3	Returns address elements in an alternative language from the reference data. The alternative languages depends on the country to which the address belongs.
English	Returns address elements in English when the reference data contains the data in English. Returns the other address elements in the default language of the region to which the address belongs.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Note: An address reference data set might contain some address elements in a non-default language but not others. If the transformation cannot find an element in the language that the property specifies, the transformation returns the element in the default language.

When you set a preferred language option, verify that the character set that the Preferred Script property specifies is compatible with the output address data that you expect.

Multilanguage Support for Belgium Addresses

The following table describes the languages that you can specify for addresses in Belgium:

Option	Description
Database	Default value. Returns addresses in the main language of the region to which the address belongs. The language might be Flemish, French, or German.
English	Returns the province, locality, and street information in English if the address reference data contains the information in English. Returns the other address elements in the main language of the region to which the address belongs.
Alternative 1	Returns the province, locality, and street information in Flemish.
Alternative 2	Returns the province, locality, and street information in French.
Alternative 3	Returns the province, locality, and street information in German.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Multilanguage Support for Canada Addresses

The following table describes the languages that you can specify for addresses in Canada:

Option	Description
Database	Default value. Returns addresses in English for all provinces except Quebec. Returns Quebec addresses in French.
English	Returns all addresses in English.
Alternative 1	Returns all addresses in English.
Alternative 2	Returns Quebec addresses in French. In provinces other than Quebec, the transformation returns the street descriptors, directional information, and province names in French and returns other address elements in English.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Multilanguage Support for China Addresses

The following table describes the languages that you can specify for addresses in China:

Option	Description
Database	Default value. Returns all address information in Chinese.
English	Returns the English-language versions of street descriptor and street directional values. Returns all other address information in the Chinese language. The English address elements omit transliteration elements such as "shi."
Alternative 1	Returns all address information in the database language.
Alternative 2	Returns all address information in the database language.
Alternative 3	Returns all address information in the database language.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Consider the following rules and guidelines when you select the preferred language:

- To return the address in the Chinese language, select Database, Alternative 1, Alternative 2, or Alternative 3.
To return the address in a Chinese character set, set the Preferred Script property to Database.
- To return street descriptor and street directional information in the English language, select English.

To return the address in a Latin or ASCII character set, set the Preferred Script property to a LATIN or ASCII value.

- If you select a LATIN or ASCII value as the preferred script and Database as the preferred language, address validation returns the address data in Pinyin.

Multilanguage Support for Finland Addresses

The following table describes the languages that you can specify for addresses in Finland:

Option	Description
Database	Default value. Returns all address information in Finnish.
Alternative 1	Returns all address information in the database language.
Alternative 2	Returns the street, locality, and province information in Swedish. Returns all other information in Finnish.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Multilanguage Support for Hong Kong Addresses

The following table describes the languages that you can specify for addresses in Hong Kong:

Option	Description
Database	Default value. Returns all address information in Chinese.
English	Returns all address information in English.
Alternative 1	Returns all address information in the database language.
Alternative 2	Returns all address information in English.
Alternative 3	Returns all address information in the database language.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Consider the following rules and guidelines when you select the preferred language for Hong Kong:

- To return the address in a Chinese character set, set the Preferred Script property to Database.
- To return the address in a Latin or ASCII character set, set the Preferred Script property to a LATIN or ASCII value.
- The language of the input data can affect the operation of the Preserve Input option on a Hong Kong address. Address validation identifies the input language as English when the input data uses 7-bit ASCII characters and includes an English-language descriptor.

Multilanguage Support for Ireland Addresses

The following table describes the languages that you can specify for addresses in Ireland:

Option	Description
Database	Default value. Returns all address information in English.
English	Returns all address information in English.
Alternative 1	Returns all address information in English.
Alternative 2	Returns the street, locality, and county information in Irish. Returns all other address information in English.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Multilanguage Support for Israel Addresses

The following table describes the languages that you can specify for addresses in Israel:

Option	Description
Database	Default value. Returns all address information in Hebrew.
English	Returns all address information in English.
Alternative 1	Returns all address information in Hebrew.
Alternative 2	Returns all address information in English.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Consider the following rules and guidelines when you select the preferred language:

- To return the addresses in a Hebrew character set, set the Preferred Script property to Database.
- To return the addresses in a Latin or ASCII character set, set the Preferred Script property to a LATIN or ASCII value.
- If you select a Latin character set as the preferred script and you select Hebrew as the preferred language, address validation transliterates the Hebrew address into Latin characters. For optimal results in a Latin character set, select English as the preferred language.

Multilanguage Support for Macau Addresses

The following table describes the languages that you can specify for addresses in Macau:

Option	Description
Database	Default value. Returns all address information in Chinese.
Alternative 1	Returns all address information in the database language.
Alternative 2	Returns all address information in Portuguese.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

- To return the address in a Chinese character set, set the Preferred Script property to Database.
- To return the address in a Latin or ASCII character set, set the Preferred Script property to a LATIN or ASCII value.
- The language of the input data can affect the operation of the Preserve Input option on a Macau address. Address validation identifies the input language as Portuguese when the input data uses 7-bit ASCII characters and includes a Portuguese-language descriptor.

Multilanguage Support for Switzerland

The following table describes the languages that you can specify for addresses in Switzerland:

Option	Description
Database	Default value. Returns addresses in the main language of the region to which the address belongs. For example, address validation returns a Zurich address in German and a Geneva address in French.
English	Returns the locality and province information in English if the reference address database contains the information in English. Returns the other address elements in the main language of the region to which the address belongs. Address validation returns the locality information in English for some localities, for example Geneva and Zurich.
Alternative 1	Returns the province and locality information in German.
Alternative 2	Returns the province and locality information in French.
Alternative 3	Returns the province and locality information in Italian.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Note: Address validation also returns street information for addresses in Biel/Bienne in the alternative language that you configure.

Multilanguage Support for Taiwan

The following table describes the languages that you can specify for addresses in Taiwan:

Option	Description
Database	Default value. Returns all address information in Chinese.
English	Returns all address information in English.
Preserve Input	Returns the address information in the input language. Address validation preserves the language if the reference data contains the address information in the input language. If address validation detects more than one supported language in the input address, it returns the address in the database language. If Address Verification cannot return an element in the input language, it returns the element in the database language.

Consider the following rules and guidelines when you select the preferred language:

- To return the address in a Chinese character set, set the Preferred Script parameter to Database.
- To return the address in a Latin or ASCII character set, set the Preferred Script parameter to a LATIN or ASCII value.
- The language of the input data can affect the operation of the Preserve Input option on a Taiwan address. Address validation identifies the input language as English when the input data uses 7-bit ASCII characters and includes an English-language descriptor.

Preferred Script

Determines the character set that the Address Validator transformation uses for output data.

The following table describes the options on the property:

Option	Description
ASCII (Simplified)	Returns an address in ASCII characters.
ASCII (Extended)	Returns an address in ASCII characters and expands any special character in an address. For example, Ö transliterates to OE.
Database	Returns an address in the character set that the address reference data uses for the default language. Database is the default value.
Latin	Returns an address in the Latin character set.
Latin (Alt)	Returns an address in an alternative Latin character set. For example, specify Latin to return a South Korea address in the Revised Romanization transliteration. Specify Latin (Alt) to return a South Korea address in the older, ISO/TR 11941 transliteration.
Postal Admin	Returns an address in the script that the postal service local to the address prefers.

Option	Description
Postal Admin (Alt)	Returns an address in a script that the postal service local to the address approves as an alternative script.
Preserve Input	Returns address data in the character set that the input address uses.

The transformation can process a data source that contains data in multiple languages and character sets. The transformation converts all input data to the Unicode UCS-2 character set and processes the data in the UCS-2 format. After the transformation processes the data, it converts the data in each address record to the character set that you specify in the property. The process is called transliteration.

Transliteration can use the numeric representations of each character in a character set when it converts characters for processing. Transliteration can also convert characters phonetically when there is no equivalent numeric representation of a character. If the Address Validator transformation cannot map a character to UCS-2, it converts the character to a space.

Note: If you update the preferred language or the preferred script on the transformation, verify that the language and the character code that you select are compatible.

Ranges To Expand

Determines how the Address Validator transformation returns suggested addresses for a street address that does not specify a house number. Use the property when the transformation runs in suggestion list mode.

The Address Validator transformation reads a partial or incomplete street address in suggestion list mode. The transformation compares the address to the address reference data, and it returns all similar addresses to the end user. If the input address does not contain a house number, the transformation can return one or more house number suggestions for the street. The Ranges to Expand property determines how the transformation returns the addresses.

The transformation can return the range of valid house numbers in a single address, or it can return a separate address for each valid house number. The transformation can also return an address for each number in the range from the lowest to the highest house number on the street.

The following table describes the options on the property:

Option	Description
All	Address validation returns a suggested address for every house number in the range of possible house numbers on the street.
None	Address validation returns a single address that identifies the lowest and highest house numbers in the valid range for the street.
Only with valid items	Address validation returns a suggested address for every house number that the address reference data recognizes as a deliverable address.

Note: Suggestion list mode can use other elements in the address to specify the valid range of street numbers. For example, a ZIP Code might identify the city block that contains the address mailbox. The Address Validator transformation can use the ZIP Code to identify the lowest and highest valid house numbers on the block.

If the transformation cannot determine a house number range within practical limits, the number of suggested addresses can grow to an unusable size. To restrict the number of addresses that the Ranges to Expand property generates, set the Flexible Range Expansion property to On.

Standardize Invalid Addresses

Determines if the address validation process standardizes the data values in an undeliverable address. The property applies to address records that return a Match Code status in the range I1 through I4.

When you standardize the data, you increase the likelihood that a downstream data process returns accurate results. For example, a duplicate analysis mapping might return a higher match score for two address records that present common address elements in the same format.

Address validation can standardize the following address elements:

- Street suffix elements, such as road and boulevard.
- Predirectional and postdirectional elements, such as north, south, east, and west.
- Delivery service elements, such as Post Office Box.
- Sub-building elements, such as apartment, floor, and suite.
- State or province names. Standardization returns the abbreviated forms of the names.

The following table describes the options on the property:

Option	Description
Off	Address validation does not correct data errors. Default option.
On	Address validation corrects data errors.

Parameter Usage

You can assign a parameter to specify the standardization policy for data errors. Enter OFF or ON as the parameter value. Enter the value in uppercase.

Tracing Level

Sets the amount of detail that is included in the log.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Certification Reports

You can generate a report that describes the status of the addresses that you submit for validation in certified mode. The report verifies that the address set meets the certification standards that the mail carrier defines.

The Address Validator transformation generates reports for the following standards:

Address Machine Approval System (AMAS)

Australia Post defines the AMAS certification standard.

Coding Accuracy Support System (CASS)

The USPS defines the CASS certification standard.

SendRight

New Zealand Post defines the SendRight certification standard.

Software Evaluation and Recognition Program (SERP)

Canada Post defines the SERP certification standard.

Note: You can use the Address Validator transformation to certify French address records to the National Address Management Service (SNA) certification standard. However, you do not generate a report for SNA address certification in the Address Validator transformation.

When you provide the mail items that meet the certification standards to the mail carrier, you submit the certification report with the address set. The report includes data about the organization. You can enter the data when you configure the Address Validator transformation. The transformation adds the organization data to the report file.

AMAS Report Fields

When you configure an AMAS report, you provide information about the organization that submits the certified address record set to Australia Post. Save or print the report, and include the report with the address records that you submit to Australia Post.

Use the **Reports** view to enter the information.

The following table describes the information that you enter:

Field	Description
Report File Name	Name and location of the report that address validation creates. By default, the Address Validator creates the report in the <code>bin</code> directory of the Data Integration Service machine. To write the report file to another location on the Data Integration Service machine, enter the file path and file name. You can enter a fully qualified path or a relative path. The relative path uses the <code>bin</code> directory as a root directory. The directory that you specify must exist before you run the address validation mapping.
Name of Address List	Name of the address record set that you submit to Australia Post.
List Processor Name	Name of the organization that submits the address record set.
Name of List Manager/Owner	Name of the manager or owner of the address data in the organization.
Phone Number	Contact telephone number of the organization that submits the address record set.
Address	Address of the organization that submits the address record set.

RELATED TOPICS:

- [“Defining a Certification Report” on page 126](#)

CASS Report Fields

When you configure a CASS report, you provide information about the organization that submits the certified address record set to the USPS. Save or print the report, and include the report with the address records that you submit to the USPS.

Use the **Reports** view to enter the information.

The following table describes the information that you enter:

Field	Description
Report File Name	Name and location of the report that address validation creates. By default, the Address Validator creates the report in the <code>bin</code> directory of the Data Integration Service machine. To write the report file to another location on the Data Integration Service machine, enter the file path and file name. You can enter a fully qualified path or a relative path. The relative path uses the <code>bin</code> directory as a root directory. The directory that you specify must exist before you run the address validation mapping.
List Name/ID	Name or identification number of the address list that you submit to the mail carrier.
List Processor Name	Name of the organization that performs the address validation.
Name/Address	Postal name and address of the organization that performs the address validation.

Note: The Address Validator transformation reads metadata about the CASS reference data files from the batch and interactive reference data file for the United States. To generate a CASS report, the transformation must read the current CASS reference data files and the current batch and interactive reference data file.

RELATED TOPICS:

- [“Defining a Certification Report” on page 126](#)

SendRight Report

When you configure a SendRight report, you provide information about the organization that submits the certified address record set to New Zealand Post. Save or print the report, and include the report with the address records that you submit to New Zealand Post.

Use the **Reports** view to enter the information.

The following table describes the information that you enter:

Field	Description
Customer Name	The name of the organization that submits the address record set.
Customer NZP Number	The New Zealand Post account number of the organization that submits the address record set. If a mailhouse submits the records on behalf of the organization, enter the mailhouse transport identification (TPID) number.

Field	Description
Customer Database	The name of the file that contains the address record set. The Address Validator transformation creates the report in the location that you specify on the Content Management Service. Use the Administrator tool to set the location.
Customer Address	The address of the organization that submits the address record set.

RELATED TOPICS:

- [“Defining a Certification Report” on page 126](#)

SERP Report Fields

When you configure a SERP report, you provide information about the organization that submits the certified address record set to Canada Post. Save or print the report, and include the report with the address records that you submit to Canada Post.

Use the **Reports** view to enter the information.

The following table describes the information that you enter:

Field	Description
Report File Name	Name and location of the report that address validation creates. By default, the Address Validator creates the report in the <code>bin</code> directory of the Data Integration Service machine. To write the report file to another location on the Data Integration Service machine, enter the file path and file name. You can enter a fully qualified path or a relative path. The relative path uses the <code>bin</code> directory as a root directory. The directory that you specify must exist before you run the address validation mapping.
Customer CPC Number	Customer number issued by the Canada Post Corporation to the organization that performs the address validation.
Customer Name/ Address	Name and address of the organization that performs the address validation.

RELATED TOPICS:

- [“Defining a Certification Report” on page 126](#)

Configuring an Address Validator Transformation

Use an Address Validator transformation to validate and improve your postal address data quality.

The Address Validator transformation reads address reference data. Verify that the Developer tool can access the address reference data files you need.

1. Open the transformation.
2. Click the **General Settings** view and configure the general properties.

3. Click the **Templates** view to add input and output ports.
4. Click the **Reports** view to generate reports for postal service address certification.
5. Click the **Advanced** view to configure advanced address validation properties.
6. Connect the input and output ports.

Note: Connect input ports that you do not want the Address Transformation to validate to the **Passthrough** input port group.

Adding Ports to the Address Validator Transformation

Use the **Templates** view to add ports to the Address Validator transformation.

1. Click the **Templates** view.
2. Expand a template.
 - Choose the **Basic Model** template to add common address fields.
 - Choose the **Advanced Model** template to add specialized address fields.
3. Expand the input port group that corresponds to the format of your input data. The input port groups are **Discrete**, **Multiline**, and **Hybrid**.
4. Select input ports.

Tip: Click the **CTRL** key to select multiple ports.
5. Right-click the ports and select **Add port to transformation**.
6. Expand the output port group that contains the fields you require.
7. Right-click the ports and select **Add port to transformation**.
8. To add passthrough ports for columns you do not want to validate, click the **Ports** tab, select the **Passthrough** input port group, and click **New**.

Creating User-Defined Templates

Create templates to group the address ports that you plan to reuse.

You create custom templates by selecting ports from the Basic and Advanced templates. You can select the custom templates when you create subsequent Address Validator transformations.

Note: Templates are not repository objects. Templates reside on the machine you use to create them.

1. Select the **Templates** view.
2. Click **New**.
3. Type a name for the template.
4. Expand the **Basic Model** or **Advanced Model** template and select the ports you require.
5. Click **OK**.

Defining Address Validator Models

Address Validator models define the default input and output ports for Address Validator transformations.

Address Validator transformations do not contain default input and output ports. However, you can define a model to specify the input and output ports that Address Validator transformations use.

Note: Models are not repository objects. Models reside on the machine you use to create them.

To define an Address Validator model, you perform the following steps:

1. Select the **Templates** view.
2. Expand the **Basic Model** or **Advanced Model** template and select the ports you require.
3. Select **Create default AV model using selected ports**.
4. To reset the model and remove all ports, select **Clear default AV model**.

Defining a Certification Report

When you define a certification report in the Address Validator transformation, you configure options on the **General Settings** and **Reports** views.

1. On the **General Settings** view, set the **Mode** option to *Certified*.
2. On the **Reports** view, select the type of reports to generate. You can select the following types of report:

Option	Description
AMAS Report	Contains information that Australia Post requires to process record set.
CASS Report	Contains information that the USPS requires to process the record set.
SendRight Report	Contains information that New Zealand Post requires to process the record set.
SERP Report	Contains information that Canada Post requires to process the record set.

3. For each report type that you select, enter the report details.

Submit the report file to the mail carrier with the list of address records that you validated with the Address Validator transformation.

RELATED TOPICS:

- [“AMAS Report Fields” on page 122](#)
- [“CASS Report Fields” on page 123](#)
- [“SendRight Report” on page 123](#)
- [“SERP Report Fields” on page 124](#)

Address Validator Transformation in a Non-native Environment

The Address Validator transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Address Validator Transformation on the Blaze Engine

The Address Validator transformation is supported with the following restrictions:

- The Address Validator transformation cannot generate a certification report.
- The Address Validator transformation fails validation when it is configured to run in Interactive mode or Suggestion List mode.

Address Validator Transformation on the Spark Engine

The Address Validator transformation is supported with the following restrictions:

- The Address Validator transformation cannot generate a certification report.
- The Address Validator transformation fails validation when it is configured to run in Interactive mode or Suggestion List mode.
- A mapping that runs with an Address Validator transformation ignores the value on the **Execution Instances** advanced property on the transformation.

Address Validator Transformation in a Streaming Mapping

Streaming mappings have the same processing rules as batch mappings on the Spark engine.

Address Validator Transformation on the Databricks Spark Engine

The Address Validator transformation is supported with the following restrictions:

- The Address Validator transformation cannot generate a certification report.
- The Address Validator transformation fails validation when it is configured to run in Interactive mode or Suggestion List mode.
- A mapping that runs with an Address Validator transformation ignores the value on the **Execution Instances** advanced property on the transformation.

CHAPTER 5

Aggregator Transformation

This chapter includes the following topics:

- [Aggregator Transformation Overview, 128](#)
- [Aggregator Transformations in Dynamic Mappings, 129](#)
- [Developing an Aggregator Transformation, 129](#)
- [Aggregator Transformation Ports, 129](#)
- [Aggregate Expressions, 130](#)
- [Group By Ports, 132](#)
- [Aggregator Caches, 135](#)
- [Sorted Input for an Aggregator Transformation, 135](#)
- [Aggregator Transformation Advanced Properties, 137](#)
- [Creating a Reusable Aggregator Transformation, 137](#)
- [Creating a Non-Reusable Aggregator Transformation, 138](#)
- [Tips for Aggregator Transformations, 138](#)
- [Troubleshooting Aggregator Transformations, 139](#)
- [Aggregator Transformation in a Non-native Environment, 139](#)

Aggregator Transformation Overview

Configure an Aggregator transformation to perform aggregate calculations, such as averages and sums, against groups of data. You can use an Aggregator transformation to remove duplicate rows. The Aggregator transformation is an active transformation.

The Aggregator transformation is different from the Expression transformation because you can configure the Aggregator transformation to perform calculations on groups of data. An Expression transformation returns a result on a row by row basis.

For example, you can calculate the average salary for employees in each department of a organization. Configure a group by department number. Configure an expression to calculate the average salary and to return the result for each unique department number.

Use the transformation language to create aggregate expressions.

The Data Integration Service performs aggregate calculations as it reads data and stores the data in an aggregate cache. You can sort the input data to increase performance. The Data Integration Service does not create the cache if you sort the input data.

Aggregator Transformations in Dynamic Mappings

You can use an Aggregator transformation in a dynamic mapping. You can configure dynamic ports in the transformation and reference the generated ports.

You can perform the following tasks to configure an Aggregator transformation in a dynamic mapping:

Reference a dynamic port or a generated port as a Group By column.

You can include dynamic ports or generated ports as Group By columns. If you specify a dynamic port as a Group By column, you are specifying that all generated ports in the dynamic port are Group By columns. For this reason, you cannot specify a generated port as a Group By column if you specified the parent dynamic port as a Group By column. If you reference a generated port, and the generated port does not exist at run time, the mapping fails. You can parameterize the Group By columns to indicate which columns to group at run time.

Reference a generated port in the aggregate expression.

You can include a generated port in the aggregate expression. If the port does not exist at run time, the mapping fails. You cannot reference a dynamic port in the aggregate expression.

Create the aggregate expression in an output port.

You cannot create an aggregate expression in a generated port or a dynamic port. The aggregate expression cannot be a dynamic expression.

Parameterize values in the aggregate expression.

You can include parameters in the aggregate expression. However, You cannot use an expression parameter or a port parameter in the aggregate expression.

Developing an Aggregator Transformation

When you create an Aggregator transformation, define the expression to run against each row. Define a group by port list to return results by.

To create an Aggregator transformation, perform the following steps:

1. Define the transformation and create the ports.
2. Configure an aggregate expression on a variable or output port.
3. Define a group of ports to return aggregate results by.

Aggregator Transformation Ports

An Aggregator transformation has multiple port types that you can use to configure groups and to aggregate expressions.

An Aggregator transformation Ports view has the following fields:

Name

Name of the port.

Type

The port data type.

Precision

Length of the field.

Scale

The number of positions to the right of the decimal for numeric data.

Input

Indicates that data is from an upstream transformation.

Output

Indicates that the port returns the value of an expression. The expression can include non-aggregate expressions and conditional clauses. You can create multiple aggregate output ports.

Pass-Through

Indicates that the port is an input-output port that returns the data unchanged.

Variable

Indicates that the port can store values or calculations to use in an expression. Variable ports cannot be input or output ports. They pass data within the transformation only.

Expression

The expression to aggregate the rows or groups of rows.

Default value

The default value for a port with invalid or null values.

Input Rules

A set of rules that filter the ports to include or exclude in the transformation based on port names or data type. Configure input rules when you define dynamic ports.

Aggregate Expressions

Configure aggregate expressions in the variable ports or the output ports of an Aggregator transformation. You cannot enter an expression if the port is an input port, a pass-through port, or a dynamic port.

An aggregate expression can include conditional clauses and functions that are not aggregate functions. The aggregate function can also include an aggregate function nested within another aggregate function, such as:

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

The aggregate expression result varies based on the group by ports in the transformation. For example, the following aggregate expression finds the total quantity of items sold:

```
SUM( QUANTITY )
```

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

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

Aggregate Functions

Configure aggregate functions within an Aggregator transformation. You can nest one aggregate function within another aggregate function.

The transformation language includes the following aggregate functions:

- ANY
- AVG
- COLLECT_LIST
- COLLECT_MAP
- COUNT
- FIRST
- LAST
- MAX (Date)
- MAX (Number)
- MAX (String)
- MEDIAN
- MIN (Date)
- MIN (Number)
- MIN (String)
- PERCENTILE
- STDDEV
- SUM
- VARIANCE

If you use a port in an expression in the Aggregator transformation but you do not use the port within an aggregate function, the Data Integration Service uses the last row in the port to process the expression.

For example, you create an Aggregator transformation that contains the ports `COMMISSIONS` and `SALARY`. The port `SALARY` is a group-by port.

You might use the following expression in an output port:

```
SUM(COMMISSIONS)
```

The Data Integration Service processes the Aggregator function and returns the sum of the values in the port `COMMISSIONS` in the output port.

You might modify the expression to the following expression:

```
SUM(COMMISSIONS) + COMMISSIONS
```

To process the expression, the Data Integration Service returns the sum of the values in the port `COMMISSIONS` and adds the value of the last row in the port `COMMISSIONS` to the return value in the output port.

For a different output port, you might use the following expression:

```
SUM(COMMISSIONS) + SALARY
```

To process the expression, the Data Integration Service returns the sum of the values in the port `COMMISSIONS` and adds the value in the last row of the port `SALARY` to the return value in the output port. Note that the values in each row of the port `SALARY` are the same because the `SALARY` port is a group-by port.

Nested Aggregate Functions

You can include multiple single-level or multiple nested functions in different output ports in an Aggregator transformation.

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 Developer tool marks the mapping or mapplet invalid. If you need to create both single-level and nested functions, create separate Aggregator transformations.

Conditional Clauses in Aggregate Expressions

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 )
```

Group By Ports

You can define groups of rows to aggregate instead of running an aggregation across all the input data. For example, you can calculate the total company sales or you can find the total sales grouped by region.

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

When you group values, the Data Integration Service produces one row for each group. If you do not group values, the Data Integration Service returns one row for all input rows. The Data Integration Service returns the last row of each group with the result of the aggregation. You can specify to return a specific row. For example, if you use the FIRST aggregator function, the Data Integration Service returns the first row.

When you select multiple group by ports in the Aggregator transformation, the Data Integration Service uses a port order to determine the order by which it groups. The group order can affect the results. Order the group by ports to ensure the appropriate grouping. You can change the port order after you select the ports in the group/

For example, you might create an output port called Price_Out. The expression for Price_Out is SUM (Qty * Price). You define Store_ID and Item as the group by ports. The transformation returns the total price for each item by store.

The input rows might contain the following data:

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

Store_ID	Item	Qty	Price
201	battery	4	1.59
301	battery	1	2.45

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

Store_Id	Item
101	battery
101	AAA
201	battery
301	battery

The Data Integration Service returns the Store_ID, Item, Qty, and Price from the last row with the sum of (Price * Qty) for each item by store :

Store_ID	Item	Qty	Price	Price_Out
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

Configure Group By Ports

Define the group by ports on the **Group By** tab of the transformation **Properties** view.

The following image shows the Group By tab:

The Group By tab contains the following options:

Specify by

Select **Value** or **Parameter**. Select **Value** to use port names. Choose **Parameter** to use a port list parameter.

Add

Accepts a port name that you type in manually. You must type a valid port name before you click **Add**.

Choose

Click **Choose** to select ports to add to the group. The Developer tool provides a list of ports from the transformation to choose from.

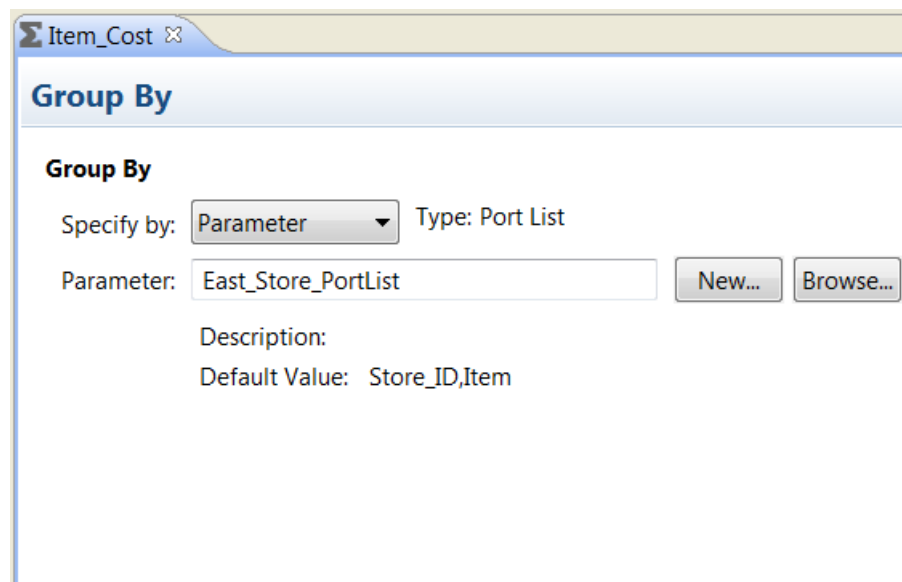
Move Up and Move Down

You can change the order of the ports in the group. Select the port name and then click one of the move buttons to move it up or down in the sort order.

Group By Parameters

You can configure a port list parameter that contains one or more ports to include in the group. Create a port list parameter by selecting ports from a list of the ports in the transformation.

The following image shows the **Group By** tab when you use a parameter to identify the ports in the group:



You can browse for a port list parameter or click **New** to create a port list parameter. If you choose to create a port list parameter, you can select the ports from a list of the ports in the transformation.

Default Values of Group By Ports

The Data Integration Service does not create a group when a group by port contains null values. You can define a default value for each port in the group to replace any null input value. Then, the Data Integration Service can include the rows in the aggregation totals.

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 group by output port, named CORRECTED_ITEM, using the following expression:

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

Aggregator Caches

When you run a mapping that uses an Aggregator transformation, the Data Integration Service creates an index cache and data cache in memory to run the transformation. If the Data Integration Service requires more space than available in the memory cache, it stores overflow data in cache files.

The Data Integration Service creates the following caches for the Aggregator transformation:

- Index cache that stores group values as configured in the group by ports.
- Data cache that stores calculations based on the group by ports.

The Data Integration Service does not use cache memory to run an Aggregator transformation with sorted ports. You do not need to configure cache memory for Aggregator transformations that use sorted ports.

Sorted Input for an Aggregator Transformation

You can increase Aggregator transformation performance with the sorted input option.

When you use sorted input, the Data Integration Service assumes all data is sorted by group and it performs aggregate calculations as it reads rows for a group. When required, the Data Integration Service stores group information in memory. To use the Sorted Input option, you must pass sorted data to the Aggregator transformation. If you use sorted input, the Aggregator transformation provides sorted output.

When you do not use sorted input, the Data Integration Service performs aggregate calculations as it reads. Because the data is not sorted, the Data Integration Service stores data for each group until it reads the entire source to ensure that 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 Data Integration Service performs an aggregation for the three rows in the 101/battery group when it finds the 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, the Data Integration Service fails the mapping run.

Sorted Input Conditions

Certain conditions might prevent you from using sorted input.

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

- The aggregate expression contains nested aggregate functions.
- Source data is data driven.

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

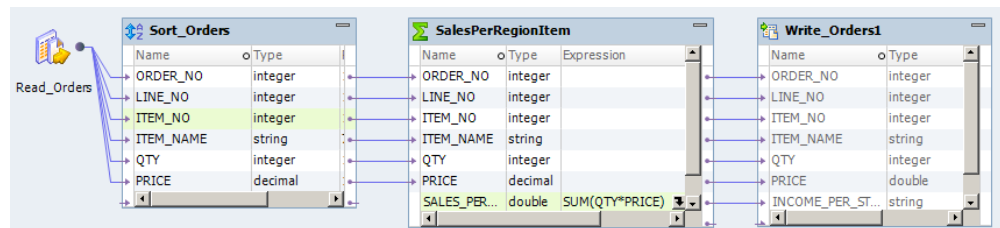
Sorting Data in an Aggregator Transformation

To use sorted input, you pass sorted data through an Aggregator transformation.

You must sort data by the Aggregator group by ports in the order they appear in the Aggregator transformation.

For relational and flat file input, 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 before 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.

The following mapping shows a Sorter transformation configured to sort the source data in ascending order by ITEM_NO:



The Sorter transformation sorts the data as follows:

ITEM_NO	ITEM_NAME	QTY	PRICE
345	Soup	4	2.95
345	Soup	1	2.95
345	Soup	2	3.25
546	Cereal	1	4.49
546	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

Aggregator Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Aggregator transformation.

Configure the following advanced properties for an Aggregator transformation:

Cache Directory

Directory where the Data Integration Service creates the index cache files and data cache files. Verify that the directory exists and contains enough disk space for the cache files.

Enter multiple directories separated by semicolons to increase performance during cache partitioning. Cache partitioning creates a separate cache for each partition that processes the transformation.

Default is the CacheDir system parameter. You can configure another system parameter or user-defined parameter for this property.

Data Cache Size

Amount of memory that the Data Integration Service allocates to the data cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.

Index Cache Size

Amount of memory that the Data Integration Service allocates to the index cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.

Sorted Input

Indicates that input data is presorted by groups. Select this option only if the mapping passes sorted data to the Aggregator transformation.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

RELATED TOPICS:

- [“Cache Size” on page 71](#)

Creating a Reusable Aggregator Transformation

Create a reusable Aggregator transformation to use in multiple mappings or mapplets.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
The **New** dialog box appears.
3. Select the Aggregator transformation.
4. Click **Next**.

5. Enter a name for the transformation.
6. Click **Finish**.
The transformation appears in the editor.
7. Click the **New** button to add a port to the transformation.
8. Edit the port to set the name, datatype, and precision.
9. Determine the type of each port: input, output, passthrough, or variable.
10. Click the Expression field to configure aggregate expressions for output ports. You can select ports and parameters to define the aggregate expression.
11. Click the **Advanced** view and edit the transformation properties.

Creating a Non-Reusable Aggregator Transformation

Create a non-reusable Aggregator transformation in a mapping or mapplet.

1. In a mapping or mapplet, drag an Aggregator transformation from the Transformation palette to the editor.
The transformation appears in the editor.
2. In the **Properties** view, edit the transformation name and the description.
3. On the **Ports** tab, click the **New** button to add ports to the transformation.
4. Edit the ports to set the name, datatype, and precision.
5. Determine the type of each port: input, output, passthrough, or variable.
6. Configure aggregate expressions for output ports.
7. In the **Advanced** view, edit transformation properties.

Tips for Aggregator Transformations

You can use tips to use Aggregator transformations more effectively.

Use sorted input to decrease the use of aggregate caches.

Sorted input reduces the amount of data cached during mapping run and improves 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 the data before aggregating it.

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

Only a sorted Aggregator transformation provides sorted output.

If you use unsorted input and you want to produce sorted output, use a Sorter transformation after the Aggregator transformation.

Troubleshooting Aggregator Transformations

You can troubleshoot Aggregator transformations.

[I selected sorted input but the mapping takes the same amount of time as before.](#)

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

- The aggregate expression contains nested aggregate functions.
- Source data is data driven.

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

[A mapping with an Aggregator transformation causes slow performance.](#)

The Data Integration Service may be paging to disk. You can increase performance by increasing the index and data cache sizes in the transformation properties.

Aggregator Transformation in a Non-native Environment

The Aggregator transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Aggregator Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Aggregate Functions

If you use a port in an expression in the Aggregator transformation but you do not use the port within an aggregate function, the run-time engine might use any row in the port to process the expression.

The row that the run-time engine uses might not be the last row in the port. Processing is distributed, and thus the run-time engine might not be able to determine the last row in the port.

Data Cache Optimization

The data cache for the Aggregator transformation is optimized to use variable length to store binary and string data types that pass through the Aggregator transformation. The optimization is enabled for record sizes up to 8 MB. If the record size is greater than 8 MB, variable length optimization is disabled.

When variable length is used to store data that passes through the Aggregator transformation in the data cache, the Aggregator transformation is optimized to use sorted input and a pass-through Sorter transformation is inserted before the Aggregator transformation in the run-time mapping.

To view the Sorter transformation, view the optimized mapping or view the execution plan in the Blaze validation environment.

During data cache optimization, the data cache and the index cache for the Aggregator transformation are set to Auto. The sorter cache for the Sorter transformation is set to the same size as the data cache for the Aggregator transformation. To configure the sorter cache, you must configure the size of the data cache for the Aggregator transformation.

Aggregator Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Aggregate Functions

If you use a port in an expression in the Aggregator transformation but you do not use the port within an aggregate function, the run-time engine might use any row in the port to process the expression.

The row that the run-time engine uses might not be the last row in the port. Processing is distributed, and thus the run-time engine might not be able to determine the last row in the port.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

Aggregator Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- A streaming pipeline contains more than one Aggregator transformation.
- A streaming pipeline contains an Aggregator transformation and a Rank transformation.
- An Aggregator transformation is upstream from a Lookup transformation.
- An Aggregator transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.

Aggregator Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Aggregate Functions

If you use a port in an expression in the Aggregator transformation but you do not use the port within an aggregate function, the run-time engine might use any row in the port to process the expression.

The row that the run-time engine uses might not be the last row in the port. Processing is distributed, and thus the run-time engine might not be able to determine the last row in the port.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

CHAPTER 6

Association Transformation

This chapter includes the following topics:

- [Association Transformation Overview, 142](#)
- [Memory Allocation, 143](#)
- [Association Transformation Advanced Properties, 144](#)

Association Transformation Overview

The Association transformation processes output data from a Match transformation. It creates links between duplicate records that are assigned to different match clusters, so that these records can be associated together in data consolidation and master data management operations.

The Association transformation generates an **AssociationID** value for each row in a group of associated records and writes the ID values to an output port.

The Consolidation transformation reads the output from the Association transformation. Use a Consolidation transformation to create a master record based on records with common association ID values.

The Association transformation accepts string and numerical data values on input ports. If you add an input port of another data type, the transformation converts the port data values to strings.

The AssociationID output port writes integer data. The transformation can write string data on an AssociationID port if the transformation was configured in an earlier version of Informatica Data Quality.

Example: Associating Match Transformation Outputs

The following table contains three records that could identify the same individual:

ID	Name	Address	City	State	ZIP	SSN
1	David Jones	100 Admiral Ave.	New York	NY	10547	987-65-4321
2	Dennis Jones	1000 Alberta Ave.	New Jersey	NY	-	987-65-4321
3	D. Jones	Admiral Ave.	New York	NY	10547-1521	-

A duplicate analysis operation defined in a Match transformation does not identify all three records as duplicates of each other, for the following reasons:

- If you define a duplicate search on name and address data, records 1 and 3 are identified as duplicates but record 2 is omitted.

- If you define a duplicate search on name and Social Security number data, records 1 and 2 are identified as duplicates but record 3 is omitted.
- If you define a duplicate search on all three attributes (name, address, and Social Security number), the Match transformation may identify none of the records as matches.

The Association transformation links data from different match clusters, so that records that share a cluster ID are given a common AssociationID value. In this example, all three records are given the same AssociationID, as shown in the following table:

ID	Name	Address	City	State	Zip	SSN	Name and Address Cluster ID	Name and SSN Cluster ID	Association ID
1	David Jones	100 Admiral Ave.	New York	NY	10547	987-65-4320	1	1	1
2	Dennis Jones	1000 Alberta Ave.	New Jersey	NY	-	987-65-4320	2	1	1
3	D. Jones	Alberta Ave.	New York	NY	10547-1521	-	1	2	1

You can consolidate the duplicate record data in the Consolidation transformation.

Memory Allocation

You can set the minimum amount of cache memory that the Association transformation uses. The default setting is 400,000 bytes.

Set the value in the **Cache File Size** property on the **Advanced** tab.

The default value represents the minimum amount of memory that the transformation uses. The Association transformation tries to obtain a multiple of the default value, depending on the number of ports you associate on. The transformation uses this formula to obtain cache memory:

$(\text{Number of association ports} + 1) \times \text{default cache memory}$

For example, if you configure seven association ports, the transformation attempts to allocate 3.2 million bytes, or 3.05 MB, to cache memory.

If you change the default setting, the transformation does not try to obtain additional memory.

Note: If you enter a cache memory value that is lower than 65536, the Association transformation reads the value in megabytes.

Association Transformation Advanced Properties

The Association transformation contains advanced properties that determine the cache memory behavior and the tracing level.

You can configure the following advanced properties:

Cache File Directory

Specifies the directory to which the Data Integration Service writes temporary data for the current transformation. The Data Integration Service writes temporary files to the directory when the volume of input data is greater than the available system memory. The Data Integration Service deletes the temporary files after the mapping runs.

You can enter a directory path on the property, or you can use a parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory. The default value is the CacheDir system parameter.

Cache File Size

Determines the amount of system memory that the Data Integration Service uses to sort the input data on the transformation.

Before it sorts the data, the Data Integration Service allocates the amount of memory that you specify. If the sort operation generates a greater amount of data, the Data Integration Service writes the excess data to the cache file directory. If the sort operation requires more memory than the system memory and the file storage can provide, the mapping fails.

The transformation reads the value in bytes. The default value is 400,000 bytes. The maximum value is 2,147,483,647 bytes. You can use a parameter to specify the cache file size.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

CHAPTER 7

Bad Record Exception Transformation

This chapter includes the following topics:

- [Bad Record Exception Transformation Overview, 145](#)
- [Bad Record Exception Output Record Types, 146](#)
- [Bad Record Exception Management Process Flow, 147](#)
- [Bad Record Exception Mappings, 147](#)
- [Bad Record Exception Ports , 149](#)
- [Bad Record Exception Configuration View, 151](#)
- [Bad Record Exception Issue Assignment , 153](#)
- [Exception Transformation Advanced Properties, 154](#)
- [Configuring a Bad Record Exception Transformation, 154](#)
- [Bad Record Exception Mapping Example, 155](#)

Bad Record Exception Transformation Overview

The Bad Record Exception transformation is an active transformation that reads the output of a data quality process and identifies records that require manual review. The Bad Record Exception transformation is a multiple-group transformation.

Configure a Bad Record Exception transformation to analyze the output of a process that identifies data quality issues in records. A record that has a data quality issue that needs further review is an exception.

The Bad Record Exception transformation receives input from another transformation or from a data object in another mapping. The input to the Bad Record transformation must contain one or more quality issue ports that receive text descriptions of data quality problems. The input to the Bad Record Exception transformation can also contain a numeric record score that the transformation can use to determine the data quality of each record. Set an upper and lower score threshold in the Exception transformation to classify records as good and bad quality based on the record score. The Bad Record Exception transformation writes exceptions and associated quality issue text to a Bad Records table.

For example, an organization needs to validate customer addresses before sending out some mail to the customers. A developer creates a mapping that validates customer city, state, and ZIP code against reference tables with a Labeler transformation. The Labeler transformation validates the fields and adds a record score to each row based on the results. The Labeler transformation also adds text that describes the quality issues for each record that has an error. The Labeler transformation adds quality issue text such as

city not valid, or ZIP code blank to each exception. The Bad Record Exception transformation writes customer records that need manual review to the Bad Records table. Data analysts review and correct the bad records in the Analyst tool.

Bad Record Exception Output Record Types

The Bad Record Exception examines input record scores to determine record quality. It returns records to different output groups

The Exception transformation identifies the following types of records based on each record score:

Good records

Records with scores greater than or equal to the upper threshold. Good records are valid and do not require review. For example, if configure the upper threshold as 90, any record with a 90 or higher does not need review.

Bad records

Records with scores less than the upper threshold and scores greater than or equal to the lower threshold. Bad records are the exceptions that you need to review in the Analyst tool. For example, when the lower threshold is 40, any record with a score from 40 to 90 needs manual review.

Rejected records

Records with scores less than the lower threshold. Rejected records are not valid. By default, the Exception transformation drops rejected records from the data flow. For this example, any record with a score 40 or less is a rejected record.

Note: If the quality issues fields are NULL, the record is not an exception. When any quality issue contains text or it contains an empty string, the record is an exception. Verify that a quality issue port contains null values when a field has no error. If the quality issue ports contain blanks instead of null values then the Exception transformation flags every record as an exception. When a user needs to correct the issues in the Analyst tool, the user cannot filter the exceptions by data quality issue.

When a record has a score less than zero or greater than 100, the row is not valid. The Data Integration Service logs an error message that the row is not valid and it skips processing the record.

If you do not connect a record score as input to the Exception transformation, the transformation writes all records that contain quality issues to the bad records table.

When you include the Bad Record Exception transformation in a Mapping task, you can configure a Human task in the same workflow to include a manual review of the exceptions. The Human task starts when a Mapping task ends in the workflow. The Human task requires users to access the Analyst tool to resolve the quality issues. A user can update the data and change the quality status of each record in the bad records table.

Bad Record Exception Management Process Flow

The Exception transformation receives record scores from data quality transformations and creates tables that contain records with different levels of data quality. You must configure the data quality transformations to find quality issues and provide a record score for each row.

You can configure the data quality transformations in a single mapping, or you can create mappings for different stages of the data quality process.

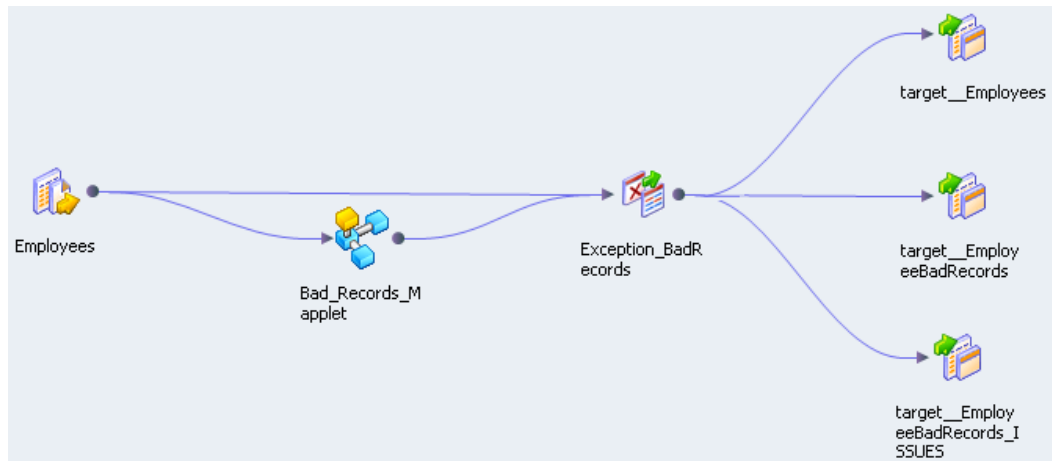
Complete the following bad record exception management tasks:

1. In the Developer tool, define transformations that generate score values for source data based on data quality issues that you define. Define transformations that return text to describe the quality of the source data. For example, you can configure a Labeler transformation to check source data against reference tables and then write a descriptive label for each comparison. You can define an IF/THEN rule in a Decision transformation to examine a data field. You can define multiple transformations and maplets that perform different data quality operations.
2. Configure an Exception transformation to analyze the record scores that it receives from the data quality operations. Configure the transformation to write records to database tables based on score values in the records. You can create separate tables for good records, bad records, quality issues, and rejected records.
3. Assign a quality issue port to each input port that might contain bad data.
4. Optionally, configure target data objects for good and bad records. Connect the Exception transformation output ports to the target data objects in the mapping.
5. Create the target data object for bad records. Choose to generate a bad records table and add it to the mapping. When you generate a bad records table, the Developer tool also generates a quality issues table. Add the quality issues table to the mapping.
6. Add the mapping to a workflow.
7. Configure a Human task to assign manual review of bad records to users. Users can review and update the bad records in the Analyst tool.

Bad Record Exception Mappings

When you create a mapping that identifies bad record exceptions, you configure the mapping to write records to one or more database targets based on the quality of the data in the records.

The following figure shows an example Bad Record Exception mapping:



The mapping contains the following objects:

Data source

An Employees data source that contains the records to analyze for data quality.

Maplet

The Bad_Records_Maplet contains transformations that check for and add quality issues and record scores to source records. Rules are transformations that analyze the data and find the quality issues. For example, you can include a Labeler transformation to compare input data to reference tables. Depending on the results, you can configure the Labeler transformation to return quality issues as additional columns in the rows. You can configure a Decision transformation that uses IF, THEN, ELSE statements to examine the data and apply quality issues and record scores to the input data.

Exception transformation

The Exception transformation determines which records to write to the data targets including the bad records table and the issues table.

Good record table

The Exception transformation writes all good-quality records to the target_Employees table.

Bad Record table

The Exception transformation writes all bad-quality records to the target_EmployeeBadRecords table. Bad records require manual review.

Issues table

The Exception transformation writes quality issues to the target_EmployeeBadRecords_ISSUES table. When you view the bad records in the Analyst tool, the user interface links the quality issues to the bad records.

Optionally, the Exception transformation can write rejected records to a rejected records table. You must choose to create a separate output group for rejected records on the **Configuration** view of the transformation.

Bad Record Exception Quality Issues

Quality issues are text strings that describe the type of data quality problem that caused a low record score. The Bad Record Exception transformation receives quality issues associated with each source row that

contains a low record score. You can configure different types of transformations that determine quality issues and record scores.

For example, you might create a Decision transformation that examines the phone number. The Decision transformation generates the record score and the quality issues for phone number.

The following decision strategy identifies phone numbers of incorrect length in a Decision transformation:

```
IF LENGTH(Phone_Number) > 10 THEN
  Score:=50
  Phone_Quality_Issue:='Phone num too long'
ELSEIF LENGTH(Phone_Number) < 10 THEN
  Score:=50
  Phone_Quality_Issue:=' Phone num too short'
ELSE
  Score:=90
ENDIF
```

When you configure the Exception transformation, you must associate Phone_Quality_Issue with the Phone_Number port. The ports are from different input groups.

The Exception transformation reads the scores generated by the Decision transformation and assigns records with a score of "50" to the bad records group of output ports. It writes the Phone_Quality_Issue to the Issues group of output ports.

Human Tasks

When you configure a workflow that contains an Exception transformation, you include the mapping in a Mapping task. You add a Human task to the same workflow. The Human task requires one or more users to correct the exception records in the Analyst tool.

The Mapping task identifies the records in the source data that contain unresolved data quality issues. Data analysts use the Analyst tool to resolve the issues and to update the data quality status of each record.

When you configure a Human task, you create one or more task instances and one or more task steps. A task instance represents the data set that a user must work on. A task step represents the type of work that a user performs on the records in his or her task instance. You can create multiple task instances so that different users work on different parts of the data in the Analyst tool.

A user can update the status of the bad records in the Analyst tool in one of the following ways:

- If a record is valid, the user updates the table metadata to confirm the record for persistent storage in the database.
- If a record is not valid, the user updates the table metadata to remove the record from the database at a later stage in the workflow.
- If the record status is not confirmed, the user updates the table metadata so that the record returns to the workflow for further processing in a Mapping task.

For more information about Human tasks, see the *Informatica Developer Workflow Guide*.

Bad Record Exception Ports

Configure the input and output ports on the **Ports** tab of the Bad Record Exception transformation.

The Bad Record Exception transformation contains groups of input and output ports.

The following figure shows the **Ports** tab:

Ports							
Name	Type	Precisi...	Scale	Input	Output	Default	Description
Inputs							
Data (3)							
1	EmployeeID	decimal	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	Employee_Name	decimal	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3	Phone_Number	string	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Quality Issues (3)							
1	EmployeeID_Quality	decimal	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	Name_Quality_Issue	decimal	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3	Phone_Quality_Issue	decimal	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Control (1)							
1	Score	double	15	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Outputs							
Standard Output (4)							
1	Score	double	15	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
2	EmployeeID	decimal	30	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
3	Employee_Name	decimal	30	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
4	Phone_Number	string	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Bad Records (6)							
1	Workflow_ID	string	64	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
2	Row_Identifier	bigint	19	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
3	Record_Status	string	20	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
4	EmployeeID	decimal	30	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
5	Employee_Name	decimal	30	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Bad Record Exception Transformation Input Ports

A Bad Record Exception transformation has separate input groups for the data, the quality issues, and the record score.

The Bad Record Exception transformation has the following input groups:

Data

The source data fields.

Quality Issues

Contains ports that describe record quality problems. Quality issue ports contain strings such as "Excess_Characters" or "Bad_Data_Format." You can have multiple quality issues in each record. The transformation does not associate ports in the Quality Issues group to source data fields until you assign the issues to data ports in the **Issue Assignment** view.

Control

The record score. The Exception transformation analyzes the record score to determine if input rows are exceptions. If you do not connect the Score port, the Exception transformation identifies a row as an exception if a Quality Issues port contains data.

Bad Record Exception Transformation Output

A Bad Record Exception transformation has multiple output groups.

The Bad Record Exception transformation has the following output groups:

Standard Output

The good quality records that you do not need to examine for data quality issues.

Each record in the Standard Output group contains a Score port that represents the data quality of the record.

Upper Threshold

The upper limit for the bad record score range. The transformation identifies records with scores that are greater than or equal to the upper threshold as good records.

Data Routing Options

The types of output records. In the default configuration, the transformation writes good records to the Standard Output and writes bad records to the Bad Records table. By default, the transformation does not write the rejected records to a database table.

Standard Output

The types of record that the transformation writes to the standard output ports. Default is Good Records.

Bad Record Table

The types of record that the transformation writes to the bad record output ports. Default is Bad Records.

Create separate output group for rejected records

Creates a separate output group for the rejected records. The option is clear by default.

Generate bad records table

Creates a database table to contain the bad record data. When you select the option, the Exception transformation creates the database table, adds a data object to the Model repository, and adds an instance of the object to the mapping canvas. You can generate the bad records table for an Exception transformation instance in a mapping. When you generate the bad records table, the Developer tool also creates an issues table to store descriptive metadata about the records.

Note: The Developer tool adds a 12-character suffix to each column name in the bad record tables. If you use an Oracle database, the source column name can contain no more than 18 characters.

Generating the Bad Records Table and the Issues Table

When you add the transformation to a mapping, you can generate the Bad Records table and the Issues table. The Developer tool adds the tables to the Model repository.

1. Click **Generate bad records table** to generate the table.
The **Create Relational Data Object** dialog box appears.
2. Browse the database connections. Select a connection to the database to contain the table.
3. Enter a name for the Bad Records table. The Developer tool applies the name that you enter to the Bad Records table and to the Issues table.

The Developer tool appends the following string to the Issues table name:

`_ISSUE`

If you connect to an Oracle database, the Bad Records table name must contain no more than 24 characters.

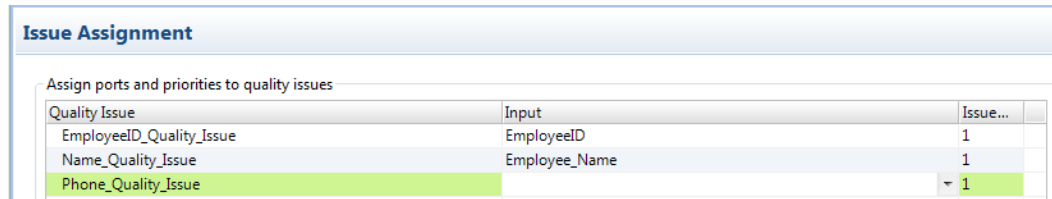
4. Enter a name for the Bad Records data object in the Model repository.
5. Click **Finish**.

The Developer tool adds the tables to the mapping canvas and to the Model repository.

Bad Record Exception Issue Assignment

You must assign ports and priorities to data quality issues.

The following figure shows the **Issue Assignment** view:



The screenshot shows the 'Issue Assignment' view with a table titled 'Assign ports and priorities to quality issues'. The table has three columns: 'Quality Issue', 'Input', and 'Issue...'. There are three rows of data:

Quality Issue	Input	Issue...
EmployeeID_Quality_Issue	EmployeeID	1
Name_Quality_Issue	Employee_Name	1
Phone_Quality_Issue		1

The **Issue Assignment** view contains the following fields:

Quality Issue

Each quality issue port that you defined in the Quality Issues input group appears in the **Quality Issue** column.

Input

The **Input** column contains the data ports that you assign to quality issues in the **Issue Assignment** view. Associate an input port with each quality issue port. Each input port that contains bad quality data must have at least one corresponding quality issue port that indicates a type of issue. You might select the Phone_Number port for the Phone_Quality_Issue, for example. You can assign a port to more than one quality issue.

Issue Priority

Issue priorities determine which quality issues are the most important when you assign the same input port to multiple quality issues. If more than one quality issue occurs for an input field, the Data Integration Service applies the highest priority quality issue. If more than one quality issue exists for an input port and the issues have the same priority, the Data Integration Service applies the topmost quality issue in the list. Enter a priority from 1 to 99, where 1 represents the highest priority.

Define issue priorities to filter the records in the Analyst tool.

Assigning Ports to Quality Issues

Assign a port to associate with each quality issue. The Developer tool creates ports in the Issues output group for each association that you add in the **Issue Assignment** view.

1. For each quality issue, click the **Input** field to display a list of input ports.
2. Select an input port to associate with the quality issue.
You can choose the same port for more than one issue.
3. Click the **Issue** column and select a priority for the quality issue.

Exception Transformation Advanced Properties

Configure properties that determine how the Data Integration Service processes data for the Exception transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Configuring a Bad Record Exception Transformation

When you configure a Bad Record Exception transformation, configure the input ports and the quality issues that can occur in each port. Define the upper and lower thresholds for determining data quality. Configure where to write the exceptions and the reject records.

1. Create a reusable or a nonreusable Bad Record Exception transformation.
 - To create a reusable transformation, select **File > New > Transformation** and select a Bad Record Exception transformation.
 - To create a nonreusable transformation, open a mapping and add an Exception transformation to the mapping canvas. Select a Bad Record Exception transformation from the wizard.
2. Click **Next**, or click **Finish**.

If you click **Next**, you can update the default threshold values and the data routing options before you create the transformation.
3. Configure the input ports.
 - If you create a reusable transformation, select the **Ports** tab, and add ports for the data you want to connect to the transformation.
 - If you create a nonreusable transformation, add other objects to the mapping canvas and drag input ports to the transformation.
4. Select the **Configuration** view.
5. Configure the upper and lower score thresholds.
6. In the **Data Routing Options** section, configure the standard output and exception table properties to set where the transformation writes each type of record.

Configure where to write the Good Records, Bad Records, and Rejected Records. You can write them to standard output or to the Bad Records table.
7. Open the **Issue Assignment** view. Assign data quality issues to the data ports.

Assign a priority to each issue. If a port contains values with multiple issues, the transformation displays the top priority issue.
8. Select the option to generate a bad records table. Enter the database connection and table name information. The table must come from the default schema.

- When you generate a bad records table, you generate a table for the records and an additional table for the data quality issues that relate to the records. The transformation creates a database object in the Model repository.
9. Connect the transformation output ports to one or more data targets. Connect the output ports to the data objects that correspond to the output options you set on the **Configuration** view.
- If you create a reusable transformation, add the transformation to a mapping and connect the output ports.
 - If you create a nonreusable transformation, the transformation connects the ports to the bad record table. You connect output ports to any other data target.

Bad Record Exception Mapping Example

An organization conducts a data project to review new customer data. The organization needs to verify that customer contact data is valid. The following example shows how to define a Bad Record Exception transformation that receives records from a maplet that does a data quality analysis of customer records.

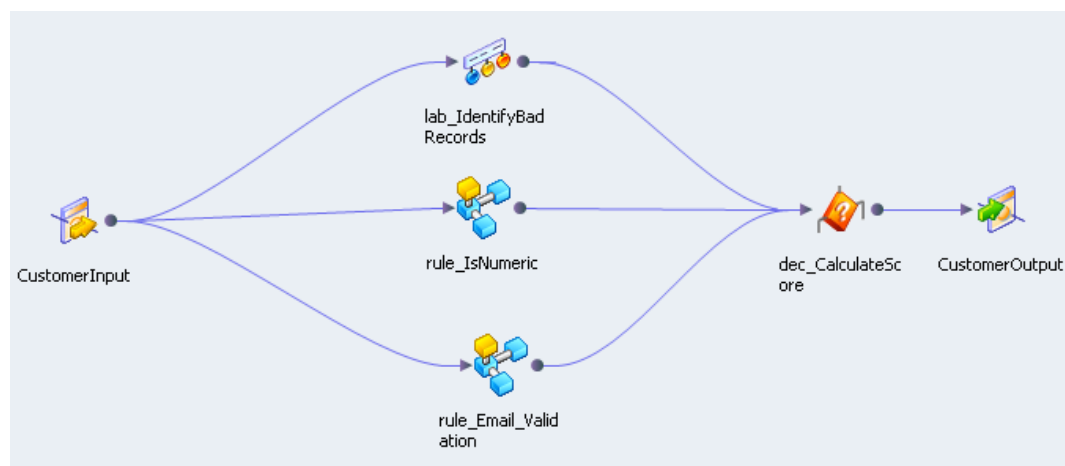
Create a maplet with data quality transformations that evaluate the format and the accuracy of the customer data. The maplet includes transformations that generate a record score based on the results of the data quality analysis. The transformations also define the quality issues for the data based on the results of the analysis.

Bad Record Exception Maplet

Create a maplet that contains data quality transformations to check the values of certain fields. The transformations check reference tables and content sets to determine if the fields in the records are valid. The transformations apply a record score to each record based on the results. The Exception transformation receives the records from the maplet and routes each record to the appropriate output based on the record score.

The maplet consists of Labeler transformations, Decision transformations, and Expression transformations.

The following figure shows the objects in the maplet:



The maplet performs the following tasks:

- A Labeler transformation verifies the locality, the state, the country code, the zip code, and the postal code data that it receives in the input ports. The transformation contains a strategy for each port. The strategies compares the source data to reference tables and identifies values that are not valid.
- An Expression transformation maplet verifies that the phone number is numeric and verifies that the number contains 10 digits.
- A Labeler transformation and an Expression transformation maplet verifies that the email address is valid. The Expression transformation verifies the structure of the email string. The Labeler transformation checks the IP address against a reference table of international IP address suffixes.
- A Decision transformation receives the output from the transformation and the maplets. It calculates an overall record score for the customer contact record.

Create a bad record exception mapping that includes the maplet. The bad record exception mapping includes an Exception transformation that writes the exceptions to a bad records database table. A data analyst researches and updates exception records in the bad record table with the Analyst tool.

Bad Record Exception Example Input Groups

The Exception transformation has three input groups. The transformation has a Data group that receives the source data. It has the Quality Issues group, which receives data quality issues found by data quality transformations. It also has a Control group that contains the record score for the row.

The following figure shows the input groups in the Exception transformation:

	Name	Type	Precision	Scale	Input	Output
Inputs						
Data (11)						
1	CUST_ID	decimal	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	COMPANY	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	CONTACT	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	TITLE	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	ADDR1	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	ADDR2	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	ADDR3	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
8	ADDR4	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
9	COUNTRY	string	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
10	PHONE	string	50	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
11	EMAIL	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Quality Issues (7)						
1	CompanyStatus	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	LocalityStatus	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	ProvinceStatus	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	CountryStatus	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	ZipStatus	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	PhoneStatus	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	EmailStatus	string	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Control (1)						
1	Score	double	15	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Bad Record Exception Example Mapping Output

Add a Write transformation to the mapping and connect the standard output ports to the data object. The mapping also contains the bad record database object and the issues database object you created on the **Configuration** view.

Bad Records Table

The Bad Records table contains the exceptions with record scores between the lower and upper thresholds.

The following figure shows the bad records that the Exception transformation returns:

Name: [exc_BadRecords.Bad_Output_DI](#)

Workflow_ID	Row_Identifier	Record_Status	CUST_ID	COMPANY	CONTACT	TITLE	ADDR1	ADDR2
1	DummyWor ... 0	INVALID	7121657	WAITROSE	MRS LACI WINI...	DIRECTOR OF IS	380-394 N END...	FULHAM
2	DummyWor ... 1	INVALID	7121649	WAITROSE	MR NICHOLAS...	SENIOR PROJE...	EATON CNTR C...	EATON
3	DummyWor ... 2	INVALID	7121647	VARSHÉE SUPE...	MRS REUVEN C...	MANAGER	834 LONDON RD	THORNTON HE
4	DummyWor ... 3	INVALID	1002138	VICTORY SUPE...	MR RAY ARIAS	COMPUTER SPE...	22 FITCHBURG...	AYER
5	DummyWor ... 4	INVALID	1002137	VICTORY SUPE...	MR NEVILLE DE...	SR PROJ MANA...	21 TIMPANY BLV	GARDNER
6	DummyWor ... 5	INVALID	1002109	USDA FOREST...	MR MICHEAL F...	COMPUTER SPE...	1621 N KENT S...	ROSSLYN
7	DummyWor ... 6	INVALID	1002026	US ARMY CORP...	MR ROBERT EV...	PROGRAMMER...	20 MASS AVE NW	WASHINGTON
8	DummyWor ... 7	INVALID	1002062	US NAVY	MR JOHN WELCH	COMPUTER SPEC	DATA PROCESS...	OAK HARBOUR
9	DummyWor ... 8	INVALID	1062004	TRICOR INDUS...	MR MICHAEL G...	V.P.	8181 PROFESSI...	LANDOVER
10	DummyWor ... 9	INVALID	1001921	THE CORNER S...	MR DENIS LEE	MANAGER (\$\$...	102 MID STR 5...	REDHILL
11	DummyWor ... 10	INVALID	7121217	THE CORNER S...	MRS TESSI SAN...	PROGRAMMER	192 BATTERSE...	BATTERSEA

The Bad Records table includes all of the fields in the source record. A bad record also includes the following fields:

Workflow_ID

The name of the workflow that included the Exception transformation. The workflow contains the Exception transformation Mapping task and the Human task to review the issues. The Workflow_ID contains `DummyWorkflowID` if the Exception transformation is not in a workflow .

Row_Identifier

A unique number that identifies each row.

Record_Status

A record status for the Analyst tool. Each record in the Bad Records table receives an `Invalid` status. You can maintain the record status when you update records in the Analyst tool.

Issues Table

The Issues table contains one row for each row in the Bad Records table. Each row contains the issues that the data quality analysis found for the source record.

The following figure shows the columns in the Issues table:

Output
Name: [exc_BadRecords.Issues_DI](#)

Workflow_ID	Row_Identifier	COMPANY	DQAPRIORITY_...	ADDR2	DQAPRIORITY_...	ADDR3	DQAPRIORITY_...	COUNTRY	DQ...
1	DummyWor ... 0		1	invalid_locality	2		3		3
2	DummyWor ... 1		1	invalid_locality	2	invalid_province	3		3
3	DummyWor ... 2		1	invalid_locality	2		3		3
4	DummyWor ... 3		1		2		3		3
5	DummyWor ... 4		1		2		3		3
6	DummyWor ... 5		1	invalid_locality	2		3		3
7	DummyWor ... 6		1		2		3		3
8	DummyWor ... 7		1	invalid_locality	2		3		3
9	DummyWor ... 8		1	invalid_locality	2		3		3
10	DummyWor ... 9		1		2		3		3
11	DummyWor ... 10		1	invalid_locality	2		3		3

The Issues table contains the following columns:

Workflow_ID

Identifies the workflow that created the record. The workflow contains the Exception transformation Mapping task and the Human task to review the issue.

Row_Identifier

Identifies the record row in the database table. The row identifier identifies which row in the Bad Records table corresponds to the row in the Issues table.

Issue Field Name

The field name is the name of the field that might quality issues. When the field contains an error the column value is the quality issue text. In the figure above, the ADDR2 field name contains the `invalid_locality` quality issue.

DQAPriority

The issue priority. When multiple issues occur for the same field, the issue with the highest priority appears in the Issue Field Name.

Good Records Table

Each record in the Good Records table has a record score greater than the upper threshold. In this example, the upper threshold is 90.

The following figure shows the good records that the Exception transformation returns:

Output								
Name: exc_BadRecords.Output_DI								
	Score	CUST_ID	COMPANY	CONTACT	TITLE	ADDR1	ADDR2	ADDR3
1	100	7121669	YEOVALE STOR...	MRS OPPORTU...	OWNER/CONSULTANT	1 MARGROVE T...	BARNSTAPLE	DEVON
2	100	7121667	WHARFEDALE...	MRS PRAGER	GENERAL MANAGER	458 SOUTHCOA...	HULL	NORTH HI
3	100	1002195	WORLDSPAN	MR CHRISTOPH...	SENIOR ANALYST	N8J1-1 300 GA...	ATLANTA	GA
4	100	1002187	WINN DIXIE ST...	MS CORINA GA...	VICE PRESIDENT	1805 WAYNE M...	GOLDSBORO	NC
5	100	1002181	WINN DIXIE ST...	MR JENS GONY...	COMPUTER OPERATOR	506 W GANNO...	ZEBULON	NC
6	100	1002183	WINN DIXIE ST...	MS MERCIE VA...	MANAGER OF DBA	5715 GUNN HWY	TAMPA	FL
7	100	1000029	WINN DIXIE ST...	MR LYLE HICK...	MGR, DBA	3123 HWY 28 E	PINEVILLE	LA
8	100	1002178	WINN DIXIE ST...	MR HUSSEIN D...	AUTOMATED SYSTEMS MGR	2080 S FRONTA...	VICKSBURG	MS
9	100	1002177	WINN DIXIE ST...	MR REUVEN RH...	CHIEF DBA	695 S SEMORA...	ORLANDO	FL
10	100	1002180	WINN DIXIE ST...	MS GOLDI WEI...	MGR COORD	11957 S APOPK...	ORLANDO	FL

The Good Records table records contain the record score and the source data fields.

CHAPTER 8

Case Converter Transformation

This chapter includes the following topics:

- [Case Converter Transformation Overview, 160](#)
- [Case Strategy Properties, 160](#)
- [Configuring a Case Converter Strategy, 161](#)
- [Case Converter Transformation Advanced Properties, 161](#)
- [Case Converter Transformation in a Non-native Environment, 162](#)

Case Converter Transformation Overview

The Case Converter transformation is a passive transformation that standardizes the case of the alphabetic characters in input strings.

You can select a case conversion format, such as uppercase, lowercase, title case, and sentence case. You can also reverse the current case of each character in the input data.

The Case Converter transformation can use the values in the Valid column of a reference table to define the case of the input characters. When the transformation finds a match between an input value and a valid value, the transformation applies the case of the valid value to the input value. You can use reference tables when the case conversion type is **Title Case** or **Sentence case**.

You can create multiple case conversion strategies in a Case Converter transformation. Each strategy uses a single conversion type.

Case Strategy Properties

You can configure properties for case conversion strategies.

On the **Strategies** view, you can configure the following case conversion properties:

Conversion Type

Defines the case conversion method that a strategy uses. You can apply the following case conversion types:

- **Uppercase.** Converts all letters to uppercase.
- **Sentence Case.** Capitalizes the first letter of the field data string.

- **Toggle Case.** Converts lowercase letters to uppercase and uppercase letters to lowercase.
- **Title Case.** Capitalizes the first letter in each substring.
- **Lowercase.** Converts all letters to lowercase.

The default case conversion method is uppercase.

Leave Uppercase Words Unchanged

Overrides the chosen capitalization for uppercase strings.

Delimiters

Defines how capitalization functions for title case conversion. For example, choose a dash as a delimiter to transform "smith-jones" to "Smith-Jones." The default delimiter is the space character.

Reference Table

Applies the capitalization format specified by a reference table. Applies only if the case conversion option is **Title Case** or **Sentence case**. Click **New** to add a reference table to the strategy.

Note: If a reference table match occurs at the start of a token, the next character in the token changes to uppercase. For example, if the input string is mcdonald and the reference table has an entry for Mc, the output string is McDonald.

Configuring a Case Converter Strategy

To change the case of input strings, configure the settings in the **Strategies** view of a Case Converter transformation.

1. Select the **Strategies** view.
2. Click **New**.
The **New Strategy** wizard opens.
3. Optionally, edit the strategy name and description.
4. Click the **Inputs** and **Outputs** fields to select ports for the strategy.
5. Configure the strategy properties. The default conversion strategy is **Uppercase**.
6. Click **Next**.
7. Optionally, add reference tables to customize the case options for input data that matches reference table entries. Reference table case customization only applies to title case and sentence case strategies.
8. Click **Finish**.

Case Converter Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Case Converter transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Case Converter Transformation in a Non-native Environment

The Case Converter transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 9

Classifier Transformation

This chapter includes the following topics:

- [Classifier Transformation Overview, 163](#)
- [Classifier Models, 163](#)
- [Classifier Algorithms, 164](#)
- [Classifier Transformation Options, 164](#)
- [Classifier Strategies, 165](#)
- [Classifier Transformation Advanced Properties, 165](#)
- [Configuring a Classifier Strategy, 165](#)
- [Classifier Analysis Example, 166](#)
- [Classifier Transformation in a Non-native Environment, 171](#)

Classifier Transformation Overview

The Classifier transformation is a passive transformation that analyzes input fields and identifies the type of information in each field. Use a Classifier transformation when input fields contain multiple text values.

When you configure the Classifier transformation, you select a classifier model and a classifier algorithm. A classifier model is a type of reference data object. A classifier algorithm is a set of rules that calculates the number of similar words in a string and the relative positions of the words. The transformation compares the algorithm analysis with the content of the classifier model. The transformation returns the model classification that identifies the dominant type of information in the string.

The Classifier transformation can analyze strings of significant length. For example, you can use the transformation to classify the contents of email messages, social media messages, and document text. You pass the contents of each document or message to a field in a data source column, and you connect the column to a Classifier transformation. In each case, you prepare the data source so that each field contains the complete contents of a document or string you want to analyze.

Classifier Models

The Classifier transformation uses a reference data object called a classifier model to analyze input data. You select the classifier model when you configure a Classifier transformation. The transformation compares

a column of input data with the classifier model data and returns a label that describes the type of information in each input field.

A classifier model contains reference data rows and label values. The rows represent the input data on the port that you might connect to the Classifier transformation. The label values describe the types of information that the data rows contain. When you configure a classifier model, you assign a label to each reference data row in the model.

To link the reference data rows to the labels in a classifier model, you compile the model. The compilation process generates a series of logical associations between the data rows and the label values. When you run a mapping that reads the model, the Data Integration Service applies the model logic to the Classifier transformation input data. The Data Integration Service returns the labels that most accurately describe the information in each input data field.

You create a classifier model in the Developer tool. The Model repository stores the classifier model object. The Developer tool writes the data rows, the labels, and the compilation data to a file in the Informatica directory structure.

Classifier Algorithms

When you add a classifier model to a transformation strategy, you also select a classifier algorithm. The algorithm determines how the transformation compares the classifier model data to the input data.

You can select the **Naive Bayes** algorithm or the **Maximum entropy** algorithm.

Consider the following factors when you select an algorithm:

- The Maximum entropy algorithm performs a more thorough analysis than the Naive Bayes algorithm.
- A mapping that uses the Naive Bayes algorithm runs faster than a mapping that uses the Maximum entropy algorithm on the same data.
- Select the Maximum entropy algorithm with the classifier model that Informatica includes in the Core Accelerator.

Classifier Transformation Options

The Classifier transformation displays configurable options on a series of tabs or views in the Developer tool.

When you open a reusable transformation, the options appear on a series of tabs in the transformation editor. When you open a nonreusable transformation in a mapping, the options appear on a series of views in the mapping editor. Select the mapping properties tab to see the views on a nonreusable transformation.

You can select the following views:

General

View and update the transformation name and description.

Ports

View the input and output ports on the transformation.

Note: In a reusable Classifier transformation, the General and Ports views are combined in the **Overview** tab.

Strategies

Add, remove, or edit a strategy.

Dependencies

View the input and output ports on each strategy.

Advanced

Set the level of detail that the transformation writes to log files.

Classifier Strategies

A strategy is a set of data analysis operations that a transformation performs on input data. You create at least one strategy in the Classifier transformation. A Classifier strategy reads a single input port.

You define one or more operations in a strategy. A classifier operation identifies a classifier model and classifier algorithm to apply to the input port data. Each operation writes to a different output port. Create multiple operations in a strategy when you want to analyze an input port in different ways.

Note: When you want to identify the languages used in source data, select the Maximum entropy algorithm in the classifier operation.

Classifier Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Classifier transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Configuring a Classifier Strategy

You configure a strategy to identify the types of information in the data. Each strategy analyzes an input port.

In a nonreusable transformation, connect input ports to the transformation before you configure the strategy.

1. Open the transformation and select the **Strategies** view.
2. Click **New Strategy**.
The strategy creation wizard opens.
3. Enter a name and optional description for the strategy.
4. Select an input port from the Inputs field.

5. Verify that the input port precision value is high enough to read all fields on the input port. The port truncates the input data when it reaches the precision limit.
6. Select or clear the option to add score data to the strategy output.
7. Click **Next**.
8. Confirm the type of classifier operation, and click **Next**.
9. Select a classifier algorithm. You can select the following algorithms:
 - Naive Bayes
 - Maximum entropy

Note: To identify the language used in the source data, select the maximum entropy algorithm.
10. Verify the output port.

The transformation creates a single output port for each operation in a strategy. You can edit the port name and precision.
11. Select a classifier model.

The wizard lists the classifier model objects in the Model repository.
12. Click **Next** to add another operation to the strategy. Otherwise, click **Finish**.

Classifier Analysis Example

You are a data steward at a software company that released a new smartphone application. The company wants to understand the public response to the application and the media coverage it receives. The company asks you and your team to analyze social media comments about the application.

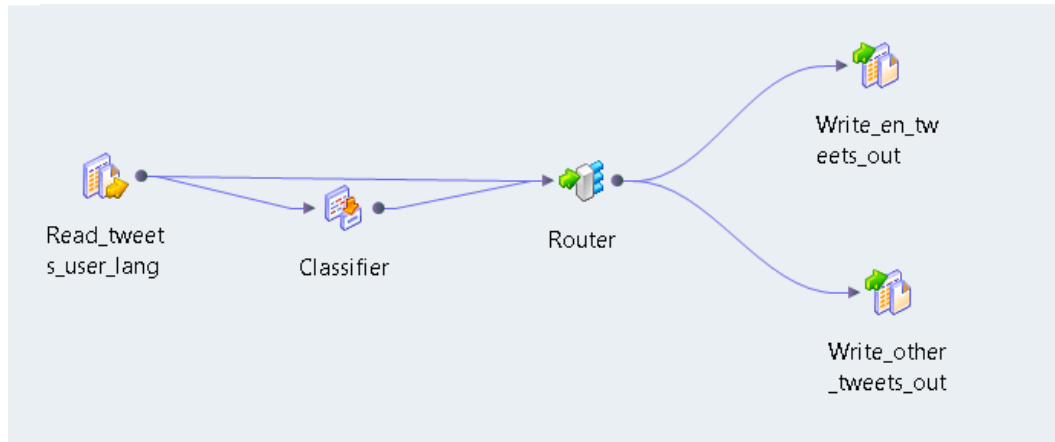
You decide to capture data from twitter feeds that discuss smartphones. You use the twitter application programming interface to filter the twitter data stream. You create a data source that contains the twitter data you want to analyze.

Because the twitter feeds contain messages in multiple languages, you must identify the language used in each message. You decide to use a Classifier transformation to analyze the languages. You create a mapping that identifies the languages in the source data and writes the twitter messages to English and non-English data targets.

Create the Classifier Mapping

You create a mapping that reads a data source, classifies the languages in the data, and writes the data to targets based on the languages they contain.

The following image shows the mapping in the Developer tool:



The mapping you create contains the following objects:

Object Name	Description
Read_tweet_user_lang	Data source. Contains the twitter messages
Classifier	Classifier transformation. Identifies the languages used in the twitter messages.
Router	Router transformation. Routes the twitter messages to data target objects according to the languages they contain.
Write_en_tweets_out	Data target. Contains twitter messages in English.
Write_other_tweets_out	Data target. Contains non-English-language twitter messages.

Input Data Sample

The following data fragment shows a sample of the twitter data that you analyze in the mapping:

Twitter Message

```
RT @GanaphoneS3: Faltan 10 minutos para la gran rifa de un iPhone 5...
RT @Clarified: How to Downgrade Your iPhone 4 From iOS 6.x to iOS 5.x (Mac)...
RT @jerseyjazz: The razor was the iPhone of the early 2000s
RT @KrissiDevine: Apple Pie that I made for Thanksgiving. http://t.com/s9ImzFxO
RT @sophieHz: Dan yang punya 2 kupon undian. Masuk dalam kotak undian yang berhadiah Samsung
RT @IsabelFreitas: o galaxy tem isso isso isso e a bateria ã melhor que do iPhone
RT @PremiusIpad: Faltan 15 minutos para la gran rifa de un iPhone 5...
RT @payyton3: I want apple cider
RT @wiesteronder: Retweet als je iets van Apple, Nike, Adidas of microsoft hebt!
```

Data Source Configuration

The data source contains a single port. Each row on the port contains a single twitter message.

The following table describes the configuration of the data source:

Port Name	Port Type	Precision
text	n/a	200

Classifier Transformation Configuration

The Classifier transformation uses a single input port and output port. The transformation input port reads the text field from the data source. The output port contains the language identified for each twitter message in the text field. The Classifier transformation uses ISO country codes to identify the language.

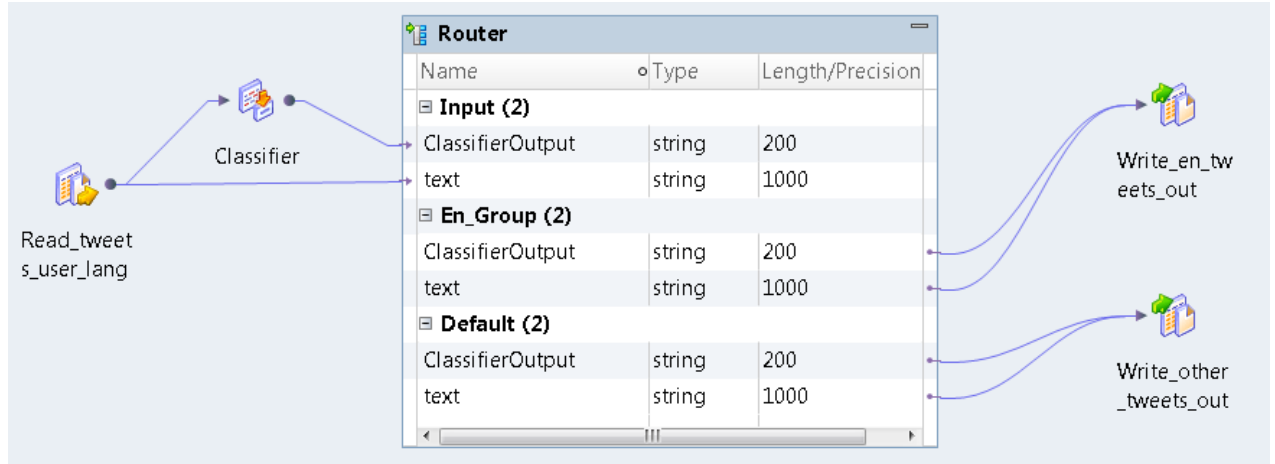
The following table describes the configuration of the Classifier transformation:

Port Name	Port Type	Precision	Strategy
text_input	Input	200	Classifier1
Classifier_Output	Output	2	Classifier1

Router Transformation Configuration

The Router transformation uses two input ports. It reads the twitter messages from the data source and the ISO country codes from the Classifier transformation. The Router transformation routes the data on the input ports to different output ports based on a condition that you specify.

The following image shows the Router transformation port groups and port connections:



The following table describes the configuration of the Router transformation:

Port Name	Port Type	Port Group	Precision
Classifier_Output	Input	Input	2
text	Input	Input	200
Classifier_Output	Input	Default	2
text	Input	Default	200
Classifier_Output	Input	En_Group	2
text	Input	En_Group	200

You configure the transformation to create data streams for English-language messages and for messages in other languages. To create a data stream, add an output port group to the transformation. Use the **Groups** options on the transformation to add the port group.

To determine how the transformation routes data to each data stream, you define a condition on a port group. The condition identifies a port and specifies a possible value on the port. When the transformation finds an input port value that matches the condition, it routes the input data to the port group that applies the condition.

Define the following condition on the En_Group:

```
ClassifierOutput='en'
```

Note: The Router transformation reads data from two objects in the mapping. The transformation can combine the data in each output group because it does not alter the row sequence defined in the data objects.

Data Target Configuration

The mapping contains a data target for English-language twitter messages and a target for messages in other languages. You connect the ports from a Router transformation output group to a data target.

The following table describes the configuration of the data targets:

Port Name	Port Type	Precision
text	n/a	200
Classifier_Output	n/a	2

Classifier Mapping Outcome

When you run the mapping, the Classifier transformation identifies the language of each twitter message. The Router transformation writes the message text to data targets based on the language classifications.

The following data fragment shows a sample of the English-language target data:

ISO Country Code	Twitter Message
en	RT @Clarified: How to Downgrade Your iPhone 4 From iOS 6.x to iOS 5.x (Mac)...
en	RT @jerseyjazz: The razor was the iPhone of the early 2000s
en	RT @KrissiDevine: Apple Pie that I made for Thanksgiving. http://t.com/s9ImzFxO
en	RT @payyton3: I want apple cider

The following data fragment shows a sample of the target data identified for other languages:

ISO Country Code	Twitter Message
es	RT @GanaphoneS3: Faltan 10 minutos para la gran rifa de un iPhone 5...
id	RT @sophieHz: Dan yang punya 2 kupon undian. Masuk dalam kotak undian yang berhadiah Samsung Champ.
pt	RT @IsabelFreitas: o galaxy tem isso isso isso e a bateria ã melhor que do iPhone
es	RT @PremiusIpad: Faltan 15 minutos para la gran rifa de un iPhone 5...
nl	RT @wiesteronder: Retweet als je iets van Apple, Nike, Adidas of microsoft hebt! http://t.co/Je6Ts00H

Classifier Transformation in a Non-native Environment

The Classifier transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 10

Comparison Transformation

This chapter includes the following topics:

- [Comparison Transformation Overview, 172](#)
- [Field Matching Strategies, 172](#)
- [Identity Matching Strategies, 175](#)
- [Configuring a Comparison Strategy, 175](#)
- [Comparison Transformation Advanced Properties, 176](#)
- [Comparison Transformation in a Non-native Environment, 176](#)

Comparison Transformation Overview

The Comparison transformation is a passive transformation that evaluates the similarity between pairs of input strings and calculates the degree of similarity for each pair as a numerical score.

When you configure the transformation, you select a pair of input columns and assign a matching strategy to them.

The Comparison transformation outputs match scores in a range from 0 to 1, where 1 indicates a perfect match.

Note: The strategies available in the Comparison transformation are also available in the Match transformation. Use the Comparison transformation to define match comparison operations that you will add to a matching maplet. You can add multiple Comparison transformations to the maplet. Use the Match transformation to define match comparisons in a single transformation. You can embed a matching maplet in a Match transformation.

Field Matching Strategies

The Comparison transformation includes predefined field matching strategies that compare pairs of input data fields.

Bigram

Use the Bigram algorithm to compare long text strings, such as postal addresses entered in a single field.

The Bigram algorithm calculates a match score for two data strings based on the occurrence of consecutive characters in both strings. The algorithm looks for pairs of consecutive characters that are common to both strings. It divides the number of pairs that match in both strings by the total number of character pairs.

Bigram Example

Consider the following strings:

- larder
- lerder

These strings yield the following Bigram groups:

```
l a, a r, r d, d e, e r
l e, e r, r d, d e, e r
```

Note that the second occurrence of the string "e r" within the string "lerder" is not matched, as there is no corresponding second occurrence of "e r" in the string "larder".

To calculate the Bigram match score, the transformation divides the number of matching pairs (6) by the total number of pairs in both strings (10). In this example, the strings are 60% similar and the match score is 0.60.

Hamming Distance

Use the Hamming Distance algorithm when the position of the data characters is a critical factor, for example in numeric or code fields such as telephone numbers, ZIP Codes, or product codes.

The Hamming Distance algorithm calculates a match score for two data strings by computing the number of positions in which characters differ between the data strings. For strings of different length, each additional character in the longest string is counted as a difference between the strings.

Hamming Distance Example

Consider the following strings:

- Morlow
- Marlowes

The highlighted characters indicate the positions that the Hamming algorithm identifies as different.

To calculate the Hamming match score, the transformation divides the number of matching characters (5) by the length of the longest string (8). In this example, the strings are 62.5% similar and the match score is 0.625.

Edit Distance

Use the Edit Distance algorithm to compare words or short text strings, such as names.

The Edit Distance algorithm calculates the minimum "cost" of transforming one string to another by inserting, deleting, or replacing characters.

Edit Distance Example

Consider the following strings:

- Levenston

- Levenshtein

The highlighted characters indicate the operations required to transform one string into the other.

The Edit Distance algorithm divides the number of unchanged characters (8) by the length of the longest string (11). In this example, the strings are 72.7% similar and the match score is 0.727.

Jaro Distance

Use the Jaro Distance algorithm to compare two strings when the similarity of the initial characters in the strings is a priority.

The Jaro Distance match score reflects the degree of similarity between the first four characters of both strings and the number of identified character transpositions. The transformation weights the importance of the match between the first four characters by using the value that you enter in the Penalty property.

Jaro Distance Properties

When you configure a Jaro Distance algorithm, you can configure the following properties:

Penalty

Determines the match score penalty if the first four characters in two compared strings are not identical. The transformation subtracts the full penalty value for a first-character mismatch. The transformation subtracts fractions of the penalty based on the position of the other mismatched characters. The default penalty value is 0.20.

Case Sensitive

Determines whether the Jaro Distance algorithm considers character case when it compares characters.

Jaro Distance Example

Consider the following strings:

- 391859
- 813995

If you use the default Penalty value of 0.20 to analyze these strings, the Jaro Distance algorithm returns a match score of 0.513. This match score indicates that the strings are 51.3% similar.

Reverse Hamming Distance

Use the Reverse Hamming Distance algorithm to calculate the percentage of character positions that differ between two strings, reading from right to left.

The Hamming Distance algorithm calculates a match score for two data strings by computing the number of positions in which characters differ between the data strings. For strings of different length, the algorithm counts each additional character in the longest string as a difference between the strings.

Reverse Hamming Distance Example

Consider the following strings, which use right-to-left alignment to mimic the Reverse Hamming algorithm:

- 1-999-9999
- **011-01-999-9991**

The highlighted characters indicate the positions that the Reverse Hamming Distance algorithm identifies as different.

To calculate the Reverse Hamming match score, the transformation divides the number of matching characters (9) by the length of the longest string (15). In this example, the match score is 0.6, indicating that the strings are 60% similar.

Identity Matching Strategies

The Comparison transformation includes predefined identity matching strategies that you can use to find matches for individuals, addresses, or corporate entities.

The following table describes the match operation that each identity matching strategy performs:

Identity Matching Strategy	Match Operation
Address	Identifies an address match.
Contact	Identifies a contact within an organization at a single location.
Corp Entity	Identifies an organization by its legal corporate name.
Division	Identifies an organization at an address.
Family	Identifies a family by a family name and address or telephone number.
Fields	Identifies custom fields that you select.
Household	Identifies members of the same family at the same residence.
Individual	Identifies an individual by name and either ID or date of birth.
Organization	Identifies an organization by name.
Person Name	Identifies a person by name.
Resident	Identifies a person at an address.
Wide Contact	Identifies a contact within an organization regardless of location.
Wide Household	Identifies members of the same family at regardless of location.

Note: Identity matching strategies read reference data files called **populations**. Contact your Informatica Administrator user for information about population data files installed on your system.

Configuring a Comparison Strategy

To configure a comparison strategy, edit the settings in the **Strategies** view of a Comparison transformation.

1. Select the **Strategies** view.
2. Select a comparison strategy from the **Strategies** section.

3. In the **Fields** section, double-click a cell in the **Available Fields** column to select an input.

Note: You must select an input for each row that displays a bolded input name in the **Input Fields** column.

Comparison Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Comparison transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Comparison Transformation in a Non-native Environment

The Comparison transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 11

Consolidation Transformation

This chapter includes the following topics:

- [Consolidation Transformation Overview, 177](#)
- [Consolidation Mappings, 178](#)
- [Consolidation Transformation Ports, 178](#)
- [Consolidation Transformation Views, 178](#)
- [Simple Strategies, 181](#)
- [Row-Based Strategies, 182](#)
- [Advanced Strategies, 183](#)
- [Simple Consolidation Functions, 183](#)
- [Row-Based Consolidation Functions, 187](#)
- [Consolidation Mapping Example, 191](#)
- [Configuring a Consolidation Transformation, 192](#)
- [Consolidation Transformation in a Non-native Environment, 193](#)

Consolidation Transformation Overview

The Consolidation transformation is an active transformation that analyzes groups of related records and creates a consolidated record for each group. Use the Consolidation transformation to consolidate record groups generated by transformations such as the Key Generator, Match, and Association transformations.

The Consolidation transformation generates consolidated records by applying strategies to groups of related records. The transformation contains an output port that indicates which record is the consolidated record. You can choose to limit the transformation output to include only consolidated records.

For example, you can consolidate groups of duplicate employee records that the Match transformation generates. The Consolidation transformation can create a consolidated record that contains merged data from all records in the group.

You can configure the Consolidation transformation to use different types of strategies based on your consolidation requirements. Use simple strategies to create a consolidated record from multiple records. When you use simple strategies, you specify a strategy for each port. Use a row-based strategy to analyze rows in the record group and create a consolidated record with the values from one of the rows. Use an advanced strategy to create a consolidated record by applying an expression that you create.

Consolidation Mappings

To consolidate records, create a mapping that creates groups of related records. Add a Consolidation transformation to a mapping, and configure the transformation to consolidate each record group into a single master record.

Connect a Consolidation transformation to other transformations according to the business objectives and the data requirements. To consolidate matched records, you can connect the Consolidation transformation to a Match transformation. To consolidate records as part of exception record management, connect the Consolidation transformation to an Exception transformation. If you use a Key Generator transformation to group records, you can connect a Consolidation transformation directly to the Key Generator transformation. The Consolidation transformation creates a consolidated record for each group that the Key Generator transformation creates.

Mapping Output in Native and Hadoop Environments

When you run a consolidation mapping in a native environment and in a Hadoop environment, the Consolidation transformation can generate different results. Because the mapping runs on multiple nodes in Hadoop, the input records can enter the Consolidation transformation in a different order than in the native environment. As a result, the transformation can generate different sets of survivor records in each environment for the same input data set. The transformation calculations and the consolidated results are accurate for the input row order in each case.

To generate the same survivor records in native and Hadoop environments, configure the Consolidation transformation to sort the records in the following order:

- First, sort the records on the Group By port.
- Then, sort the records in the order in which the input ports appear in the transformation.

Consolidation Transformation Ports

The Developer tool creates an output port for each input port that you add. You cannot manually add output ports to the transformation. The Consolidation transformation also includes an **IsSurvivor** output port that indicates the consolidated records.

One of the input ports that you add to the Consolidation transformation must contain group keys. The Consolidation transformation requires group key information because consolidation strategies process record groups rather than entire datasets.

When you add an input port, the Developer tool creates an output port name by adding the suffix "1" to the input port name. The transformation also includes the **IsSurvivor** output port, which indicates whether a record is the consolidated record. For consolidated records, the Consolidation transformation writes the string "Y" to the **IsSurvivor** port. For input records, the Consolidation transformation writes the string "N" to the **IsSurvivor** port.

Consolidation Transformation Views

The Consolidation transformation contains views for ports, strategies, and advanced properties.

Consolidation Transformation Strategies View

The **Strategies** view contains properties for simple, row-based, and advanced strategies.

The following list describes the types of consolidation strategies:

Simple Strategy

A simple strategy analyzes all the values of a port in a record group and selects one value. You specify a simple strategy for each port. The Consolidation transformation uses the port values selected by all simple strategies to create a consolidated record. Examples of simple strategies include the most frequent value in a port, the longest value in a port, or the most frequent non-blank value in a port.

Row-Based Strategy

A row-based strategy analyzes rows in the record group and selects one row. The Consolidation transformation uses the port values from that row to create a consolidated record. Examples of row-based strategies include the highest character count, the lowest number of blank fields, or the highest count of the most frequent fields.

Advanced Strategy

An advanced strategy analyzes a record group using strategies that you define. You build advanced strategies by using consolidation functions in an expression. The Consolidation transformation creates a consolidated record based on the output of the expression. The expression that you create can also use all of the functions available in the Decision transformation.

Consolidation Transformation Advanced Properties

The Consolidation transformation contains advanced properties that determine the sort behavior, the output mode, the cache memory behavior, and the tracing level.

You can configure the following advanced properties:

Sort

Determines whether the transformation sorts the input rows on the **Group By** port data. The property is enabled by default.

Select the property if the input rows are not presorted.

Case Sensitive Sort

Determines whether the sort operation is case-sensitive. The property is enabled by default.

Output Mode

Determines whether the transformation writes all records as output or writes the consolidated records as output. The default value is All.

Cache File Directory

Specifies the directory to which the Data Integration Service writes temporary data for the current transformation. The Data Integration Service writes temporary files to the directory when the volume of input data is greater than the available system memory. The Data Integration Service deletes the temporary files after the mapping runs.

You can enter a directory path on the property, or you can use a parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory. The default value is the CacheDir system parameter.

Cache File Size

Determines the amount of system memory that the Data Integration Service uses to sort the input data on the transformation.

Before it sorts the data, the Data Integration Service allocates the amount of memory that you specify. If the sort operation generates a greater amount of data, the Data Integration Service writes the excess data to the cache file directory. If the sort operation requires more memory than the system memory and the file storage can provide, the mapping fails.

The transformation reads the value in bytes. The default value is 400,000 bytes. The maximum value is 2,147,483,647 bytes. You can use a parameter to specify the cache file size.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Cache File Size

The cache file size property determines the amount of system memory that the Data Integration Service assigns to the Consolidation transformation for sort operations. Configure the property with a value that is less than or equal to the amount of RAM on the Data Integration Service host machine.

For best performance, specify a cache file size of at least 16 MB. Set the cache file size value in bytes on the **Advanced Properties** view.

Before it begins a sort operation, the Data Integration Service allocates the memory that the cache file size property specifies. The Data Integration Service passes all input data into the Consolidation transformation before it performs a sort operation.

If the volume of input data is greater than the cache file size, the Data Integration Service writes data to the cache file directory. When it writes data to the cache file directory, the Data Integration Service consumes disk space that represents at least twice the input data volume.

Use the following formula to determine the size of incoming data:

$$[\text{number_of_input_rows} * (\text{Sum}(\text{column_size}) + 16)]$$

The following table lists the possible data types and the column size values to apply in cache file data calculations:

Data Type	Column Size
Binary	Precision + 8. Round to nearest multiple of 8.
Date/Time	29
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

Data Type	Column Size
Integer	16
String, Text	Unicode mode: 2*(precision + 5) ASCII mode: precision + 9

Simple Strategies

A simple strategy analyzes a port in a record group and returns one value. You specify a simple strategy for each port. The Consolidation transformation uses the port values selected by all simple strategies to create a consolidated record.

When you configure a strategy in the **Strategies** view of the transformation, the strategy displays the following text as the consolidation method:

Use default.

The default strategy is "Highest row ID."

You can choose from the following simple strategies:

Average

Analyzes a port in the record group and returns the average of all values.

For String and Date/time datatypes, the strategy returns the most frequently occurring value.

Longest

Analyzes a port in the record group and returns the value with the highest character count. If the highest character count is shared by two or more values, the strategy returns the first qualifying value.

Maximum

Analyzes a port in the record group and returns the highest value.

For the String datatype, the strategy returns the longest string. For the Date/time datatype, the strategy returns the most recent date.

Minimum

Analyzes a port in the record group and returns the lowest value.

For the String datatype, the strategy returns the shortest string. For the Date/time datatype, the strategy returns the earliest date.

Most frequent

Analyzes a port in the record group and returns the most frequently occurring value, including blank or null values. If the highest number of occurrences is shared by two or more values, the strategy returns the first qualifying value.

Most frequent non-blank

Analyzes a port in the record group and returns the most frequently occurring value, excluding blank or null values. If the highest number of non-blank occurrences is shared by two or more values, the strategy returns the first qualifying value.

Shortest

Analyzes a port in the record group and returns the value with the lowest character count. If the lowest character count is shared by two or more values, the strategy returns the first qualifying value.

Highest row ID

Analyzes a port in the record group and returns the value with the highest row ID.

Row-Based Strategies

A row-based strategy analyzes rows in the record group and selects one row. The Consolidation transformation uses the port values from that row to create a consolidated record. The default strategy is "most data."

Choose one of the following row-based strategies:

Most data

Selects the row with the highest character count. If the highest character count is shared by two or more rows, the strategy returns the last qualifying value.

Most filled

Selects the row with the highest number of non-blank columns. If the highest number of non-blank columns is shared by two or more rows, the strategy returns the last qualifying value.

Modal exact

Selects the row with the highest count of the most frequent non-blank values. For example, consider a row that has three ports that contain the most frequent values in the record group. The count of the most frequent values for that row is "3."

If the highest count of the most frequent non-blank values is shared by two or more rows, the strategy returns the last qualifying value.

Row-Based Strategy Example

The following table displays a sample record group. The last column describes the reasons why specific row-based strategies select different rows in this record group.

Product ID	First Name	Last Name	ZIP Code	Strategy Selection
2106	Bartholomew		28516	The Most Data strategy selects this row because the row contains more characters than the other rows.
2236	Bart	Smith	28579	The Most Filled strategy selects this row because the row has more non-blank columns than the other rows.
2236	<Blank>	Smith	28516	The Modal Exact strategy selects this row because the row contains the highest count of the most frequent values.

Advanced Strategies

You can use advanced strategies to create consolidation strategies from predefined functions. You can use consolidation functions and other Informatica functions.

You can create expressions that contain simple consolidation functions or row-based consolidation functions. You use the simple consolidation functions to build a consolidated record based on port values in the record group. You use the row-based consolidation functions to select a row from the record group.

Consolidation expressions must populate all of the output ports in the Consolidation transformation. If consolidation expressions do not use all output ports, the transformation causes mappings to fail.

You can use a simple or row-based strategy as a template for an advanced strategy. Configure a simple or row-based strategy and then select Advanced. The Consolidation transformation generates an expression with functions that perform the strategy. You can add more functions to implement additional requirements.

Simple Consolidation Functions

Simple consolidation functions select a value from all the port values in a record group. When you use a simple consolidation function, you provide the function with a port and the group by port.

CONSOL_AVG

Analyzes a port in the record group and returns the average of all values.

Syntax

```
CONSOL_AVG(string, group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

The average of all values in a port.

For String and Date/time datatypes, the function returns the most frequently occurring value.

Example

The following expression uses the `CONSOL_AVG` function to find the average value of the SalesTotal input port:

```
SalesTotal1:= CONSOL_AVG(SalesTotal, GroupKey)
```

In this expression, the `CONSOL_AVG` function uses the GroupKey port to identify a record group. Within that record group, the function analyzes the SalesTotal port and returns the average value. The expression writes the average value to the SalesTotal1 output port.

CONSOL_LONGEST

Analyzes a port in the record group and returns the value with the highest character count.

Syntax

```
CONSOL_LONGEST(string, group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

The port value with the highest character count.

If the highest character count is shared by two or more values, the strategy returns the first qualifying value.

Example

The following expression uses the `CONSOL_LONGEST` function to analyze the `FirstName` input port and find the value with the highest character count:

```
FirstName1:= CONSOL_LONGEST(FirstName, GroupKey)
```

In this expression, the `CONSOL_LONGEST` function uses the `GroupKey` port to identify a record group. Within that record group, the function analyzes the `FirstName` port and returns the longest value. The expression writes this value to the `FirstName1` output port.

CONSOL_MAX

Analyzes a port in the record group and returns the highest value.

Syntax

```
CONSOL_MAX(string, group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

The highest port value.

For the String datatype, the function returns the longest string. For the Date/time datatype, the function returns the most recent date.

Example

The following expression uses the `CONSOL_MAX` function to analyze the `SalesTotal` input port and find the highest value:

```
SalesTotal1:= CONSOL_MAX(SalesTotal, GroupKey)
```

In this expression, the `CONSOL_MAX` function uses the `GroupKey` port to identify a record group. Within that record group, the function analyzes the `SalesTotal` port and returns the highest value. The expression writes this value to the `SalesTotal1` output port.

CONSOL_MIN

Analyzes a port in the record group and returns the lowest value.

Syntax

```
CONSOL_MIN(string, group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

The lowest port value.

For the String datatype, the function returns the shortest string. For the Date/time datatype, the function returns the earliest date.

Example

The following expression uses the `CONSOL_MIN` function to analyze the `SalesTotal` input port and find the lowest value:

```
SalesTotal1:= CONSOL_MIN(SalesTotal, GroupKey)
```

In this expression, the `CONSOL_MIN` function uses the `GroupKey` port to identify a record group. Within that record group, the function analyzes the `SalesTotal` port and returns the lowest value. The expression writes this value to the `SalesTotal1` output port.

CONSOL_MOSTFREQ

Analyzes a port in the record group and returns the most frequently occurring value, including blank or null values.

Syntax

```
CONSOL_MOSTFREQ(string, group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

The most frequently occurring value, including blank or null values.

If the highest number of occurrences is shared by two or more values, the strategy returns the first qualifying value.

Example

The following expression uses the `CONSOL_MOSTFREQ` function to analyze the Company input port and find the most frequently occurring value:

```
Company1:= CONSOL_MOSTFREQ(Company, GroupKey)
```

In this expression, the `CONSOL_MOSTFREQ` function uses the GroupKey port to identify a record group. Within that record group, the function analyzes the Company port and returns the most frequently occurring value. The expression writes this value to the Company1 output port.

CONSOL_MOSTFREQ_NB

Analyzes a port in the record group and returns the most frequently occurring value, excluding blank or null values.

Syntax

```
CONSOL_MOSTFREQ_NB(string, group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

The most frequently occurring value, excluding blank or null values.

If the highest number of occurrences is shared by two or more values, the strategy returns the first qualifying value.

Example

The following expression uses the `CONSOL_MOSTFREQ_NB` function to analyze the Company input port and find the most frequently occurring value:

```
Company1:= CONSOL_MOSTFREQ_NB(Company, GroupKey)
```

In this expression, the `CONSOL_MOSTFREQ_NB` function uses the GroupKey port to identify a record group. Within that record group, the function analyzes the Company port and returns the most frequently occurring value. The expression writes this value to the Company1 output port.

CONSOL_SHORTEST

Analyzes a port in the record group and returns the value with the lowest character count.

Syntax

```
CONSOL_SHORTEST(string, group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

The port value with the lowest character count.

If the lowest character count is shared by two or more values, the strategy returns the first qualifying value.

Example

The following expression uses the `CONSOL_SHORTEST` function to analyze the `FirstName` input port and find the value with the lowest character count:

```
FirstName1:= CONSOL_SHORTEST(FirstName, GroupKey)
```

In this expression, the `CONSOL_SHORTEST` function uses the `GroupKey` port to identify a record group. Within that record group, the function analyzes the `FirstName` port and returns the shortest value. The expression writes this value to the `FirstName1` output port.

Row-Based Consolidation Functions

Use row-based consolidation functions to select a record in a record group. You must use row-based consolidation functions within `IF-THEN-ELSE` statements.

CONSOL_GETROWFIELD

Reads the row identified by a row-based consolidation function and returns the value for the port you specify. You use a numeric argument to specify a port.

You must use the `CONSOL_GETROWFIELD` function in conjunction with one of the following row-based consolidation functions:

- `CONSOL_MODALEXACT`
- `CONSOL_MOSTDATA`
- `CONSOL_MOSTFILLED`

For each input port in a row-based consolidation function, you must use one instance of the `CONSOL_GETROWFIELD` function .

Syntax

```
CONSOL_GETROWFIELD(value)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>value</i>	Required	Number that indicates an input port in the row-based consolidation function. Use "0" to specify the leftmost port in the function. Use subsequent numbers to indicate other ports.

Return Value

The value of the port you specify. The function reads this value from a row identified by a row-based consolidation function.

Example

The following expression uses the `CONSOL_GETROWFIELD` function in conjunction with the `CONSOL_MOSTDATA` function:

```
IF (CONSOL_MOSTDATA(First_Name, Last_Name, GroupKey, GroupKey))
THEN
First_Name1 := CONSOL_GETROWFIELD(0)
Last_Name1 := CONSOL_GETROWFIELD(1)
GroupKey1 := CONSOL_GETROWFIELD(2)
ELSE
First_Name1 := First_Name
Last_Name1 := Last_Name
GroupKey1 := GroupKey
ENDIF
```

In this expression, the `CONSOL_MOSTDATA` function analyzes rows in a record group and identifies a single row. The `CONSOL_GETROWFIELD` functions use consecutive numbers to read the port values for that row and write the values to the output ports.

CONSOL_MODALExact

Identifies the row with the highest count of the most frequent values.

For example, consider a row that has three ports that contain the most frequent values in the record group. The count of the most frequent values for that row is "3."

You must use this function in conjunction with the `CONSOL_GETROWFIELD` function. The `CONSOL_GETROWFIELD` returns the values from the row that the `CONSOL_MODALExact` function identifies.

Syntax

```
CONSOL_MODALExact(string1, [string2, ..., stringN,]
group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

TRUE for the row that scores the highest count of the most frequent fields, FALSE for all other rows.

Example

The following expression uses the `CONSOL_MODALSEXACT` function to find the row that contains the highest count of the most frequent fields:

```
IF (CONSOL_MODALSEXACT(First_Name,Last_Name,GroupKey,GroupKey))
THEN
First_Name1 := CONSOL_GETROWFIELD(0)
Last_Name1 := CONSOL_GETROWFIELD(1)
GroupKey1 := CONSOL_GETROWFIELD(2)
ELSE
First_Name1 := First_Name
Last_Name1 := Last_Name
GroupKey1 := GroupKey
ENDIF
```

In this expression, the `CONSOL_MODALSEXACT` function analyzes rows in a record group and identifies a single row. The `CONSOL_GETROWFIELD` functions use consecutive numbers to read the port values for that row and write the values to the output ports.

CONSOL_MOSTDATA

Identifies the row that contains the most characters across all ports

You must use this function in conjunction with the `CONSOL_GETROWFIELD` function. The `CONSOL_GETROWFIELD` returns the values from the row that the `CONSOL_MOSTDATA` function identifies.

Syntax

```
CONSOL_MOSTDATA(string1, [string2, ..., stringN],
group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

TRUE for the row that contains the most characters across all ports, FALSE for all other rows.

Example

The following expression uses the `CONSOL_MOSTDATA` function to find the row that contains the most characters:

```
IF (CONSOL_MOSTDATA(First_Name,Last_Name,GroupKey,GroupKey))
THEN
First_Name1 := CONSOL_GETROWFIELD(0)
Last_Name1 := CONSOL_GETROWFIELD(1)
GroupKey1 := CONSOL_GETROWFIELD(2)
ELSE
First_Name1 := First_Name
Last_Name1 := Last_Name
GroupKey1 := GroupKey
ENDIF
```

In this expression, the `CONSOL_MOSTDATA` function analyzes rows in a record group and identifies a single row. The `CONSOL_GETROWFIELD` functions use consecutive numbers to read the port values for that row and write the values to the output ports.

CONSOL_MOSTFILLED

Identifies the row that contains the highest number of non-blank fields.

You must use this function in conjunction with the `CONSOL_GETROWFIELD` function. The `CONSOL_GETROWFIELD` returns the values from the row that the `CONSOL_MOSTFILLED` function identifies.

Syntax

```
CONSOL_MOSTFILLED(string1, [string2, ..., stringN,]  
                 group by)
```

The following table describes the arguments for this command:

Argument	Required/Optional	Description
<i>string</i>	Required	Input port name.
<i>group by</i>	Required	Name of the input port that contains the group identifier.

Return Value

TRUE for the row that contains the highest number of non-blank fields, FALSE for all other rows.

Example

The following expression uses the `CONSOL_MOSTFILLED` function to find the row that contains the most characters:

```
IF (CONSOL_MOSTFILLED(First_Name,Last_Name,GroupKey,GroupKey))  
THEN  
  First_Name1 := CONSOL_GETROWFIELD(0)  
  Last_Name1  := CONSOL_GETROWFIELD(1)  
  GroupKey1   := CONSOL_GETROWFIELD(2)  
ELSE  
  First_Name1 := First_Name  
  Last_Name1  := Last_Name  
  GroupKey1   := GroupKey  
ENDIF
```

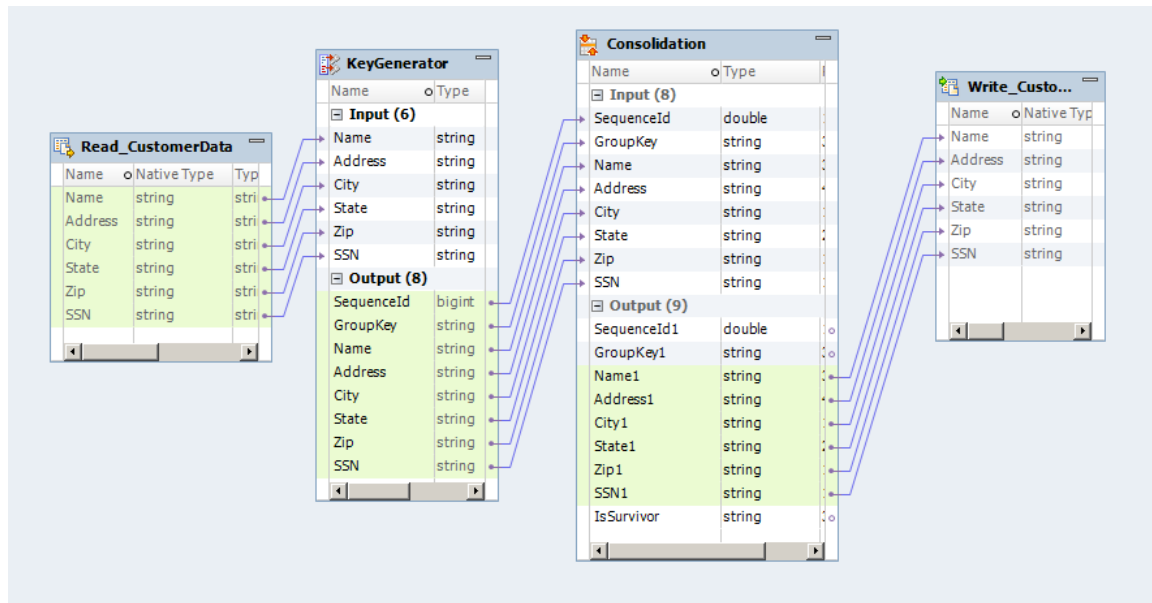
In this expression, the `CONSOL_MOSTFILLED` function analyzes rows in a record group and identifies a single row. The `CONSOL_GETROWFIELD` functions use consecutive numbers to read the port values for that row and write the values to the output ports.

Consolidation Mapping Example

Your organization needs to consolidate duplicate customer records. To consolidate the customer records, you group the data with a Key Generator transformation and use the Consolidation transformation to consolidate the records.

You create a mapping with a data source containing customer records, a Key Generator transformation, a Consolidation transformation, and a data target. The mapping groups customer records, consolidates the groups, and writes a single consolidated record.

The following figure shows the mapping:



Input Data

The input data that you want to analyze contains customer information.

The following table contains the input data for this example:

Name	Address	City	State	Zip	SSN
Dennis Jones	100 All Saints Ave	New York	NY	10547	987-65-4320
Dennis Jones	1000 Alberta Rd	New York	NY	10547	987-65-4320
D Jones	100 All Saints Ave	New York	NY	10547-1521	

Key Generator Transformation

Use the Key Generator transformation to group the input data based on the ZIP code port.

The transformation returns the following data:

SequenceId	GroupKey	Name	Address	City	State	ZIP Code	SSN
1	10547	Dennis Jones	100 All Saints Ave	New York	NY	10547	987-65-4320
2	10547	Dennis Jones	1000 Alberta Rd	New York	NY	10547	
3	10547	D Jones	100 All Saints Ave	New York	NY	10547-1521	987-65-4320

Consolidation Transformation

Use the Consolidation transformation to generate the consolidated record.

Configure the Consolidation transformation to use the row-based strategy type. Select the Modal Exact strategy to choose the row with the highest count of the most frequent values. The Modal Exact strategy uses the values from that row to generate a consolidated record. The consolidated record is the record with the "Y" value in the IsSurvivor port.

The transformation returns the following data:

GroupKey	Name	Address	City	State	ZIP Code	SSN	IsSurvivor
10547	Dennis Jones	100 All Saints Ave	New York	NY	10547	987-65-4320	N
10547	Dennis Jones	1000 Alberta Rd	New York	NY	10547		N
10547	D Jones	100 All Saints Ave	New York	NY	10547-1521	987-65-4320	N
10547	D Jones	100 All Saints Ave	New York	NY	10547-1521	987-65-4320	Y

Consolidation Mapping Output

Configure the Consolidation transformation so that the mapping output contains only consolidated records.

In this example, you are reasonably confident that the most frequent values selected by the Modal Exact strategy are the correct port values. To write only consolidated records to the mapping target, you select the **Advanced** view and set the output mode to "Survivor Only."

When you run the mapping, the mapping output only contains consolidated records.

Configuring a Consolidation Transformation

When you configure the Consolidation transformation, you choose strategy types, choose strategies or write expressions, select a grouping port, and configure advanced options.

1. Select the **Consolidation** view.
2. Choose a strategy type.
3. Configure the strategy.
 - For the simple strategy type, select a strategy for each port.
 - For the row-based strategy type, select a strategy.
 - For the advanced strategy type, create an expression that uses consolidation functions.
4. In the Group By field, select the port that contains the group identifier.
5. Enable sorting on the **Advanced** view if the input data is not sorted.
6. Configure the output to contain consolidated records or all records.

Consolidation Transformation in a Non-native Environment

The Consolidation transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Consolidation Transformation on the Blaze Engine

The Consolidation transformation might process data differently in the native environment and in a non-native environment.

The transformation is supported with the following restrictions:

- The transformation might process records in a different order in each environment.
- The transformation might identify a different record as the survivor in each environment.

Consolidation Transformation on the Spark Engine

The Consolidation transformation might process data differently in the native environment and in a non-native environment.

The transformation is supported with the following restrictions:

- The transformation might process records in a different order in each environment.
- The transformation might identify a different record as the survivor in each environment.

Consolidation Transformation on the Databricks Spark Engine

The Consolidation transformation might process data differently in the native environment and in a non-native environment.

The transformation is supported with the following restrictions:

- The transformation might process records in a different order in each environment.
- The transformation might identify a different record as the survivor in each environment.

CHAPTER 12

Data Masking Transformation

This chapter includes the following topics:

- [Data Masking Transformation Overview, 194](#)
- [Masking Techniques, 194](#)
- [Masking Rules, 206](#)
- [Special Mask Formats, 210](#)
- [Default Value File, 214](#)
- [Data Masking Transformation Configuration, 215](#)
- [Data Masking Transformation Runtime Properties, 217](#)
- [Data Masking Example, 218](#)
- [Data Masking Transformation Advanced Properties, 220](#)
- [Data Masking Transformation in a Non-native Environment, 220](#)

Data Masking Transformation Overview

The Data Masking transformation changes sensitive production data to realistic test data for non-production environments. The Data Masking transformation modifies source data based on masking techniques that you configure for each column.

Create masked data for software development, testing, training, and data mining. You can maintain data relationships in the masked data and maintain referential integrity between database tables.

The Data Masking transformation provides masking rules based on the source datatype and masking technique that you configure for a column. For strings, you can restrict which characters in a string to replace. You can restrict which characters to apply in a mask. For numbers and dates, you can provide a range of numbers for the masked data. You can configure a range that is a fixed or percentage variance from the original number. The Data Integration Service replaces characters based on the locale that you configure for the transformation.

Masking Techniques

The masking technique is the type of data masking to apply to the selected column.

You can select one of the following masking techniques for an input column:

Random

Produces random, non-repeatable results for the same source data and masking rules. You can mask date, numeric, and string datatypes. Random masking does not require a seed value. The results of random masking are non-deterministic.

Expression

Applies an expression to a source column to create or mask data. You can mask all data types.

Key

Replaces source data with repeatable values. The Data Masking transformation produces deterministic results for the same source data, masking rules, and seed value. You can mask date, numeric, and string data types.

Substitution

Replaces a column of data with similar but unrelated data from a dictionary. You can mask the string data type.

Dependent

Replaces the values of one source column based on the values of another source column. You can mask the string data type.

Tokenization

Replaces source data with data generated based on customized masking criteria. The Data Masking transformation applies rules specified in a customized algorithm. You can mask the string data type.

Encryption

Replaces source data with encrypted values based on encryption criteria that you configure in the transformation. You can encrypt the string data type.

Special Mask Formats

Credit card number, email address, IP address, phone number, SSN, SIN, or URL. The Data Masking transformation applies built-in rules to intelligently mask these common types of sensitive data.

No Masking

The Data Masking transformation does not change the source data.

Default is No Masking.

Random Masking

Random masking generates random nondeterministic masked data. The Data Masking transformation returns different values when the same source value occurs in different rows. You can define masking rules that affect the format of data that the Data Masking transformation returns. Mask numeric, string, and date values with random masking.

Masking String Values

Configure random masking to generate random output for string columns. To configure limitations for each character in the output string, configure a mask format. Configure filter characters to define which source characters to mask and the characters to mask them with.

You can apply the following masking rules for a string port:

Range

Configure the minimum and maximum string length. The Data Masking transformation returns a string of random characters between the minimum and maximum string length.

Mask Format

Define the type of character to substitute for each character in the input data. You can limit each character to an alphabetic, numeric, or alphanumeric character type.

Source String Characters

Define the characters in the source string that you want to mask. For example, mask the number sign (#) character whenever it occurs in the input data. The Data Masking transformation masks all the input characters when Source String Characters is blank.

Result String Replacement Characters

Substitute the characters in the target string with the characters you define in Result String Characters. For example, enter the following characters to configure each mask to contain uppercase alphabetic characters A - Z:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Masking Numeric Values

When you mask numeric data, you can configure a range of output values for a column. The Data Masking transformation returns a value between the minimum and maximum values of the range depending on port precision. To define the range, configure the minimum and maximum ranges or configure a blurring range based on a variance from the original source value.

You can configure the following masking parameters for numeric data:

Range

Define a range of output values. The Data Masking transformation returns numeric data between the minimum and maximum values.

Blurring Range

Define a range of output values that are within a fixed variance or a percentage variance of the source data. The Data Masking transformation returns numeric data that is close to the value of the source data. You can configure a range and a blurring range.

Masking Date Values

To mask date values with random masking, either configure a range of output dates or choose a variance. When you configure a variance, choose a part of the date to blur. Choose the year, month, day, hour, minute, or second. The Data Masking transformation returns a date that is within the range you configure.

You can configure the following masking rules when you mask a datetime value:

Range

Sets the minimum and maximum values to return for the selected datetime value.

Blurring

Masks a date based on a variance that you apply to a unit of the date. The Data Masking transformation returns a date that is within the variance. You can blur the year, month, day, hour, minute, or second. Choose a low and high variance to apply.

Expression Masking

Expression masking applies an expression to a port to change the data or create new data. When you configure expression masking, create an expression in the Expression Editor. Select input and output ports, functions, variables, and operators to build expressions.

You can concatenate data from multiple ports to create a value for another port. For example, you need to create a login name. The source has first name and last name columns. Mask the first and last name from lookup files. In the Data Masking transformation, create another port called Login. For the Login port, configure an expression to concatenate the first letter of the first name with the last name:

```
SUBSTR (FIRSTNM, 1, 1) || LASTNM
```

Select functions, ports, variables, and operators from the point-and-click interface to minimize errors when you build expressions.

The Expression Editor displays the output ports which are not configured for expression masking. You cannot use the output from an expression as input to another expression. If you manually add the output port name to the expression, you might get unexpected results.

When you create an expression, verify that the expression returns a value that matches the port datatype. The Data Masking transformation returns zero if the data type of the expression port is numeric and the data type of the expression is not the same. The Data Masking transformation returns null values if the data type of the expression port is a string and the data type of the expression is not the same.

Repeatable Expression Masking

Configure repeatable expression masking when a source column occurs in more than one table and you need to mask the column from each table with the same value.

When you configure repeatable expression masking, the Data Masking transformation saves the results of an expression in a storage table. If the column occurs in another source table, the Data Masking transformation returns the masked value from the storage table instead of from the expression.

Dictionary Name

When you configure repeatable expression masking you must enter a dictionary name. The dictionary name is a key that allows multiple Data Masking transformations to generate the same masked values from the same source values. Define the same dictionary name in each Data Masking transformation. The dictionary name can be any text.

Storage Table

The storage table contains the results of repeatable expression masking between sessions. A storage table row contains the source column and a masked value pair. The storage table for expression masking is a separate table from the storage table for substitution masking.

Each time the Data Masking transformation masks a value with a repeatable expression, it searches the storage table by dictionary name, locale, column name and input value. If it finds a row in the storage table, it returns the masked value from the storage table. If the Data Masking transformation does not find a row, it generates a masked value from the expression for the column.

You need to encrypt storage tables for expression masking when you have unencrypted data in the storage and use the same dictionary name as key.

Encrypting Storage Tables for Expression Masking

You can use transformation language encoding functions to encrypt storage tables. You need to encrypt storage tables when you have enabled storage encryption.

1. Create a mapping with the IDM_EXPRESSION_STORAGE storage table as source.
2. Create a Data Masking transformation.
3. Apply the expression masking technique on the masked value ports.
4. Use the following expression on the MASKEDVALUE port:

```
Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))
```

5. Link the ports to the target.

Example

For example, an Employees table contains the following columns:

```
FirstName  
LastName  
LoginID
```

In the Data Masking transformation, mask LoginID with an expression that combines FirstName and LastName. Configure the expression mask to be repeatable. Enter a dictionary name as a key for repeatable masking.

The Computer_Users table contains a LoginID, but no FirstName or LastName columns:

```
Dept  
LoginID  
Password
```

To mask the LoginID in Computer_Users with the same LoginID as Employees, configure expression masking for the LoginID column. Enable repeatable masking and enter the same dictionary name that you defined for the LoginID Employees table. The Integration Service retrieves the LoginID values from the storage table.

Create a default expression to use when the Integration Service cannot find a row in the storage table for LoginID. The Computer_Users table does not have the FirstName or LastName columns, so the expression creates a less meaningful LoginID.

Storage Table Scripts

Informatica provides scripts that you can run to create the storage table. The scripts are in the following location:

```
<PowerCenter installation directory>\client\bin\Extensions\DataMasking
```

The directory contains a script for Sybase, Microsoft SQL Server, IBM DB2, and Oracle databases. Each script is named <Expression_<database type>.

Rules and Guidelines for Expression Masking

Use the following rules and guidelines for expression masking:

- You cannot use the output from an expression as input to another expression. If you manually add the output port name to the expression, you might get unexpected results.
- Use the point-and-click method to build expressions. Select functions, ports, variables, and operators from the point-and-click interface to minimize errors when you build expressions.
- If the Data Masking transformation is configured for repeatable masking, and the storage table does not exist, the Integration Service substitutes the source data with default values.

Key Masking

A column configured for key masking returns deterministic masked data each time the source value and seed value are the same. The Data Masking transformation returns unique values for the column.

When you configure a column for key masking, the Data Masking transformation creates a seed value for the column. You can change the seed value to produce repeatable data between different Data Masking transformations. For example, configure key masking to enforce referential integrity. Use the same seed value to mask a primary key in a table and the foreign key value in another table.

You can define masking rules that affect the format of data that the Data Masking transformation returns. Mask string and numeric values with key masking.

Masking String Values

You can configure key masking in order to generate repeatable output for strings. Configure a mask format to define limitations for each character in the output string. Configure source string characters that define which source characters to mask. Configure result string replacement characters to limit the masked data to certain characters.

You can configure the following masking rules for key masking strings:

Seed

Apply a seed value to generate deterministic masked data for a column. You can enter a number between 1 and 1,000.

Mask Format

Define the type of character to substitute for each character in the input data. You can limit each character to an alphabetic, numeric, or alphanumeric character type.

Source String Characters

Define the characters in the source string that you want to mask. For example, mask the number sign (#) character whenever it occurs in the input data. The Data Masking transformation masks all the input characters when Source String Characters is blank. The Data Masking transformation does not always return unique data if the number of source string characters is less than the number of result string characters.

Result String Characters

Substitute the characters in the target string with the characters you define in Result String Characters. For example, enter the following characters to configure each mask to contain all uppercase alphabetic characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Masking Numeric Values

Configure key masking for numeric source data to generate deterministic output. When you configure a column for numeric key masking, you assign a random seed value to the column. When the Data Masking transformation masks the source data, it applies a masking algorithm that requires the seed.

You can change the seed value for a column to produce repeatable results if the same source value occurs in a different column. For example, you want to maintain a primary-foreign key relationship between two tables. In each Data Masking transformation, enter the same seed value for the primary-key column as the seed value for the foreign-key column. The Data Masking transformation produces deterministic results for the same numeric values. The referential integrity is maintained between the tables.

Masking Datetime Values

When you can configure key masking for datetime values, the Data Masking transformation requires a random number as a seed. You can change the seed to match the seed value for another column in order to return repeatable datetime values between the columns.

The Data Masking transformation can mask dates between 1753 and 2400 with key masking. If the source year is in a leap year, the Data Masking transformation returns a year that is also a leap year. If the source month contains 31 days, the Data Masking transformation returns a month that has 31 days. If the source month is February, the Data Masking transformation returns February.

The Data Masking transformation always generates valid dates.

Substitution Masking

Substitution masking replaces a column of data with similar but unrelated data. Use substitution masking to replace production data with realistic test data. When you configure substitution masking, define the dictionary that contains the substitute values.

The Data Masking transformation performs a lookup on the dictionary that you configure. The Data Masking transformation replaces source data with data from the dictionary. Dictionary files can contain string data, datetime values, integers, and floating point numbers. Enter datetime values in the following format:

`mm/dd/yyyy`

You can substitute data with repeatable or non-repeatable values. When you choose repeatable values, the Data Masking transformation produces deterministic results for the same source data and seed value. You must configure a seed value to substitute data with deterministic results. The Integration Service maintains a storage table of source and masked values for repeatable masking.

You can substitute more than one column of data with masked values from the same dictionary row. Configure substitution masking for one input column. Configure dependent data masking for the other columns that receive masked data from the same dictionary row.

Dictionaries

A dictionary is a reference table that contains the substitute data and a serial number for each row in the table. Create a reference table for substitution masking from a flat file or relational table that you import into the Model repository.

The Data Masking transformation generates a number to retrieve a dictionary row by the serial number. The Data Masking transformation generates a hash key for repeatable substitution masking or a random number for non-repeatable masking. You can configure an additional lookup condition if you configure repeatable substitution masking.

You can configure a dictionary to mask more than one port in the Data Masking transformation.

When the Data Masking transformation retrieves substitution data from a dictionary, the transformation does not check if the substitute data value is the same as the original value. For example, the Data Masking transformation might substitute the name John with the same name (John) from a dictionary file.

The following example shows a dictionary table that contains first name and gender:

SNO	GENDER	FIRSTNAME
1	M	Adam
2	M	Adeel
3	M	Adil
4	F	Alice
5	F	Alison

In this dictionary, the first field in the row is the serial number, and the second field is gender. The Integration Service always looks up a dictionary record by serial number. You can add gender as a lookup condition if you configure repeatable masking. The Integration Service retrieves a row from the dictionary using a hash key, and it finds a row with a gender that matches the gender in the source data.

Use the following rules and guidelines when you create a reference table:

- Each record in the table must have a serial number.
- The serial numbers are sequential integers starting at one. The serial numbers cannot have a missing number in the sequence.
- The serial number column can be anywhere in a table row. It can have any label.

If you use a flat file table to create the reference table, use the following rules and guidelines:

- The first row of the flat file table must have column labels to identify the fields in each record. The fields are separated by commas. If the first row does not contain column labels, the Integration Service takes the values of the fields in the first row as column names.
- If you create a flat file table on Windows and copy it to a UNIX machine, verify that the file format is correct for UNIX. For example, Windows and UNIX use different characters for the end of line marker.

Storage Tables

The Data Masking transformation maintains storage tables for repeatable substitution between sessions. A storage table row contains the source column and a masked value pair. Each time the Data Masking transformation masks a value with a repeatable substitute value, it searches the storage table by dictionary name, locale, column name, input value, and seed. If it finds a row, it returns the masked value from the storage table. If the Data Masking transformation does not find a row, it retrieves a row from the dictionary with a hash key.

The dictionary name format in the storage table is different for a flat file dictionary and a relational dictionary. A flat file dictionary name is identified by the file name. The relational dictionary name has the following syntax:

```
<Connection object>_<dictionary table name>
```

Informatica provides scripts that you can run to create a relational storage table. The scripts are in the following location:

```
<PowerCenter Client installation directory>\client\bin\Extensions\DataMasking
```

The directory contains a script for Sybase, Microsoft SQL Server, IBM DB2, and Oracle databases. Each script is named Substitution_<database type>. You can create a table in a different database if you configure the SQL statements and the primary key constraints.

You need to encrypt storage tables for substitution masking when you have unencrypted data in the storage and use the same seed value and dictionary to encrypt the same columns.

Encrypting Storage Tables for Substitution Masking

You can use transformation language encoding functions to encrypt storage tables. You need to encrypt storage tables when you have enabled storage encryption.

1. Create a mapping with the `IDM_SUBSTITUTION_STORAGE` storage table as source.
2. Create a Data Masking transformation.
3. Apply the substitution masking technique on the input value and the masked value ports.
4. Use the following expression on the `INPUTVALUE` port:

```
Enc_Base64(AES_Encrypt(INPUTVALUE, Key))
```

5. Use the following expression on the `MASKEDVALUE` port:

```
Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))
```

6. Link the ports to the target.

Substitution Masking Properties

You can configure the following masking rules for substitution masking:

- **Repeatable Output.** Returns deterministic results between sessions. The Data Masking transformation stores masked values in the storage table.
- **Seed Value.** Apply a seed value to generate deterministic masked data for a column. Enter a number between 1 and 1,000.
- **Unique Output.** Force the Data Masking transformation to create unique output values for unique input values. No two input values are masked to the same output value. The dictionary must have enough unique rows to enable unique output.
When you disable unique output, the Data Masking transformation might not mask input values to unique output values. The dictionary might contain fewer rows.
- **Unique Port.** The port used to identify unique records for substitution masking. For example, you want to mask first names in a table called `Customer`. If you select the table column that contains the first names as the unique port, the data masking transformation replaces duplicate first names with the same masked value. If you select the `Customer_ID` column as the unique port, the data masking transformation replaces each first name with a unique value.
- **Optimize Dictionary Usage.**
Applicable if you select the **Repeatable Output** option. Increases the usage of masked values from a dictionary.
- **Dictionary Information.** Configure the reference table that contains the substitute data values. Click **Select Source** to select a reference table.
 - **Dictionary Name.** Displays the name of the reference table that you select.
 - **Dictionary Connection.** Displays the name of the connection that contains the dictionary.
 - **Serial Number Column.** Choose the column to return to the Data Masking transformation.
 - **Sort Column.** The dictionary column on which you want to sort entries. Specify a sort column to generate deterministic results even if the order of entries in the dictionary changes. For example, if you move a relational dictionary and the order of entries changes, sort on the serial number column to consistently mask the data.

Note: The column that you choose must contain unique values. You cannot use a column that might contain duplicate values to sort the data.

- **Output Column.** Choose the column to return to the Data Masking transformation.
- **Lookup condition.** Configure a lookup condition to further qualify what dictionary row to use for substitution masking. The lookup condition is similar to the WHERE clause in an SQL query. When you configure a lookup condition you compare the value of a column in the source with a column in the dictionary.

For example, you want to mask the first name. The source data and the dictionary have a first name column and a gender column. You can add a condition that each female first name is replaced with a female name from the dictionary. The lookup condition compares gender in the source to gender in the dictionary.

- **Input port.** Source data column to use in the lookup.
- **Dictionary column.** Dictionary column to compare the input port to.

Rules and Guidelines for Substitution Masking

Use the following rules and guidelines for substitution masking:

- If a storage table does not exist for a unique repeatable substitution mask, the session fails.
- If the dictionary contains no rows, the Data Masking transformation returns an error message.
- When the Data Masking transformation finds an input value with the locale, dictionary, and seed in the storage table, it retrieves the masked value, even if the row is no longer in the dictionary.
- If you delete a connection object or modify the dictionary, truncate the storage table. Otherwise, you might get unexpected results.
- If the number of values in the dictionary is less than the number of unique values in the source data, the Data Masking Transformation cannot mask the data with unique repeatable values. The Data Masking transformation returns an error message.

Dependent Masking

Dependent masking substitutes multiple columns of source data with data from the same dictionary row.

When the Data Masking transformation performs substitution masking for multiple columns, the masked data might contain unrealistic combinations of fields. You can configure dependent masking in order to substitute data for multiple input columns from the same dictionary row. The masked data receives valid combinations such as, "New York, New York" or "Chicago, Illinois."

When you configure dependent masking, you first configure an input column for substitution masking. Configure other input columns to be dependent on that substitution column. For example, you choose the ZIP code column for substitution masking and choose city and state columns to be dependent on the ZIP code column. Dependent masking ensures that the substituted city and state values are valid for the substituted ZIP code value.

Note: You cannot configure a column for dependent masking without first configuring a column for substitution masking.

Configure the following masking rules when you configure a column for dependent masking:

Dependent column

The name of the input column that you configured for substitution masking. The Data Masking transformation retrieves substitute data from a dictionary using the masking rules for that column. The

column you configure for substitution masking becomes the key column for retrieving masked data from the dictionary.

Output column

The name of the dictionary column that contains the value for the column you are configuring with dependent masking.

Dependent Masking Example

A data masking dictionary might contain address rows with the following values:

SNO	STREET	CITY	STATE	ZIP	COUNTRY
1	32 Apple Lane	Chicago	IL	61523	US
2	776 Ash Street	Dallas	TX	75240	US
3	2229 Big Square	Atleeville	TN	38057	US
4	6698 Cowboy Street	Houston	TX	77001	US

You need to mask source data with valid combinations of the city, state, and ZIP code from the Address dictionary.

Configure the ZIP port for substitution masking. Enter the following masking rules for the ZIP port:

Rule	Value
Dictionary Name	Address
Serial Number Column	SNO
Output Column	ZIP

Configure the City port for dependent masking. Enter the following masking rules for the City port:

Rule	Value
Dependent Column	ZIP
Output Column	City

Configure the State port for dependent masking. Enter the following masking rules for the State port:

Rule	Value
Dependent Column	ZIP
Output Column	State

When the Data Masking transformation masks the ZIP code, it returns the correct city and state for the ZIP code from the dictionary row.

Tokenization Masking

Use the tokenization masking technique to mask source string data based on criteria that you specify in an algorithm. For example, you can create an algorithm that contains a fake email address to replace field entries in the source data.

You can configure the format of the masked data using Tokenization masking. You must assign a tokenizer name to the masking algorithm before you can use it. The tokenizer name references the masking algorithm (JAR) used. Specify the tokenizer name when you apply the tokenization masking technique.

Configuring Tokenization Masking

Perform the following tasks before you use the tokenization masking technique:

1. Browse to the `tokenprovider` directory in the path: `<Informatica_home>\services\shared`.
2. Open the following XML file: `com.informatica.products.ilm.tx-tokenizerprovider.xml`.
3. Add the tokenizer name and the fully qualified name of the class file for each tokenizer you want to use. Implement the tokenizer class within the `com.informatica.products.ilm.tx-tokenprovider-<Build-Number>.jar` class in the `tokenprovider` directory. For each tokenizer, enter the information in the XML file as in the following example:

```
<TokenizerProvider>
  <Tokenizer Name="CCTokenizer"
  ClassName="com.informatica.tokenprovider.CCTokenizer"/>
</TokenizerProvider>
```

Where:

- `Tokenizer Name` is the user-defined name in quotes.
- `ClassName` is the user-defined name for the `CLASSNAME` attribute. Implement this from within `com.informatica.products.ilm.tx-tokenprovider-<Build-Number>.jar`.

After configuration, you can use the Tokenization masking technique. Enter the tokenizer name to specify the algorithm to use when you create a mapping.

Encryption

Encryption masking applies encryption algorithms to mask source data.

Mask string data types with encryption masking.

You can choose to preserve the format and length of the source data or the length of the source data. You can also choose to change the format and length of the source data after encryption.

You can choose characters that you do not want to encrypt.

After you encrypt the source data, you can also decrypt it to get back the original data. To decrypt the data, you must create and run a mapping that uses the same encryption technique with the same pass phrase that you used to encrypt the source data. Set the mode to `Decryption`.

Note: If the source data contains UTF-8 four byte characters, you cannot use encryption to mask the data.

Select one of the following encryption techniques:

Preserve Format and Metadata

Use the `Preserve Format and Metadata` encryption option to preserve the format and the length of the source data. When you choose to preserve format and metadata, all uppercase characters are replaced with uppercase characters, lowercase characters are replaced with lowercase characters, numbers are replaced with numbers, and special characters are replaced with special characters after encryption. For example, an email address `Abc123@xyz.com` might become `Mpz849#dje!kuw`. In this example, if you

configure "@" and "." characters as Do Not Encrypt Characters, the email might become Mpz849@dje.kuw.

Preserve Metadata

Use the Preserve Metadata encryption option to preserve the length of the source data. When you choose to preserve metadata, the length of the data remains the same after encryption. For example, a first name Alexander might become jl6#HB91v, where the length remains the same as in the source data.

Change Metadata

Use the Change Metadata encryption option to change the length of the source data after encryption. When you choose to change metadata, the encrypted data does not retain the length and format of the source data. For example, a city name London might become Xuep@8f5, fmch529, or 6ky#ke33h*we.

Note: Before you use the Change Metadata encryption option, you must change the precision of the column you want to apply encryption on in the database.

Use the following formula to calculate the precision and round up the value to the next higher integer:

$$\text{Required Precision} = (1.33 * \text{Original Precision}) + 24$$

After you change the column precision in the database, you must update the column precision in the mapping. To update the column precision you can either reimport the metadata from the updated database, or manually change the column precision in each transformation in the mapping.

Masking Rules

Masking rules are the options that you configure after you choose the masking technique.

When you choose the random or key masking technique, you can configure the mask format, the source string characters, and the result string characters. You can configure range or blurring with random masking.

The following table describes the masking rules that you can configure for each masking technique:

Masking Rule	Description	Masking Technique	Source Datatype
Mask Format	Mask that limits each character in an output string to alphabetic, numeric, or alphanumeric character.	Random and Key	String
Source String Characters	Set of source characters to mask or to exclude from masking.	Random and Key	String
Result String Replacement Characters	A set of characters to include or exclude in a mask.	Random and Key	String

Masking Rule	Description	Masking Technique	Source Datatype
Range	<p>A range of output values.</p> <ul style="list-style-type: none"> - Numeric. The Data Masking transformation returns numeric data between the minimum and maximum values. - String. Returns a string of random characters between the minimum and maximum string length. - Date/Time. Returns a date and time within the minimum and maximum datetime. 	Random	Numeric String Date/Time
Blurring	<p>Range of output values with a fixed or percent variance from the source data. The Data Masking transformation returns data that is close to the value of the source data. Datetime columns require a fixed variance. columns require a fixed variance.</p>	Random	Numeric Date/Time

Mask Format

Configure a mask format to limit each character in the output column to an alphabetic, numeric, or alphanumeric character. Use the following characters to define a mask format:

A, D, N, X, +, R

Note: The mask format contains uppercase characters. When you enter a lowercase mask character, the Data Masking transformation converts the character to uppercase.

The following table describes mask format characters:

Character	Description
A	Alphabetical characters. For example, ASCII characters a to z and A to Z.
D	Digits 0 to 9. The data masking transformation returns an "X" for characters other than digits 0 to 9.
N	Alphanumeric characters. For example, ASCII characters a to z, A to Z, and 0-9.
X	Any character. For example, alphanumeric or symbol.
+	No masking.
R	Remaining characters. R specifies that the remaining characters in the string can be any character type. R must appear as the last character of the mask.

For example, a department name has the following format:

nnn-<department_name>

You can configure a mask to force the first three characters to be numeric, the department name to be alphabetic, and the dash to remain in the output. Configure the following mask format:

DDD+AAAAAAAAAAAAAAAA

The Data Masking transformation replaces the first three characters with numeric characters. It does not replace the fourth character. The Data Masking transformation replaces the remaining characters with alphabetical characters.

If you do not define a mask format, the Data Masking transformation replaces each source character with any character. If the mask format is longer than the input string, the Data Masking transformation ignores the extra characters in the mask format. If the mask format is shorter than the source string, the Data Masking transformation masks the remaining characters with format R.

Note: You cannot configure a mask format with the range option.

Source String Characters

Source string characters are source characters that you choose to mask or not mask. The position of the characters in the source string does not matter. The source characters are case sensitive.

You can configure any number of characters. When Characters is blank, the Data Masking transformation replaces all the source characters in the column.

Select one of the following options for source string characters:

Mask Only

The Data Masking transformation masks characters in the source that you configure as source string characters. For example, if you enter the characters A, B, and c, the Data Masking transformation replaces A, B, or c with a different character when the character occurs in source data. A source character that is not an A, B, or c does not change. The mask is case sensitive.

Mask All Except

Masks all characters except the source string characters that occur in the source string. For example, if you enter the filter source character “-” and select Mask All Except, the Data Masking transformation does not replace the “-” character when it occurs in the source data. The rest of the source characters change.

Source String Example

A source file has a column named Dependents. The Dependents column contains more than one name separated by commas. You need to mask the Dependents column and keep the comma in the test data to delimit the names.

For the Dependents column, select Source String Characters. Choose Don't Mask and enter “,” as the source character to skip. Do not enter quotes.

The Data Masking transformation replaces all the characters in the source string except for the comma.

Result String Replacement Characters

Result string replacement characters are characters you choose as substitute characters in the masked data. When you configure result string replacement characters, the Data Masking transformation replaces characters in the source string with the result string replacement characters. To avoid generating the same output for different input values, configure a wide range of substitute characters, or mask only a few source characters. The position of each character in the string does not matter.

Select one of the following options for result string replacement characters:

Use Only

Mask the source with only the characters you define as result string replacement characters. For example, if you enter the characters A, B, and c, the Data Masking transformation replaces every character in the source column with an A, B, or c. The word “horse” might be replaced with “BAcBA.”

Use All Except

Mask the source with any characters except the characters you define as result string replacement characters. For example, if you enter A, B, and c result string replacement characters, the masked data never has the characters A, B, or c.

Result String Replacement Characters Example

To replace all commas in the Dependents column with semicolons, complete the following tasks:

1. Configure the comma as a source string character and select Mask Only.
The Data Masking transformation masks only the comma when it occurs in the Dependents column.
2. Configure the semicolon as a result string replacement character and select Use Only.
The Data Masking transformation replaces each comma in the Dependents column with a semicolon.

Range

Define a range for numeric, date, or string data. When you define a range for numeric or date values the Data Masking transformation masks the source data with a value between the minimum and maximum values. When you configure a range for a string, you configure a range of string lengths.

String Range

When you configure random string masking, the Data Masking transformation generates strings that vary in length from the length of the source string. Optionally, you can configure a minimum and maximum string width. The values you enter as the maximum or minimum width must be positive integers. Each width must be less than or equal to the port precision.

Numeric Range

Set the minimum and maximum values for a numeric column. The maximum value must be less than or equal to the port precision. The default range is from one to the port precision length.

Date Range

Set minimum and maximum values for a datetime value. The minimum and maximum fields contain the default minimum and maximum dates. The default datetime format is MM/DD/YYYY HH24:MI:SS. The maximum datetime must be later than the minimum datetime.

Blurring

Blurring creates an output value within a fixed or percent variance from the source data value. Configure blurring to return a random value that is close to the original value. You can blur numeric and date values.

Blurring Numeric Values

Select a fixed or percent variance to blur a numeric source value. The low blurring value is a variance below the source value. The high blurring value is a variance above the source value. The low and high values must be greater than or equal to zero. When the Data Masking transformation returns masked data, the numeric data is within the range that you define.

The following table describes the masking results for blurring range values when the input source value is 66:

Blurring Type	Low	High	Result
Fixed	0	10	Between 66 and 76
Fixed	10	0	Between 56 and 66
Fixed	10	10	Between 56 and 76
Percent	0	50	Between 66 and 99
Percent	50	0	Between 33 and 66
Percent	50	50	Between 33 and 99

Blurring Date Values

Mask a date as a variance of the source date by configuring blurring. Select a unit of the date to apply the variance to. You can select the year, month, day, or hour. Enter the low and high bounds to define a variance above and below the unit in the source date. The Data Masking transformation applies the variance and returns a date that is within the variance.

For example, to restrict the masked date to a date within two years of the source date, select year as the unit. Enter two as the low and high bound. If a source date is 02/02/2006, the Data Masking transformation returns a date between 02/02/2004 and 02/02/2008.

By default, the blur unit is year.

Special Mask Formats

Special mask formats are masks that you can apply to common types of data. With a special mask format, the Data Masking transformation returns a masked value that has a realistic format, but is not a valid value.

For example, when you mask an SSN, the Data Masking transformation returns an SSN that is the correct format but is not valid. You can configure repeatable masking for Social Security numbers.

Configure special masks for the following types of data:

- Social Security numbers
- Credit card numbers
- Phone numbers
- URL addresses
- Email addresses
- IP Addresses
- Social Insurance Numbers

When the source data format or datatype is invalid for a mask, the Data Integration Service applies a default mask to the data. The Integration Service applies masked values from the default value file. You can edit the default value file to change the default values.

Credit Card Number Masking

The Data Masking transformation generates a logically valid credit card number when it masks a valid credit card number. The length of the source credit card number must be between 13 to 19 digits. The input credit card number must have a valid checksum based on credit card industry rules.

The source credit card number can contain numbers, spaces, and hyphens. If the credit card has incorrect characters, or is the wrong length, the Integration Service writes an error to the session log. The Integration Service applies a default credit card number mask when the source data is invalid.

The Data Masking transformation does not mask the six digit Bank Identification Number (BIN). For example, the Data Masking transformation might mask the credit card number 4539 1596 8210 2773 as 4539 1516 0556 7067. The Data Masking transformation creates a masked number that has a valid checksum.

Email Address Masking

Use the Data Masking transformation to mask the email address that contains string value. The Data Masking transformation can mask an email address with random ASCII characters or replace the email address with a realistic email address.

You can apply the following types of masking with email address:

Standard email masking

The Data Masking transformation returns random ASCII characters when it masks an email address. For example, the Data Masking transformation can mask Georgesmith@yahoo.com as KtrlupQAPyk@vdSKh.BIC. Default is standard.

Advanced email masking

The Data Masking transformation masks the email address with another realistic email address derived from the transformation output ports or dictionary columns.

Advanced Email Masking

With the advanced email masking type, you can mask the email address with another realistic email address. The Data Masking transformation creates the email address from the dictionary columns or from the transformation output ports.

You can create the local part in the email address from mapping output ports. Or you can create the local part in the email address from relational table or flat file columns.

The Data Masking transformation can create the domain name for the email address from a constant value or from a random value in the domain dictionary.

You can create an advanced email masking based on the following options:

Email Address Based on Dependent Ports

You can create an email address based on the Data Masking transformation output ports. Select the transformation output ports for the first name and the last name column. The Data Masking transformation masks the first name, the last name, or both names based on the values you specify for the first and last name length.

Email Address Based on a Dictionary

You can create an email address based on the columns from a dictionary. Select a reference table as the source for the dictionary.

Select the dictionary columns for the first name and the last name. The Data Masking transformation masks the first name, the last name, or both names based on the values you specify for the first and last name length.

Configuration Parameters for an Advanced Email Address Masking Type

Specify configuration parameters when you configure advanced email address masking.

You can specify the following configuration parameters:

Delimiter

You can select a delimiter, such as a dot, hyphen, or underscore, to separate the first name and last name in the email address. If you do not want to separate the first name and last name in the email address, leave the delimiter as blank.

FirstName Column

Select a Data Masking transformation output port or a dictionary column to mask the first name in the email address.

LastName Column

Select a Data Masking transformation output port or a dictionary column to mask the last name in the email address.

Length for the FirstName or LastName columns

Restricts the character length to mask for the first name and the last name columns. For example, the input data is Timothy for the first name and Smith for the last name. Select 5 as the length of the first name column. Select 1 as the length of the last name column with a dot as the delimiter. The Data Masking transformation generates the following email address:

```
timot.s@<domain_name>
```

DomainName

You can use a constant value, such as gmail.com, for the domain name. Or, you can specify another dictionary file that contains a list of domain names. The domain dictionary can be a flat file or a relational table.

IP Address Masking

The Data Masking transformation masks an IP address as another IP address by splitting it into four numbers, separated by a period. The first number is the network. The Data Masking transformation masks the network number within the network range.

The Data Masking transformation masks a Class A IP address as a Class A IP Address and a 10.x.x.x address as a 10.x.x.x address. The Data Masking transformation does not mask the class and private network address. For example, the Data Masking transformation can mask 11.12.23.34 as 75.32.42.52. and 10.23.24.32 as 10.61.74.84.

Note: When you mask many IP addresses, the Data Masking transformation can return nonunique values because it does not mask the class or private network of the IP addresses.

Phone Number Masking

The Data Masking transformation masks a phone number without changing the format of the original phone number. For example, the Data Masking transformation can mask the phone number (408)382 0658 as (607)256 3106.

The source data can contain numbers, spaces, hyphens, and parentheses. The Integration Service does not mask alphabetic or special characters.

The Data Masking transformation can mask string, integer, and bigint data.

Social Security Number Masking

The Data Masking transformation generates a Social Security number that is not valid based on the latest High Group List from the Social Security Administration. The High Group List contains valid numbers that the Social Security Administration has issued.

The default High Group List is a text file in the following location:

```
<Installation Directory>\infa_shared\SrcFiles\highgroup.txt
```

To use the High Group List file in workflows, copy the text file to the source directory that you configure for the Data Integration Service.

The Data Masking transformation generates SSN numbers that are not on the High Group List. The Social Security Administration updates the High Group List every month. Download the latest version of the list from the following location:

```
http://www.socialsecurity.gov/employer/ssns/highgroup.txt
```

Social Security Number Format

The Data Masking transformation accepts any SSN format that contains nine digits. The digits can be delimited by any set of characters. For example, the Data Masking transformation accepts the following format: `+54-*9944$#789-,*()`.

Area Code Requirement

The Data Masking transformation returns a Social Security Number that is not valid with the same format as the source. The first three digits of the SSN define the area code. The Data Masking transformation does not mask the area code. It masks the group number and serial number. The source SSN must contain a valid area code. The Data Masking transformation locates the area code on the High Group List and determines a range of unused numbers that it can apply as masked data. If the SSN is not valid, the Data Masking transformation does not mask the source data.

Repeatable Social Security Number Masking

The Data Masking transformation returns deterministic Social Security numbers with repeatable masking. The Data Masking transformation cannot return all unique Social Security numbers because it cannot return valid Social Security numbers that the Social Security Administration has issued.

URL Address Masking

The Data Masking transformation parses a URL by searching for the `://` string and parsing the substring to the right of it. The source URL must contain the `://` string. The source URL can contain numbers and alphabetic characters.

The Data Masking transformation does not mask the protocol of the URL. For example, if the URL is `http://www.yahoo.com`, the Data Masking transformation can return `http://MgLaHjCa.VsD/`. The Data Masking transformation can generate a URL that is not valid.

Note: The Data Masking transformation always returns ASCII characters for a URL.

Social Insurance Number Masking

The Data Masking transformation masks a Social Insurance number that is nine digits. The digits can be delimited by any set of characters.

If the number contains no delimiters, the masked number contains no delimiters. Otherwise the masked number has the following format:

```
xxx-xxx-xxx
```

Repeatable SIN Numbers

You can configure the Data Masking transformation to return repeatable SIN values. When you configure a port for repeatable SIN masking, the Data Masking transformation returns deterministic masked data each time the source SIN value and seed value are the same.

To return repeatable SIN numbers, enable **Repeatable Values** and enter a seed number. The Data Masking transformation returns unique values for each SIN.

SIN Start Digit

You can define the first digit of the masked SIN.

Enable **Start Digit** and enter the digit. The Data Masking transformation creates masked SIN numbers that start with the number that you enter.

Default Value File

When the source data format or datatype is invalid for a mask, the Data Integration Service applies a default mask to the data. The Integration Service applies masked values from the default value file. You can edit the default value file to change the default values.

The default value file is an XML file in the following location:

```
<Installation Directory>\infa_shared\SrcFiles\defaultValue.xml
```

To use the default value file in workflows, copy the default value file to the source directory that you configure for the Data Integration Service.

The `defaultValue.xml` file contains the following name-value pairs:

```
<?xml version="1.0" standalone="yes" ?>
<defaultValue
  default_char = "X"
  default_digit = "9"
  default_date = "11/11/1111 00:00:00"
  default_email = "abc@xyz.com"
  default_ip = "99.99.9.999"
  default_url = "http://www.xyz.com"
  default_phone = "999 999 999 9999"
```

```
default_ssn = "999-99-9999"  
default_cc = "9999 9999 9999 9999"  
default_sin = "999-999-999"  
default_seed = "500"/>
```

RELATED TOPICS:

- [“Configure the Data Integration Service” on page 215](#)

Data Masking Transformation Configuration

Use the following steps to configure a Data Masking transformation.

1. Configure execution options for the Data Integration Service.
2. Create the transformation.
3. Define the input ports.
4. Configure masking rules for each port that you want to change.
5. Preview the data to verify the results.

Configure the Data Integration Service

You can configure execution options for the Data Integration Service in Informatica Administrator (the Administrator tool).

Configure execution options to set the following default directories:

- Home directory. Contains the source directory and the cache directory.
- Source directory. Contains source files for workflows. For example, the source directory can contain the highgrp.txt and defaultvalue.xml files.
- Cache directory. Contains cache files for substitution masking.

To set values for the execution options, open the Administrator tool, and select the Data Integration Service in the **Domain Navigator**. Click the **Properties** view, and then click **Edit** in the **Execution Options** section.

RELATED TOPICS:

- [“Default Value File” on page 214](#)

Creating a Data Masking Transformation

Create a Data Masking transformation in the Developer tool.

Before you create the Data Masking transformation, create the source. Import a flat file or relational database table as a physical data object.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
The **New** dialog box appears.
3. Select the Data Masking transformation.
4. Click **Next**.

5. Enter a name for the transformation.
6. Click **Finish**.
The transformation appears in the editor.

Defining the Ports

Add Data Masking input ports on the **Overview** view. When you create an input port, the Developer tool creates a corresponding output port by default. The output port has the same name as the input port.

1. On the **Overview** view, click **New** to add a port.
2. Configure the datatype, precision, and scale for the column.
You must configure the column datatype before you define masking rules for the column.
3. To configure data masking for the port, click the arrow in the masking type column of the **Overview** view.

Configuring Data Masking for Each Port

Select a masking technique and the corresponding masking rules for a port on the Data Masking dialog box. The Data Masking dialog box appears when you click the data masking column on the **Ports** tab.

1. Enable **Apply Masking** in order to configure masking for the selected port.
The Developer tool displays a list of masking techniques that you can use based on the datatype of the port you are masking.
2. Select a masking technique from the list.
The Developer tool displays different masking rules based on the masking technique that you choose. Some of the special mask formats have no masking rules to configure.
3. Configure the masking rules.
4. Click **OK** to apply the data masking configuration for the port.
When you define data masking for a port, the Developer tool creates an output port called **out-<port name>**. The **<port name>** is the same name as the input port. The Data Masking transformation returns the masked data in the **out-<port name>** port.

Previewing the Masked Data

You can compare the masked data to the original data when you view the Data Masking transformation results in the **Data Viewer**.

1. After you configure the Data Masking transformation ports and masking rules, create a mapping that includes the physical data object source and the Data Masking transformation.
2. Connect the source to the Data Masking transformation.
3. Verify that the source has data in a shared location that the Data Integration Service can access.
4. Click the Data Masking transformation to select it in the mapping.
5. Click **Data Viewer** and click **Run**.
The Developer tool displays data for all the Data Masking transformation output ports. The ports that have the **output** prefix contain the masked data. You can compare the masked data to the original data in the **Data Viewer** view.

Data Masking Transformation Runtime Properties

You can configure Data Masking transformation runtime properties to increase performance.

Configure the following runtime properties:

Storage Connection Name

Required for masking types that use a storage connection. The name of the connection that you use as the storage connection.

Cache Size

The size of the dictionary cache in main memory. Increase the memory size in order to improve performance. Minimum recommended size is 32 MB for 100,000 records. Default is 8 MB.

Cache Directory

The location of the dictionary cache. You must have write permissions for the directory. Default is CacheDir.

Passphrase

Required for encryption masking. The passphrase generates a key to encrypt or decrypt data. In mappings that you create in Test Data Management, the passphrase is encrypted. For mappings that you create in the Developer tool, you can use the site key to encrypt the passphrase and enter the encrypted value in the text box.

Mode

Required for encryption masking. Determines whether the Data Masking transformation performs encryption or decryption of the data. Set the value to Encryption to encrypt source data. To decrypt the masked data and return the original source data, run a mapping with the same encryption technique configuration and passphrase, with the mode set to Decryption.

Use SoftHSM

Required for encryption. Choose whether to use SoftHSM during encryption. SoftHSM is more secure, but you might notice a difference in performance.

Shared Storage Table

Enables sharing of the storage table between Data Masking transformation instances. Enable Shared Storage Table when two Data Masking transformation instances use the same dictionary column for the database connection, seed value, and locale. You can also enable the shared storage table when two ports in the same Data Masking transformation use the same dictionary column for the connection, seed, and locale. Disable the shared storage table when Data Masking transformations or ports do not share the dictionary column. Default is disabled.

Storage Table Commit Interval

The number of rows to commit at a time to the storage table. Increase the value to increase performance. Configure the commit interval when you do not configure the shared storage table. Default is 100,000.

Encrypt Storage

Encrypts storage tables, such as IDM_SUBSTITUTION_STORAGE and IDM_EXPRESSION_STORAGE. Verify that you have encrypted data in storage tables before you enable the encrypt storage property. Clear this option if you do not want to encrypt the storage tables. Default is disabled.

Storage Encryption Key

The Data Masking transformation encrypts the storage based on the storage encryption key. Use the same encryption key for each session run of the same Data Masking transformation instance.

Substitution Dictionary Owner Name

Name of the substitution dictionary table owner when you select the substitution masking type. If the database user specified in the database connection is not the owner of the substitution dictionary table in a session, you need to specify the table owner.

Storage Owner Name

Name of the table owner for `IDM_SUBSTITUTION_STORAGE` or `IDM_EXPRESSION_STORAGE` when you select repeatable expression or unique repeatable substitution masking type.

Data Masking Example

A developer needs to create test data for customer applications. The data must contain realistic customer data that other developers can access in the company development environment.

The developer creates a data service that returns masked customer data such as the the customer ID, credit card number, and income. The mapping includes a Data Masking transformation that transforms the customer data.

The following figure shows the mapping:



The mapping has the following transformations:

- `Read_Customer_Data`. Contains customer credit card and income information.
- `Customer_Data_Masking` transformation. Masks all columns except `FirstName` and `LastName`. The Data Masking transformation passes the masked columns to the target.
- `Customer_TestData`. Output transformation that receives the masked customer data.

Read_Customer Data

The customer data contains the following columns:

Column	Datatype
CustomerID	Integer
LastName	String
FirstName	String
CreditCard	String

Column	Datatype
Income	Integer
Join_Date	Datetime (MM/DD/YYYY)

The following table shows sample customer data:

CustomerID	LastName	FirstName	CreditCard	Income	JoinDate
0095	Bergeron	Barbara	4539-1686-3069-3957	12000	12/31/1999
0102	Brosseau	Derrick	5545-4091-5232-8948	4000	03/03/2011
0105	Anderson	Lauren	1234-5678-9012-3456	5000	04/03/2009
0106	Boonstra	Pauline	4217-9981-5613-6588	2000	07/07/2007
0107	Chan	Brian	4533-3156-8865-3156	4500	06/18/1995

Customer Data Masking Transformation

The Data Masking transformation masks all columns in the customer row except first and last name.

The Data Masking transformation performs the following types of masking:

- Key masking
- Random masking
- Credit card masking

The following table shows the masking rules for each port in the Data Masking transformation:

Input Port	Masking Type	Masking Rules	Description
CustomerID	Key	The seed is 934. The customer ID does not have any mask format. Result string replacement characters are 1234567890.	The CustomerID mask is deterministic. The masked customer ID contains numbers.
LastName	No Masking	-	-
FirstName	No Masking	-	-
CreditCard	CreditCard	-	The Data Masking transformation masks the credit card number with another number that has a valid checksum.
Income	Random	Blurring Percent Low bound=1 High bound=10	The masked income is within ten percent of the source income.
JoinDate	Random	Blurring Unit=Year Low Bound =5 HighBound=5	The masked date is within 5 years of the original date.

Customer Test Data Results

The Customer_TestData transformation receives realistic customer data from the Data Masking transformation.

The Customer_TestData target receives the following data:

out-CustomerID	out-LastName	outFirstName	out-CreditCard	out-Income	out-JoinDate
3954	Bergeron	Barbara	4539-1625-5074-4106	11500	03/22/2001
3962	Brosseau	Derrick	5545-4042-8767-5974	4300	04/17/2007
3964	Anderson	Lauren	1234-5687-2487-9053	5433	09/13/2006
3965	Boonstra	Pauline	4217-9935-7437-4879	1820	02/03/2010
3966	Chan	Brian	4533-3143-4061-8001	4811	10/30/2000

The income is within ten percent of the original income. The join date is within five years of the original date.

Data Masking Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Data Masking transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Data Masking Transformation in a Non-native Environment

The Data Masking transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Not supported.

Data Masking Transformation on the Blaze Engine

The Data Masking transformation is supported with the following restrictions.

Mapping validation fails in the following situations:

- The transformation is configured for repeatable expression masking.

- The transformation is configured for unique repeatable substitution masking.

You can use the following masking techniques on this engine:

Credit Card
Email
Expression
IP Address
Key
Phone
Random
SIN
SSN
Tokenization
URL
Random Substitution
Repeatable Substitution
Dependent with Random Substitution
Dependent with Repeatable Substitution

Data Masking Transformation on the Spark Engine

The Data Masking transformation is supported with the following restrictions.

Mapping validation fails in the following situations:

- The transformation is configured for repeatable expression masking.
- The transformation is configured for unique repeatable substitution masking.

You can use the following masking techniques on this engine:

Credit Card
Email
Expression
IP Address
Key
Phone
Random
SIN
SSN
Tokenization
URL
Random Substitution
Repeatable Substitution
Dependent with Random Substitution

Dependent with Repeatable Substitution

To optimize performance of the Data Masking transformation, configure the following Spark engine configuration properties in the Hadoop connection:

spark.executor.cores

Indicates the number of cores that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.cores=1`

spark.executor.instances

Indicates the number of instances that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.instances=1`

spark.executor.memory

Indicates the amount of memory that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.memory=3G`

Data Masking Transformation in a Streaming Mapping

Streaming mappings have the same processing rules as batch mappings on the Spark engine.

CHAPTER 13

Data Processor Transformation

This chapter includes the following topics:

- [Data Processor Transformation Overview, 223](#)
- [Data Processor Transformation Views, 224](#)
- [Data Processor Transformation Ports, 225](#)
- [Startup Component, 227](#)
- [References, 227](#)
- [Data Processor Transformation Settings, 228](#)
- [Events, 235](#)
- [Logs, 236](#)
- [Data Processor Transformation Development, 238](#)
- [Data Processor Transformation Export and Import, 242](#)
- [Data Processor Transformation Validation, 244](#)
- [Data Processor Transformation in a Non-native Environment, 245](#)

Data Processor Transformation Overview

The Data Processor transformation processes unstructured and semi-structured file formats in a mapping. Configure the transformation to process messaging formats, HTML pages, XML, JSON, and PDF documents. You can also convert structured formats such as ACORD, HIPAA, HL7, EDI-X12, EDIFACT, and SWIFT.

A mapping uses a Data Processor transformation to change documents from one format to another. The Data Processor transformation processes files of any format in a mapping. When you create a Data Processor transformation, you define components that convert the data.

A Data Processor transformation can contain multiple components to process data. Each component might contain other components.

For example, you might receive customer invoices in Microsoft Word files. You configure a Data Processor transformation to parse the data from each word file. Extract the customer data to a Customer table. Extract order information to an Orders table.

When you create a Data Processor transformation, you define an XMap, Script, or Library. An XMap converts an input hierarchical file into an output hierarchical file of another structure. A Library converts an industry messaging type into an XML document with a hierarchy structure or from XML to an industry standard format. A Script can parse source documents to hierarchical format, convert hierarchical format to other file formats, or map a hierarchical document to another hierarchical format.

Define Scripts in the Data Processor transformation IntelliScript editor. You can define the following types of Scripts:

- **Parser.** Converts source documents to XML. The output of a Parser is always XML. The input can have any format, such as text, HTML, Word, PDF, or HL7.
- **Serializer.** Converts an XML file to an output document of any format. The output of a Serializer can be any format, such as a text document, an HTML document, or a PDF.
- **Mapper.** Converts an XML source document to another XML structure or schema. You can convert the same XML documents as in an XMap.
- **Transformer.** Modifies the data in any format. Adds, removes, converts, or changes text. Use Transformers with a Parser, Mapper, or Serializer. You can also run a Transformer as stand-alone component.
- **Streamer.** Splits large input documents, such as multi-gigabyte data streams, into segments. The Streamer processes documents that have multiple messages or records in them, such as HIPAA or EDI files.

Data Processor Transformation Views

The Data Processor transformation has multiple views that you access when you configure the transformation and run it in the Developer tool.

Some of the Data Processor transformation views do not appear in the Developer tool by default. To change the views for the transformation, click **Window > Show View > Other > Informatica**. Select the views you want to see.

The Data Processor transformation has the following fixed views:

Overview view

Configure ports and define the startup component.

References view

Add or remove schemas from the transformation.

Settings view

Configure transformation settings for encoding, output control, and XML generation.

Objects view

Add, modify, or delete Script, XMap and Library objects from the transformation.

You can also access the following views for the Data Processor transformation:

Data Processor Hex Source view

Shows an input document in hexadecimal format.

Data Processor Events view

Shows information about events that occur when you run the transformation in the Developer tool.
Shows initialization, execution, and summary events.

Script Help view

Shows context-sensitive help for the Script editor.

Data Viewer view

View example input data, run the transformation, and view output results.

Data Processor Transformation Ports

Define the Data Processor transformation ports on the transformation **Overview** view.

A Data Processor transformation can read input from a file, a buffer, or a streamed buffer from a complex file reader. You can use a flat file reader as a buffer to read an entire file at one time. You can also read an input file from a database.

The output ports that you create depend on whether you want to return a string, complex files, or rows of relational data from the transformation.

Data Processor Transformation Input Ports

When you create a Data Processor transformation, the Developer tool creates a default input port. When you define an additional input port in a Script startup component, the Developer tool creates an additional input port in the transformation.

The input type determines the type of data that the Data Integration Service passes to the Data Processor transformation. The input type determines whether the input is data or a source file path.

Configure one of the following input types:

Buffer

The Data Processor transformation receives rows of source data in the Input port. Use the buffer input type when you configure the transformation to receive data from a flat file or from an Informatica transformaton.

File

The Data Processor transformation receives the source file path in the Input port. The Data Processor startup component opens the source file. Use the file input type to parse binary files such as Microsoft Excel or Microsoft Word files. You can also use the File input type for large files that might require a lot of system memory to process with a buffer input port.

Service Parameter

The Data Processor transformation receives values to apply to variables in the service parameter ports. When you choose the variables to receive input data, the Developer tool creates a service parameter port for each variable.

Output_Filename

When you configure the default output port to return a file name instead of row data, the Developer tool creates an Output_Filename port. You can pass a file name to the Output_Filename port from a mapping.

When you define an input port you can define the location of the example input file for the port. An example input file is a small sample of the input file. Reference an example input file when you create Scripts. You also use the example input file when you test the transformation in the **Data Viewer** view. Define the example input file in the **Input Location** field.

Service Parameter Ports

You can create input ports that receive values for variables. The variables can contain any datatype such as a string, a date, or a number. A variable can also contain a location for a source document. You can reference the variables in a Data Processor component.

When you create an input port for a variable, the Developer tool shows a list of variables that you can choose from.

Creating Service Parameter Ports

You can create input ports that receive values for variables. You can also remove the ports that you create from variables.

1. Open the Data Processor transformation **Overview** view.
2. Click **Choose**.
The Developer tool displays a list of variables and indicates which variables already have ports.
3. Select one or more variables.
The Developer tool creates a buffer input port for each variable that you select. You cannot modify the port.
4. To remove a port that you create from a variable, disable the selection from the variable list. When you disable the selection, the Developer tool removes the input port.

Data Processor Transformation Output Ports

The Data Processor transformation has one output port by default. If you define additional output ports in a Script, the Developer tool adds the ports to the Data Processor transformation. You can create groups of ports if you configure the transformation to return relational data. You can also create service parameter ports and pass-through ports.

Default Output Port

The Data Processor transformation has one output port by default. When you create relational output, you can define groups of related output ports instead of the default output port. When you define an additional output port in a Script component, the Developer tool adds an additional output port to the transformation.

Configure one of the following output types for a default output port:

Buffer

The Data Processor transformation returns XML through the Output port. Choose the Buffer file type when you parse documents or when you map XML to other XML documents in the Data Processor transformation.

File

The Data Integration Service returns an output file name in the Output port for each source instance or row. The Data Processor transformation component writes the output file instead of returning data through the Data Processor transformation output ports.

When you select a File output port, the Developer tool creates an Output_FileName input port. You can pass a file name into the Output filename port. The Data Processor transformation creates the output file with a name that it receives in this port.

If the output file name is blank, the Data Integration Service returns a row error. When an error occurs, the Data Integration Service writes a null value to the Output port and returns a row error.

Choose the File output type when you transform XML to a binary data file such as a PDF file or a Microsoft Excel file.

Pass-Through Ports

You can configure pass-through ports to any Data Processor transformation. Pass-through ports are input and output ports that receive input data and return the same data to a mapping without changing it.

You can configure pass-through ports in a Data Processor transformation instance that is in a mapping.

To add a pass-through port, drag a port from another transformation in the mapping. You can also add ports in the **Ports** tab of the **Properties** view. Click **New** to add a pass-through port.

Note: When you add pass-through ports to a Data Processor transformation with relational input and hierarchical output, add the ports to the root group of the relational structure.

Data Processor transformations can include pass-through ports with custom data types.

Startup Component

A startup component defines the component that starts the processing in the Data Processor transformation. Configure the startup component on the **Overview** view.

A Data Processor transformation can contain multiple components to process data. Each component might contain other components. You must identify which component is the entry point for the transformation.

When you configure the startup component in a Data Processor transformation, you can choose an XMap, Library, or a Script component as the startup component. In terms of Scripts, you can select one of the following types of components:

- **Parser.** Converts source documents to XML. The input can have any format, such as text, HTML, Word, PDF, or HL7.
- **Mapper.** Converts an XML source document to another XML structure or schema.
- **Serializer.** Converts an XML file to an output document of any format.
- **Streamer.** Splits large input documents, such as multi-gigabyte data streams, into segments.
- **Transformer.** Modifies the data in any format. Adds, removes, converts, or changes text. Use Transformers with a Parser, Mapper, or Serializer. You can also run a Transformer as stand-alone component.

Note: If the startup component is not an XMap or Library, you can also configure the startup component in a Script instead of in the **Overview** view.

References

You can define transformation references, such as schema or maplet references, by selecting a schema or maplet to serve as a reference. Some Data Processor transformations require a hierarchical schema to define the input or output hierarchy for relevant components in the transformation. To use the schema in the transformation, you define a schema reference for the transformation. You can also use a specialized action

named the RunMapplet action to call a mapplet from a Data Processor transformation. To call a mapplet, you must first define a mapplet reference for the transformation.

You can define transformation references, such as schema or mapplet references, in the transformation **References** view.

Schema References

The Data Processor transformation references schema objects in the Model repository. The schema objects can exist in the repository before you create the transformation. You can also import schemas from the transformation **References** view.

The schema encoding must match the input encoding for Serializer or Mapper objects. The schema encoding must match the output encoding for Parser or Mapper objects. Configure the working encoding in the transformation **Settings** view.

A schema can reference additional schemas. The Developer tool shows the namespace and prefix for each schema that the Data Processor transformation references. When you reference multiple schemas with empty namespaces the transformation is not valid.

Mapplet References

You can call a mapplet from a Data Processor transformation with the RunMapplet action. Before you add the RunMapplet action to a Data Processor transformation component, you must first define a reference to the mapplet that you want to call.

Data Processor Transformation Settings

Configure code pages, XML processing options, and logging settings in the Data Processor transformation **Settings** view.

Character Encoding

A character encoding is a mapping of the characters from a language or group of languages to hexadecimal code.

When you design a Script, you define the encoding of the input documents and the encoding of the output documents. Define the working encoding to define how the IntelliScript editor displays characters and how the Data Processor transformation processes the characters.

Working Encoding

The working encoding is the code page for the data in memory and the code page for the data that appears in the user interface and work files. You must select a working encoding that is compatible with the encoding of the schemas that you reference in the Data Processor transformation.

Setting	Description
XML Special Characters Encoding	<p>Determines the representation of XML special characters. You can select None or XML.</p> <ul style="list-style-type: none"> - None. Leave as & &lt; &gt; &quot; &apos; Entity references for XML special characters are interpreted as text, for example, the character > appears as &gt; Default in None. - XML. Convert to & < > " ' Entity references for XML special characters are interpreted as regular characters. For example, &gt; appears as the following character: >
Byte Order	<p>Describes how multi-byte characters appear in the input document. You can select the following options:</p> <ul style="list-style-type: none"> - Little-endian. The least significant byte appears first. Default. - Big-endian. The most significant byte appears first. - No binary conversion.

Output Encoding

The output encoding determines how character data is encoded in the main output document.

The following table describes the encoding settings in the **Output** area:

Setting	Description
Use Working Encoding	The output encoding is the same as the working encoding.
Other	The user selects the output encoding from the list.
XML Special Characters Encoding	<p>Determines the representation of XML special characters. You can select None or XML.</p> <ul style="list-style-type: none"> - None. Leave as & &lt; &gt; &quot; &apos; Entity references for XML special characters are interpreted as text, for example, the character > appears as &gt; Default. - XML. Convert to & < > " ' Entity references for XML special characters are interpreted as regular characters. For example, &gt; appears as the following character: >
Same as Input Encoding	The output encoding is the same as the input encoding.
Byte order	<p>Describes how multi-byte characters appear in the input document. You can select the following options:</p> <ul style="list-style-type: none"> - Little-endian. The least significant byte appears first. Default. - Big-endian. The most significant byte appears first. - No binary conversion.

Rules and Guidelines for Character Encoding

Use the following rules and guidelines when you configure encodings:

- To increase performance, set the working encoding to be the same encoding as the output document.
- Set the input encoding to the same encoding as the input document.
- Set the output encoding to the same encoding as the output document.
- For languages that have multiple-byte characters, set the working encoding to UTF-8. For the input and output encoding, you can use a Unicode encoding such as UTF-8 or a double-byte code page such as Big5 or Shift_JIS.

Output Settings

Configure output control settings to control whether the Data Processor transformation creates event logs and saves output documents.

You can control the types of messages that the Data Processor transformation writes to the design-time event log. If you save the parsed input documents with the event logs, you can view the context where the error occurred in the **Event** view.

The following table describes the settings in the **Design-Time Events** area:

Setting	Description
Log design-time events	Determines whether to create a design-time event log. By default, the Data Processor transformation logs notifications, warnings, and failures in the design-time event log. You can exclude the following event types: <ul style="list-style-type: none">- Notifications- Warnings- Failures
Save parsed documents	Determines when the Data Processor transformation saves a parsed input document. You can select the following options: <ul style="list-style-type: none">- Always.- Never- On failure The default is always.

The following table describes the settings in the **Run-Time Events** area:

Setting	Description
Log run-time events	Determines whether an event log is created when you run the transformation from a mapping. <ul style="list-style-type: none">- Never.- On failure The default is Never.

The following table describes the settings in the **Output** area:

Setting	Description
Disable automatic output	Determines whether the Data Processor transformation writes the output to the standard output file. Disable standard output in the following situations: <ul style="list-style-type: none"> - You pass the output of a Parser to the input of another component before the transformation creates an output file. - You use a WriteValue action to write data directly to the output from a Script instead of passing data through the output ports.
Disable value compression	Determines whether the Data Processor transformation uses value compression to optimize memory use. Important: Do not disable value compression except when Informatica Global Customer Support advises you to disable it.

The following table describes the settings in the **Binary output port collection mode** area. You can select one of these options for binary output for a relational to hierarchical transformation with XML, Avro, or Parquet output, or for a Data Processor transformation Parser with Avro or Parquet output.

Setting	Description
Collect input rows to a single output	Determines whether the Data Processor transformation accumulates the relational input to a single binary output port.
Split output when size exceeds	Determines whether the Data Processor transformation divides the output into chunks based on a maximum stated output size.
Output row for each row (do not collect)	Determines whether the Data Processor transformation passes the output in separate rows.

Processing Settings

The processing settings define how the Data Processor transformation processes an element without a defined datatype. The settings affect Scripts. The settings do not affect elements that an XMap processes.

The following table describes the processing settings that affect XML processing in Scripts:

Setting	Description
Treat as xs:string	The Data Processor transformation treats an element with no type as a string. In the Choose XPath dialog box, the element or attribute appears as a single node.
Treat as xs:anyType	The Data Processor transformation treats an element with no type as anyType. In the Choose XPath dialog box, the element or attribute appears as a tree of nodes. One node is of xs:string type, and all named complex data types appear as tree nodes.

The following table describes a processing setting that affects Streamer processing:

Setting	Description
Streamer chunk size	This setting defines the amount of data that the Streamer reads each time from an input file stream. The Data Processor transformation applies this setting to a Streamer with a file input.

The following table describes a processing setting that affects hierarchical to relational transformation processing:

Setting	Description
Enforce strict validation	This setting determines if the Data Processor transformation performs strict validation for hierarchical input. When strict validation applies, the hierarchical input file must conform strictly to its schema. This option can be applied when the Data Processor mode is set to Output Mapping , which creates output ports for relational output. This option does not apply to mappings with JSON input from versions previous to version 10.2.1.
Normalize XML input	This setting determines if the Data Processor transformation normalizes XML input. By default, the transformation performs normalization for XML input. In some cases, you might choose to skip automatic normalization to increase performance.

XMap Settings

The XMap setting defines how the Data Processor transformation processes XMap input elements that are not transformed to output elements. The unread elements are passed to a dedicated port named **XMap_Unread_Input_Values**. The setting takes affect only when the XMap is selected as the start-up component. The setting does not affect elements that the XMap processes.

To pass unread XMap elements to a dedicated port, enable the setting **Write unread elements to an additional output port**.

XML Output Configuration

The XML generation settings define characteristics of XML output documents.

The following table describes the XML generation settings in the **Schema Title** area:

Setting	Description
Schema location	Defines the schemaLocation for the root element of the main output document.
No namespace schema location	Defines the xsi:noNamespaceSchemaLocation attribute of the root element of the main output document.

Configure XML Output Mode settings to determine how the Data Processor transformation handles missing elements or attributes in the input XML document. The following table describes the XML generation settings in the **XML Output Mode** area:

Setting	Description
As is	Do not add or remove empty elements. Default is enabled.
Full	All required and optional elements defined in the output schema are written to the output. Elements that have no content are written as empty elements.
Compact	Removes empty elements from the output. If Add for Elements is enabled, then the Data Processor transformation removes only the optional elements. If Add for Elements is disabled, the Data Processor transformation removes all empty elements. The XML output might not be valid.

The following table describes the XML generation settings in the **Default Values for Required Nodes** area:

Setting	Description
Add for elements	When the output schema defines a default value for a required element, the output includes the element with a default value. Default is enabled.
Add for attributes	When the output schema defines a default value for a required attribute, the output includes the attribute with its default value. Default is enabled.
Validate added values	Determines whether the Data Processor transformation validates empty elements that are added by the Full mode output. Default is disabled. If Validate added values is enabled and the schema does not allow empty elements, the XML output might not be valid.

The following table describes the XML generation settings in the **Processing Instructions** area:

Setting	Description
Add XML processing instructions	Defines the character encoding and XML version of the output document. Default is selected.
XML version	Defines the XML version. The XML version setting has the following options: - 1.0 - 1.1 Default is 1.0.
Encoding	Defines the character encoding that is specified in the processing instruction. The Encoding setting has the following options: - Same as output encoding. The output encoding in the processing instruction is the same as the output encoding defined in the Data Processor transformation settings. - Custom. Defines the output encoding in the processing instruction. The user types the value in the field.
Add custom processing instructions	Adds other processing instructions to the output document. Enter the processing instruction exactly as it appears in the output document. Default is Disabled.

The following table describes the XML generation settings in the **XML Root** area:

Setting	Description
Add XML root element	Adds a root element to the output document. Use this option when the output document contains more than one occurrence of the root element defined in the output schema. Default is Disabled.
Root element name	Defines a name for the root element to add to the output document.

Events

An event is a record of a processing step from a component in the Data Processor transformation. In a Script or Library, each anchor, action, or transformer generates an event. In an XMap, each mapping statement generates an event.

You can view events in the **Data Processor Events** view.

Event Types

The Data Processor transformation writes events in log files. Each event has an event type that indicates if the event was successful, the event failed, or if the event ran with errors.

A component can generate one or more events. The component can pass or fail depending on whether the events succeed or fail. If one event fails, a component fails.

The following table describes the types of events that the Data Processor transformation generates:

Event Type	Description
Notification	Normal operation.
Warning	The Data Processor transformation ran, but an unexpected condition occurred. For example, the Data Processor transformation wrote data to the same element multiple times. Each time the element is overwritten, the Data Processor transformation generates a warning.
Failure	The Data Processor transformation ran, but a component failed. For example, a required input element was empty.
Optional Failure	The Data Processor transformation ran, but an optional component failed. For example, an optional anchor was missing from the source document.
Fatal Error	The Data Processor transformation failed because of a serious error. For example, the input document did not exist.

Data Processor Events View

The **Data Processor Events** view displays events when you run a Data Processor transformation from the Developer tool.

The **Data Processor Events** view has a **Navigation** panel and a **Details** panel. The Navigation panel contains a navigation tree. The navigation tree lists the components that the transformation ran in chronological order.

Each node has an icon that represents the most serious event below it in the tree. When you select a node in the **Navigation** panel, events appear in the **Details** panel.

The navigation tree contains the following top-level nodes:

- **Service Initialization.** Describes the files and the variables that the Data Processor transformation initializes.
- **Execution.** Lists the components that the Script, Library or XMap ran.
- **Summary.** Displays statistics about the processing.

When you run an XMap, each node name in the navigation panel has a number in square brackets, such as [5]. To identify the statement that generated the events for the node, right-click in the statements grid and select Go to Row Number. Enter the node number.

When you run a Script and double-click an event in the **Navigation** panel or the **Details** panel, the Script editor highlights the Script component that generated the event. The **Input** panel of the **Data Viewer** view highlights the part of the example source document that generated the event.

Logs

A log contains a record of the Data Processor transformation. The Data Processor transformation writes events to logs.

The Data Processor transformation creates the following types of logs:

Design-time event log

The design-time event log contains events that occur when you run the Data Processor transformation in the **Data Viewer** view. View the design-time log in the **Events** view.

Run-time event log

The run-time event log contains events that occur when you run the Data Processor transformation in a mapping. You can view the run-time event log in a text editor or you can drag a run-time event log into the **Events** view of the Data Processor transformation.

User log

The user log contains events that you configure for components in a Script. The Data Processor transformation writes to the user log when you run it from the **Data Viewer** view and when you run it in a mapping. You can view the user log in a text editor.

Design-Time Event Log

The design-time event log contains the events that occur when you run the Data Processor transformation from the **Data Viewer** in the Developer tool.

When you run a Data Processor transformation from the **Data Viewer** view, the design-time event log appears in the **Data Processor Events** view. By default, the design-time event log contains notifications, warnings, and failures. In the transformation settings, you can configure the Data Processor transformation to exclude one or more event types from the log.

When you save the input documents with the log, you can click an event in the **Data Processor Events** view to find the location in the input document that generated the event. When you configure the Data Processor transformation settings, you can choose to save the input files for every run or only on failure.

The design-time event log is named `events.cme`. You can find the design-time event log for the last run of the Data Processor transformation in the following directory:

```
C:\<Installation_directory>\clients\DT\CMReports\Init\events.cme
```

The Data Processor transformation overwrites the design-time event log every time you run the transformation in the **Data Viewer**. Rename the design-time event log if you want to view it after a later run of the transformation, or if you want to compare the logs of different runs. When you close the Developer tool, the Developer does not save any files in the

Run-Time Event Log

The run-time event log records the events that occur when you run the Data Processor transformation in a mapping.

If the Data Processor transformation completes the run with no failures, it does not write an event log. If there are failures in the run, Data Processor transformation runs a second time and writes an event log during the second run. The run-time event log is named `events.cme`.

On a Windows machine, the run-time event log is in the following directory:

```
C:\<Installation_Directory>\clients\DT\CMReports\Tmp\
```

On a Linux or UNIX machine, the run-time event log for a root user is in the following directory:

```
/root/<Installation_Directory>/clients/DT/CMReports/Tmp
```

On a Linux or UNIX machine, you can find the run-time event log for a non-root user in the following directory:

```
/home/[UserName]/<Installation_Directory>/DT/CMReports/Tmp
```

Use the configuration editor to change the location of the run-time event log.

Viewing an Event Log in the Data Processor Events View

Use the **Data Processor Events** view to view a design-time event log or a run-time event log.

Open Windows Explorer, and then browse to the event log file you want to view. Drag the log from the Windows Explorer window to the **Data Processor Events** view. Right-click in the **Data Processor Events** view, and then select **Find** to search the log.

Note: To reload the most recent design-time event log, right-click the **Data Processor Events** view, and then select **Reload Project Events**.

User Log

The user log contains custom messages that you configure about failures of components in a Script.

The Data Processor transformation writes messages to the user log when you run a Script from the **Data Viewer** view and when you run it in a mapping.

When a Script component has the **on_fail** property, you can configure it to write a message to the user log when it fails. In the Script, set the **on_fail** property to one of the following values:

- LogInfo
- LogWarning
- LogError

Each run of the Script produces a new user log. The user log file name contains the transformation name with a unique GUID:

```
<Transformation_Name>_<GUID>.log
```

For example, `CalculateValue_Aa93a9d14-a01f-442a-b9cb-c9ba5541b538.log`

On a Windows machine, you can find the user log in the following directory:

```
c:\Users\[UserName]\AppData\Roaming\Informatica\DataTransformation\UserLogs
```

On a Linux or UNIX machine, you can find the user log for the root user in the following directory:

```
/<Installation_Directory>/DataTransformation/UserLogs
```

On a Linux or UNIX machine, you can find the user log for a non-root user in the following directory:

```
home/<Installation_Directory>/DataTransformation/UserLogs
```

Data Processor Transformation Development

Use the New Transformation wizard to auto-generate a Data Processor transformation, or create a blank Data Processor transformation and configure it later. If you create a blank Data Processor transformation, you must select to create a Script, XMap, Library, or Validation Rules object in the transformation. A Script can parse source documents to hierarchical format, convert hierarchical format to other file formats, or map a hierarchical document to another hierarchical format. An XMap converts an input hierarchical file into an output hierarchical file of another structure. A Library converts an industry messaging type into an XML document with a hierarchy structure or from XML to an industry standard format. Choose the schemas that define the input or output hierarchies.

1. Create the transformation in the Developer tool.
2. For a blank Data Processor transformation, perform the following additional steps:
 - a. Add schema references that define the input or output XML hierarchies.
 - b. Create a Script, XMap, Library, or Validation Rules object.
3. Configure the input and output ports.
4. Test the transformation.

Create the Data Processor Transformation

Create a Data Processor transformation in the Developer tool. If you create a blank Data Processor transformation, you must then create a Script, XMap, Library, or Validation Rules object in the transformation. Alternatively, you can use the New Transformation wizard to auto-generate a Data Processor transformation.

1. In the Developer tool, click **File > New > Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model repository location to put the transformation.
4. Select whether to create Data Processor transformation with a wizard or to create a blank Data Processor transformation.
5. If you selected to create a blank Data Processor transformation, click **Finish**.

The Developer tool creates the empty transformation in the repository. The **Overview** view appears in the Developer tool.

6. If you selected to create a Data Processor transformation with a wizard, perform the following steps:
 - a. Click **Next**.
 - b. Select an input format.
 - c. Browse to select a schema, copybook, example file, or specification file if required for certain input formats such as COBOL or JSON.
 - d. Select an output format.
 - e. Browse to select a schema, copybook, example file, or specification file if required for the output format.
 - f. Click **Finish**. The wizard creates the transformation in the repository.

The transformation might contain a Parser, Serializer, Mapper, or an object with common components. If you selected a schema, copybook, example file, or specification file the wizard also creates a schema in the repository that is equivalent to the hierarchy in the file.

Select the Schema Objects

Choose the schema objects that define the input or output hierarchies for each XMap or Script component that you plan to create.

You can add schema references on the References view or you can add the schema references when you create Script or XMap objects. A schema object must exist in the Model repository before you can reference it in a Script or XMap.

1. In the Data Processor transformation **References** view, click **Add**.
2. If the schema object exists in the Model repository, browse for and select the schema.
3. If the schema does not exist in the Model repository, click **Create a new schema object** and import a schema object from a hierarchical schema file.
4. Click **Finish** to add the schema reference to the Data Processor transformation.

Create Objects in a Blank Data Processor Transformation

Create a Script, Library, XMap, or Validation Rules object on the Data Processor transformation **Objects** view. After you create the object, you can open the object from the **Objects** view in order to configure it.

Creating a Script

Create a Script object and define the type of Script component to create. Optionally, you can define a schema reference and example source file.

1. In the Data Processor transformation **Objects** view, click **New**.
2. Enter a name for the Script and click **Next**.
3. Choose to create a Parser or Serializer. Select **Other** to create a Mapper, Transformer, or Streamer component.
4. Enter a name for the component.
5. If the component is the first component to process data in the transformation, enable **Set as startup component**.
6. Click **Next** if you want to enter a schema reference for this Script. Click **Finish** if you do not want to enter the schema reference.

7. If you choose to create a schema reference, select **Add reference to a Schema Object** and browse for the Schema object in the Model repository. Click **Create a new schema object** to create the Schema object in the Model repository.
8. Click **Next** to enter an example source reference or to enter example text. Click **Finish** if you do not want to define an example source.
Use an example source to define sample data and to test the Script.
9. If you choose to select an example source, select **File** and browse for the sample file.
You can also enter sample text in the **Text** area. The Developer tool uses the text to test a Script.
10. Click **Finish**.
The **Script** view appears in the Developer tool editor.

Creating an XMap

Create an XMap on the Data Transformation **Objects** view. When you create an XMap, you must have a schema that describes the input and the output hierarchical documents. You select the element in the schema that is the root element for the input hierarchy.

1. In the Data Processor transformation **Objects** view, click **New**.
2. Select XMap and click **Next**.
3. Enter a name for the XMap.
4. If the XMap component is the first component to process data in the transformation, enable **Set as startup component**.
Click **Next**.
5. If you choose to create a schema reference, select **Add reference to a Schema Object**, and browse for the Schema object in the Model repository.
To import a new Schema object, click **Create a new schema object**.
6. If you have a sample hierarchical file that you can use to test the XMap with, browse for and select the file from the file system.
You can change the sample hierarchical file.
7. Choose the root for the input hierarchy.
In the **Root Element Selection** dialog box, select the element in the schema that is the root element for the input hierarchical file. You can search for an element in the schema. You can use pattern searching. Enter `*<string>` to match any number of characters in the string. Enter `?<character>` to match a single character.
8. Click **Finish**.
The Developer tool creates a view for each XMap that you create. Click the view to configure the mapping.

Creating a Library

Create a Library object on the Data Transformation **Objects** view. Select the message type, component and name. Optionally, you can define a sample message type source file that you can use to test the Library object.

Before you create a Library in the Data Processor transformation, install the library software package on your computer.

1. In the Data Processor transformation **Objects** view, click **New**.

2. Select Library and click **Next**.
3. Browse to select the message type.
4. Choose to create a Parser or Serializer.
Create a Parser if the Library object input is a message type and the output is XML. Create a Serializer if the Library object input is XML and the output is a message type.
5. If the Library is the first component to process data in the Data Processor transformation, enable **Set as startup component**.
Click **Next**.
6. If you have a sample message type source file that you can use to test the Library with, browse for and select the file from the file system.
You can change the sample file.
7. Click **Finish**.
The Developer tool creates a view for each message type that you create. Click the view to access the mapping.

Creating Validation Rules

Create a Validation Rules object in the Data Processor transformation **Objects** view.

1. In the Data Processor transformation **Objects** view, click **New**.
2. Select Validation Rules and click **Next**.
3. Enter a name for the Validation Rules.
4. If you have a sample XML file that you can use to test the Validation Rules with, browse for and select the file from the file system.
You can change the sample XML file.
5. Click **Finish**.
The Developer tool creates a Validation Rules object and opens it in the Validation Rules editor.

Adding an Example Source

Choose the example source to test the Script, XMap, Library, or Validation Rules that you plan to create.

You can add an example source when you create a Script, XMap, Library, or Validation Rules. Once selected, the example source is added to the Model repository. Due to Model repository limitations, the example source file size is limited to 5 MB.

You can change the example source.

Create the Ports

Configure the input and output ports in the **Overview** view.

When you configure additional input or output ports in a Script, the Developer tool adds additional input ports and additional output ports to the transformation by default. You do not add input ports on the **Overview** view.

1. If you want to return rows of output data instead of XML, enable **Relational Output**.
When you enable relational output, the Developer tool removes the default output port.
2. Select the input port datatype, port type, precision and scale.

3. If you are not defining relational output ports, define the output port datatype, port type, precision, and scale.
4. If a Script has additional input ports, you can define the location of the example input file for the ports. Click the **Open** button in the **Input Location** field to browse for the file.
5. If you enabled relational output, click **Output Mapping** to create the output ports.
6. On the Ports view, map nodes from the **Hierarchical Output** area to fields in the **Relational Ports** area.

Testing the Transformation

Test the Data Processor transformation in the **Data Viewer** view.

Before you test the transformation, verify that you defined the startup component. You can define the startup component in a Script or you can select the startup component on the **Overview** tab. You also need to have chosen an example input file to test with.

1. Open the **Data Viewer** view.
2. Click **Run**.
The Developer tool validates the transformation. If there is no error, the Developer tool shows the example file in the **Input** area. The output results appear in the Output panel.
3. Click **Show Events** to show the **Data Processor Events** view.
4. Double-click an event in the **Data Processor Events** view in order to debug the event in the Script editor.
5. Click **Synchronize with Editor** to change the input file when you are testing multiple components, each with a different example input file.

If you modify the example file contents in the file system, the changes appear in the **Input** area.

Data Processor Transformation Export and Import

You can export a Data Processor transformation as a service and run it from a Data Transformation repository. You can also import a Data Transformation service to the Developer tool. When you import a Data Transformation service, the Developer tool creates a Data Processor transformation from the service.

Note: When you import a Data Transformation service to the Model repository, the Developer tool imports the associated schemas to the repository. If you modify the schema in the repository, sometimes the changes do not appear in the transformation schema references. You can close and open the Model repository connection, or close and open the Developer tool to cause the schema changes to appear in the transformation.

Exporting the Data Processor Transformation as a Service

You can export the Data Processor transformation as a Data Transformation service. Export the service to the file system repository of the machine where you want to run the service. You can run the service with PowerCenter, user-defined applications, or the Data Transformation CM_console command.

1. In the **Object Explorer** view, right-click the Data Processor transformation you want to export, and select **Export**.
The **Export** dialog box appears.
2. Select **Informatica > Export Data Processor Transformation** and click **Next**.

- The **Select** page appears.
3. Click **Next**.
The **Select Service Name and Destination Folder** page appears.
 4. Choose a destination folder:
 - To export the service on the machine that hosts the Developer tool, click **Service Folder**.
 - To deploy the service on another machine, click **Folder**. Browse to the `\ServiceDB` directory on the machine where you want to deploy the service.
 5. Click **Finish**.

Importing Multiple Data Transformation Services

You can import a directory of Data Transformation services from the machine where you saved the directory. When you import Data Transformation services to the Developer Model repository, the Developer tool imports the transformations, schemas and example data with the `.cmw` files. If you need to import many services, import a directory of services instead of one service at a time.

1. Click **File > Import**.
The **Import** dialog box appears.
2. Select **Informatica Import Data Transformation Services (Folder)** and click **Next**.
The **Import Data Transformation Service** page appears.
3. Browse to the directory that you want to import.
4. Browse to a location in the Repository where you want to save the transformations, then click **Finish**.
The Developer tool imports the transformations, schemas and example data with the `.cmw` file.

Importing a Data Transformation Service

You can import a Data Transformation service `.cmw` file to the Model repository to create a Data Processor transformation. The Developer tool imports the transformation, schemas and example data with the `.cmw` file.

1. Click **File > Import**.
The **Import** dialog box appears.
2. Select **Informatica Import Data Transformation Service (Single)** and click **Next**.
The **Import Data Transformation Service** page appears.
3. Browse to the service `.cmw` file that you want to import.
The Developer tool names the transformation according to the service file name. You can change the name.
4. Browse to a location in the Repository where you want to save the transformation, then click **Finish**.
The Developer tool imports the transformation, schemas and example data with the `.cmw` file.
5. To edit the transformation, double-click the transformation in the **Object Explorer** view.

Exporting a Mapping with a Data Processor Transformation to PowerCenter

When you export a mapping with a Data Processor transformation to PowerCenter, you can export the objects to a local file, and then import the mapping into PowerCenter. Alternatively, you can export the mapping directly into the PowerCenter repository.

1. In the **Object Explorer** view, select the mapping to export. Right-click and select **Export**.
The **Export** dialog box appears.
2. Select **Informatica > PowerCenter**.
3. Click **Next**.
The **Export to PowerCenter** dialog box appears.
4. Select the project.
5. Select the PowerCenter release.
6. Choose the export location, a PowerCenter import XML file or a PowerCenter repository.
7. Specify the export options.
8. Click **Next**.
The Developer tool prompts you to select the objects to export.
9. Select the objects to export and click **Finish**.
The Developer tool exports the objects to the location you selected. If you exported the mapping to a location, the Developer tool also exports the Data Processor transformations in the mapping, as services, to a folder at the location that you specified.
10. If you exported the mapping to a PowerCenter repository, the services are exported to the following directory path: %temp%\DTServiceExport2PC\
The export function creates a separate folder for each service with the following name:
<date><serviceFullName>
If the transformation includes relational mapping, a folder is created for relational to hierarchical mapping, and a separate folder for hierarchical to relational mapping.
11. Copy the folder or folders with Data Processor transformation services from the local location where you exported the files to the PowerCenter ServiceDB folder.
12. If you exported the mapping to a PowerCenter import XML file, import the mapping to PowerCenter. For more information about how to import an object to PowerCenter, see the *PowerCenter 9.6.0 Repository Guide*.

Data Processor Transformation Validation

After you export a Data Processor transformation as a service, you can run VRL validations on the service from the Data Transformation repository.

You can use a speed-enhanced Data Transformation engine for VRL validations. The speed-enhanced Data Transformation engine supports the following VRL functions:

- dt:exist
- dt:empty

- dt:date-valid
- dt:next-sequence
- dt:all-equal
- dt:lookup
- dt:regex-match

The speed-enhanced Data Transformation engine produces the output **ValidateValue**. **ValidateValue** contains the property `max_error_count`, with a default of 200 errors. When the number of errors exceeds the `max_error_count`, the validation stops.

Note: The speed-enhanced Data Transformation engine VRL syntax does not support the `<list>` tag.

Using a Speed-enhanced Data Transformation Engine for VRL Validations

After you export a Data Transformation service, you can use a speed-enhanced Data Transformation engine for VRL validations with the service.

- ▶ Set the following flag in the service `.cmw` file `optimize_vrl`.

Add the `optimize_vrl` flag to the `ServiceConfigProf` instance, as shown in the following example:

```
instance ServiceConfig = ServiceConfigProf<add_required_xml_elements,
add_required_xml_attributes, optimize_vrl>
```

Data Processor Transformation in a Non-native Environment

The Data Processor transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.*
- Databricks Spark engine. Not supported.

* For information about the Data Processor transformation support on the Spark engine, see the [KB article](#).

CHAPTER 14

Decision Transformation

This chapter includes the following topics:

- [Decision Transformation Overview, 246](#)
- [Decision Transformation Functions, 247](#)
- **[Decision Transformation Conditional Statements , 249](#)**
- [Decision Transformation Operators, 250](#)
- [Decision Transformation NULL Handling, 251](#)
- [Configuring a Decision Strategy , 251](#)
- [Decision Transformation Advanced Properties, 251](#)
- [Decision Transformation in a Non-native Environment, 252](#)

Decision Transformation Overview

The Decision transformation is a passive transformation that evaluates conditions in input data and creates output based on the results of those conditions.

Configure a Decision transformation to generate different values based on the values found in input fields. For example, if customer revenue is greater than a specific amount, you can add the string "Priority" to the customer name.

You can add multiple decision strategies to the Decision transformation. Each strategy evaluates an `IF-THEN-ELSE` conditional statement. Within this statement, you can use `ELSEIF` conditions or nest additional `IF-THEN-ELSE` statements.

The Decision transformation is similar to the Expression transformation in that it allows you to use conditional statements and functions to test source data. However, the Decision transformation is different from the Expression transformation in the following ways:

- The Decision transformation uses `IF-THEN-ELSE` statements to evaluate conditions. The Expression transformation uses `IIF` statements.
- The Decision transformation contains functions that are not available in the Expression transformation.
- Each decision strategy can generate multiple outputs.

Decision Transformation Functions

The Decision transformation provides access to predefined functions that you use to define decision strategies.

The Decision transformation expression editor contains a Decision folder. This folder contains functions that are specific to the Decision transformation. The editor also contains other folders that provide access to Expression transformation functions.

When you click a function in the expression editor, the transformation displays the usage and datatypes for the function, in addition to a description of what the function does.

Note: Not all Expression transformation functions are compatible with the Decision transformation. The Decision transformation only provides access to compatible Expression transformation functions.

List of Decision Transformation Functions

- ABS
- ADD_TO_DATE
- ASCII
- CEIL
- CHOOSE
- CHR
- CHRCODE
- CONCAT
- CONTAINS
- CONVERT_BASE
- COS
- COSH
- CRC32
- CUME
- CURDATE
- CURTIME
- DATE_COMPARE
- DATE_DIFF
- DATECONVERT
- EXP
- FLOOR
- FV
- GET_DATE_PART
- GREATEST
- IN
- INDEXOF
- INITCAP
- INSTR

- IS_DATE
- IS_NUMBER
- ISNULL
- LAST_DAY
- LEAST
- LEFTSTR
- LENGTH
- LN
- LOG
- LOWER
- LPAD
- LTRIM
- MAKE_DATE_TIME
- MAX
- MD5
- METAPHONE
- MIN
- MOD
- MONTHCOMPARE
- MOVINGAVG
- MOVINGSUM
- NPER
- PMT
- POWER
- PV
- RAND
- RATE
- REG_EXTRACT
- REG_MATCH
- REG_REPLACE
- REPLACECHR
- REPLACESTR
- REVERSE
- RIGHTSTR
- ROUND
- RPAD
- RTRIM
- SET_DATE_PART
- SIGN

- SIN
- SINH
- SOUNDEX
- SQRT
- SUBSTR
- TAN
- TANH
- TIMECOMPARE
- TO_CHAR
- TO_DATE
- TO_FLOAT
- TO_INTEGER
- TRUNC
- UPPER
- XOR

Note: Use a constant value to define the date format in the CURDATE, DATE_COMPARE, DATECONVERT, and MONTHCOMPARE functions in the Decision transformation.

Decision Transformation Conditional Statements

The Decision transformation uses IF-THEN-ELSE conditional statements to evaluate input data.

Within these conditional statements, you can use ELSEIF conditions or nest additional IF-THEN-ELSE statements. Decision transformation conditional statements use the following format:

```
// Primary condition
IF <Boolean expression>
THEN <Rule Block>
// Optional - Multiple ELSEIF conditions
ELSEIF <Boolean expression>
THEN <Rule Block>
// Optional ELSE condition
ELSE <Rule Block>
ENDIF
```

You can nest additional conditional statements within a rule block.

Decision Transformation Operators

Use Decision transformation operators to define decision strategies.

The following table describes the decision transformation operators:

Operator Type	Operator	Description
Assignment	:=	Assigns a value to a port.
Boolean	AND	Adds a required logical condition. For the parent Boolean expression to be true, all logical conditions linked by this operator must be true.
Boolean	OR	Adds a logical condition. For the parent Boolean expression to be true, at least one logical condition linked by this operator must be true.
Boolean	NOT	Specifies a negative logical condition. For the parent Boolean expression to be true, the negative condition specified by this operator must be true.
Decision	=	Tests whether compared items are equal. Use with string or numeric datatypes.
Decision	<>	Tests whether compared items are not equal. Use with string or numeric datatypes.
Decision	<	Tests whether a value is less than another value. Use with numeric datatypes.
Decision	<=	Tests whether a value is less than or equal to another value. Use with numeric datatypes.
Decision	>	Tests whether a value is greater than another value. Use with numeric datatypes.
Decision	>=	Tests whether a value is greater than or equal to another value. Use with numeric datatypes.
Numerical	-	Subtraction
Numerical	NEG	Negation
Numerical	+	Addition
Numerical	*	Multiplication
Numerical	/	Division
Numerical	%	Modulo. Returns the remainder after dividing one number by another.
String		Concatenates strings.

Decision Transformation NULL Handling

NULL handling determines how the Data Integration Service processes NULL value data in a Decision transformation.

When you enable NULL handling, the Decision transformation retains the original form of NULL input data. The transformation evaluates functions using the NULL input value.

When you disable NULL handling, the Decision transformation assigns a default value to NULL input data. The transformation evaluates functions using the default value. For example, if an integer type input field has a NULL value, the Decision transformation assigns a value of 0 to the input and evaluates functions using an input value of 0.

By default, NULL handling is not enabled in the Decision transformation. You enable NULL handling on the **Strategies** tab. You can enable NULL handling after you configure a strategy for the transformation.

Configuring a Decision Strategy

To configure a decision strategy, connect source data to the Decision transformation and edit the properties in the transformation views.

1. Open a Decision transformation.
2. Verify that the transformation contains input and output ports.
3. Select the **Decision** view.
4. Click **Add**.
5. Enter a name for the strategy.
6. In the **Expression** area, enter an `IF-THEN-ELSE` conditional statement.
7. To add a function to the expression, browse the functions in the **Functions** tab and double-click a function name.
Tip: To quickly enter a function, type the first letters of the function name and select `CTRL-Space`.
8. To add a port to the expression, browse the ports in the **Ports** tab. Double-click a port name to add it to the expression. Optionally, click **Edit Output Ports** to edit output port settings or add output ports.
9. Optionally, add comment lines by typing `"/"` followed by your comments.
10. Click **Validate** to determine if the decision expression is valid.
11. Click **OK** to save the strategy.
12. Optionally, add additional strategies. Each strategy must use unique output ports. Strategies cannot share output ports.

Decision Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Decision transformation.

Configure the following advanced properties for a Decision transformation:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Partitionable

The transformation can be processed with multiple threads. Clear this option if you want the Data Integration Service to use one thread to process the transformation. The Data Integration Service can use multiple threads to process the remaining mapping pipeline stages.

You might want to disable partitioning for a Decision transformation when the transformation uses one of the numeric functions CUME, MOVINGSUM, or MOVINGAVG. These functions calculate running totals and averages on a row-by-row basis. A partitioned transformation that uses CUME, MOVINGSUM, or MOVINGAVG functions might not return the same calculated result with each mapping run.

If the transformation does not use CUME, MOVINGSUM, or MOVINGAVG functions, enable partitioning for the transformation to optimize performance.

Decision Transformation in a Non-native Environment

The Decision transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Decision Transformation on the Spark Engine

The Decision transformation is supported with the following restrictions:

- The transformation properties must specify that the transformation is partitionable.

Decision Transformation on the Databricks Spark Engine

The Decision transformation is supported with the following restrictions:

- The transformation properties must specify that the transformation is partitionable.

CHAPTER 15

Duplicate Record Exception Transformation

This chapter includes the following topics:

- [Duplicate Record Exception Transformation Overview, 253](#)
- [Duplicate Record Exception Process Flow, 254](#)
- [Duplicate Record Exceptions, 254](#)
- [Duplicate Record Exception Configuration View , 255](#)
- [Ports, 256](#)
- [Duplicate Record Exception Transformation Advanced Properties, 259](#)
- [Duplicate Record Exception Mapping Example, 259](#)
- [Creating a Duplicate Record Exception Transformation, 264](#)

Duplicate Record Exception Transformation Overview

The Duplicate Record Exception transformation is an active transformation that reads the output of a data quality process and identifies duplicate records that require manual review. The Duplicate Record Exception transformation is a multiple-group transformation.

The Duplicate Record Exception transformation receives input from another transformation or from a data object in another mapping. The input to the Exception transformation must contain a numeric match score that the transformation can use to determine if the record is a duplicate. Set an upper and lower match score threshold in the Duplicate Record Exception transformation.

The Duplicate Record Exception transformation performs one of the following actions:

- If the match score is greater than or equal to the upper threshold, the transformation treats the record as a duplicate and writes it to a database target.
- If the match score is less than the upper threshold and greater than the lower threshold the transformation treats the record as a possible duplicate and writes it to a the record to a different target for manual review. If the record belongs to a cluster, the transformation writes all records in the cluster to the target.
- When a cluster has any match score less than the lower threshold, all records in the cluster go to the unique records output group. Clusters of size 1 are routed to the unique group, regardless of match score.

By default, the Exception transformation does not write unique records to a target. You can configure the transformation to return the unique records.

- If any match score in a cluster is not in the range 0 - 100, the Exception transformation ignores all rows in the cluster. The Data Integration Service logs a message that includes the clusterID.

Duplicate Record Exception Process Flow

The Duplicate Record Exception transformation analyzes the output of other data quality transformations and creates tables that contain records with different levels of data quality.

You can configure data quality transformations in a single mapping, or you can create mappings for different stages in the process.

You can use the Analyst tool to review and update the duplicate records that require manual review.

Use the Developer tool to perform the following tasks:

1. Create a mapping that generates score values for data quality issues.
2. Use a Match transformation in cluster mode to generate score values for duplicate record exceptions.
3. Configure the Duplicate Record Exception transformation to read the Match transformation output. Configure the transformation to write records to database tables based on match score values in the records.
4. Configure target data objects for automatic consolidation records.
5. Click the **Generate duplicate record table** option to create the duplicate record table and add it to the mapping canvas.
6. Add the mapping to a workflow.
7. Configure a Human task to assign manual review of possible duplicate records to users. Users can review and update the records in Analyst tool.

Duplicate Record Exceptions

You can use a Duplicate Record Exception transformation to identify clusters of duplicate data that needs manual review. The match scores of records in clusters determines the potential duplicates. You can configure upper and lower thresholds for match scores in the transformation. The upper and lower thresholds define the degree of similarity.

A cluster contains related records that a matching operation groups together. The Match transformation creates clusters using the duplicate analysis operation and the identity resolution operation. Each record in a cluster has the same cluster ID. When the lowest match score in a cluster is between the upper and lower thresholds, the Duplicate Record Exception transformation identifies the cluster as a duplicate record exception cluster. The Match transformation adds a cluster ID value column to all the records. Duplicate records receive the same cluster ID.

The lowest record score in a cluster determines the cluster type. A cluster might have 11 records that have a match score of 0.95 and one record with match score of 0.79. If the upper threshold is 0.9 and the lower threshold is 0.8, the Exception transformation writes the records to the unique records table.

Duplicate Record Exception Configuration View

Define the match score thresholds and configure where the Duplicate Record Exception transformation writes the different types output data.

The following figure shows the properties that you can configure:

Type	Standard Output	Duplicate Record Table
Automatic Consolidation (Above upper threshold)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Manual Consolidation (Between thresholds)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Unique Records (Below lower threshold)	<input type="checkbox"/>	<input type="checkbox"/>

Create separate output group for unique records

Generate duplicate record table

You can configure the following properties:

Lower Threshold

The lower limit for the duplicate record score range. The transformation processes records with match scores less than this value as unique records. The lower threshold value is a number from 0 to 1.

Upper Threshold

The upper limit for the duplicate record score range. The transformation processes records with match scores greater than or equal to the upper threshold as duplicate records. The upper threshold value is a number greater than the lower threshold number.

Automatic Consolidation

Clusters in which all records have match scores greater than the upper threshold. Automatic consolidation clusters do not require review. The records are duplicate. You can use the Consolidation transformation to combine the records. By default, the Duplicate Record Exception transformation writes automatic consolidation clusters to standard output ports.

Manual Consolidation

Clusters in which all records have match scores greater than or equal to the lower threshold and at least one record has a match score less than the upper threshold. You must perform a manual review of the clusters to determine if they contain duplicate records. By default, the Duplicate Record Exception transformation writes manual consolidation records to the duplicate record table.

Unique Consolidation

Clusters with a cluster size equal to one or clusters in which any record has a match score less than the lower threshold. Unique record clusters are not duplicates. By default, the Duplicate Record Exception transformation does not write unique records to an output table.

Standard Output

The types of records that the transformation writes to the standard output ports.

Default is automatic consolidation records.

Duplicate Record Table

The types of record that the transformation writes to the duplicate record output ports. Default is manual consolidation records.

Create separate output group for unique records

Creates a separate output group for unique records. If you do not create a separate table for the unique records, you can configure the transformation to write the unique records to one of the other groups. Or, you can skip writing unique records to an output table. Default is disabled.

Generate duplicate record table

Creates a database object to contain the duplicate record cluster data. When you select this option, the Developer tool creates the database object. The Developer tool adds the object to the Model repository, adds an instance of the object to the mapping canvas, and links the ports to the object.

Generating a Duplicate Records Table

You can generate a Duplicate Records table from a Duplicate Record Exception transformation instance in a mapping.

1. In the **Configuration** view, click **Generate Duplicate Records table** to generate the table.
The **Create Relational Data Object** dialog box appears.
2. Browse for and select a connection to the database to contain the table.
3. Enter a name for the Duplicate Records table in the database.
4. Enter a name for the Duplicate Records table object in the Model repository.
5. Click **Finish**.
The Developer tool adds the new table to the mapping canvas.

Ports

The Duplicate Record Exception transformation has multiple groups of input and output ports.

The following figure shows an example of the input and output ports:

Ports							
Name	Type	Precision	Scale	Input	Output	Default	Description
Inputs							
Data (3)							
1	Employee	decimal	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	Name	string	50	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3	Addr1	string	50	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Control (3)							
1	Score	double	15	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	Row_Identifier	string	25	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3	Cluster_ID	integer	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Outputs							
Standard Output (6)							
1	Score	double	15	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
2	Row_Identifier	string	25	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
3	Cluster_ID	integer	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
4	Employee	decimal	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
5	Name	string	50	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
6	Addr1	string	50	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Cluster Data (9)							
1	Row_Identifier	bigint	19	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
2	Sequential_Cl...	bigint	19	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
3	Cluster_ID	integer	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Duplicate Record Exception Transformation Input Ports

A Duplicate Record Exception transformation has a Data group and a Control group of input ports.

The **Data** group contains user-defined ports that receive the source data.

The **Control** ports receive metadata that the Match transformation adds to the source data. The following table describes the **Control** ports:

Port	Description
Score	Decimal value between 0 and 1. Identifies the degree of similarity with the record that linked the record to the cluster.
Row_Identifier	A unique identifier for the record.
Cluster_ID	The ID of the match cluster to which the record belongs.

Duplicate Record Exception Transformation Output Ports

The Duplicate Record Exception transformation has multiple output groups. By default, the transformation writes duplicate records to the **Standard Output** group. The transformation writes potential matches to the **Cluster Data** group. You can add an output group for unique records.

You can change the record types that the transformation writes to output ports by changing the default settings on the **Configuration** view.

The following table describes the output ports for the **Standard Output** group:

Port	Description
Score	Decimal value between 0 and 1. Identifies the degree of similarity between the record and another record in a cluster.
Row_Identifier	A unique identifier for the record.
Cluster_ID	ID of the cluster that the Match transformation assigned the record to.
User-defined ports	The source data fields.

The following table describes the output ports in the **Cluster Data** group:

Port	Description
Row_Identifier	The unique identifier for the record.
Sequential_Cluster_ID	Identifies the cluster in a Human task. A workflow uses the sequential cluster ID to assign a cluster to an instance of a Human task.
Cluster_ID	Identifies the cluster that the record belongs to. The Match transformation assigns a cluster ID to all records.
Score	Decimal value from 0 to 1. Identifies the degree of similarity with the record that linked the record to the cluster.
Is_Master	String value that indicates if the record is the preferred record in the cluster. By default, the first row in the cluster is the preferred record. Value is Y or N.
Workflow_ID	ID that identifies the workflow for the record in a task. When you run the mapping outside of a workflow, the workflow ID is DummyWorkflowID.
User-defined ports	The source data ports.

Creating Ports

Add each input port to the Data group. When you add the input port, the Developer tool adds an output port with the same name to the Standard Output group, the Cluster Data group, and the Unique Records group.

1. Select the Data input group.
The group is highlighted.
2. Click **New (Insert)**.
The Developer tool adds a field to the Data group, the Standard Output group, the Cluster Data group, and the Unique Records group.
3. Change the name of the field as required.
The Developer tool changes the field name in the other groups.
4. Enter the rest of the ports that you need to add for the data source.

Duplicate Record Exception Transformation Advanced Properties

The Duplicate Record Exception transformation contains advanced properties that determine the sort behavior, the cache memory behavior, and the tracing level.

You can configure the following advanced properties:

Sort

Determines whether the transformation sorts the input rows on the **Cluster ID** port data. The property is enabled by default.

Select the property if the input rows are not presorted.

Cache File Directory

Specifies the directory to which the Data Integration Service writes temporary data for the current transformation. The Data Integration Service writes temporary files to the directory when the volume of input data is greater than the available system memory. The Data Integration Service deletes the temporary files after the mapping runs.

You can enter a directory path on the property, or you can use a parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory. The default value is the CacheDir system parameter.

Cache File Size

Determines the amount of system memory that the Data Integration Service uses to sort the input data on the transformation. The default value is 400,000 bytes.

Before it sorts the data, the Data Integration Service allocates the amount of memory that you specify. If the sort operation generates a greater amount of data, the Data Integration Service writes the excess data to the cache file directory. If the sort operation requires more memory than the system memory and the file storage can provide, the mapping fails.

Note: If you enter a value of 65536 or higher, the transformation reads the value in bytes. If you enter a lower value, the transformation reads the value in megabytes.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Duplicate Record Exception Mapping Example

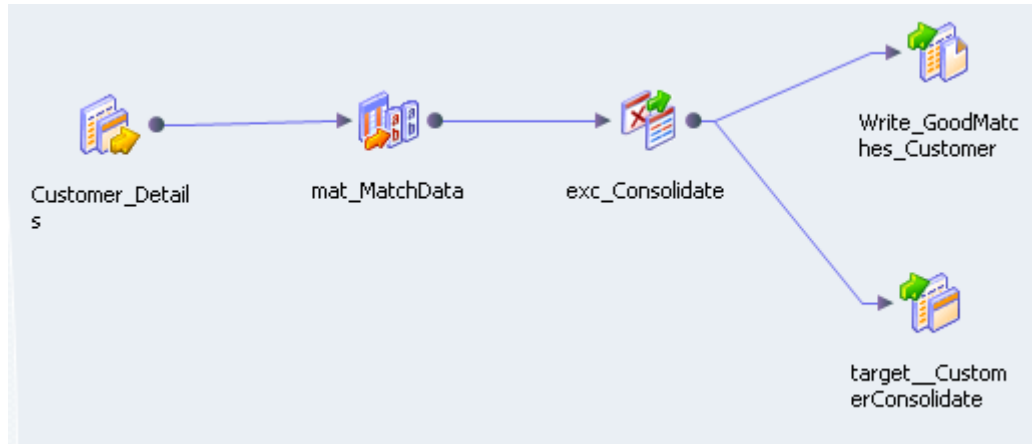
An organization runs a data project to review customer data. The organization determines that the customer data contains multiple duplicate records. The organization needs to manually review some of the records that might be duplicates.

You create a data quality mapping to identify the duplicate customer records. The mapping includes the Match transformation. The Duplicate Record Exception transformation receives the results from the Match transformation. The Duplicate Record Exception transformation writes each record cluster with an uncertain status to a database table. A data analyst reviews the data in the Analyst tool and determines which records are duplicate records.

Duplicate Record Exception Mapping

Configure a Duplicate Record Exception mapping that examines customer records and finds duplicate records.

The following figure shows the Duplicate Record Exception mapping:



The mapping contains the following objects:

Customer_Details

The data source that might contain duplicate records.

mat_MatchData

A Match transformation that examines the customer data to determine if records match. The Match transformation creates a numeric score that represents the degree of similarity between two column values. An algorithm calculates a match score as a decimal value in the range 0 through 1. An algorithm assigns a score of one when two column values are identical.

exc_Consolidate

A Duplicate Record Exception transformation that determines which records are possible duplicate customers, known duplicate customers, or unique customer records.

Write_GoodMatches_Customer table

Table that receives all the records that do not need manual review. The Duplicate Record Exception transformation writes duplicate records and unique records to this table.

Target_CustomerConsolidate table

The Exception transformation writes the possible duplicate records to the Target_CustomerConsolidate table. Records in this table require manual review in the Analyst tool.

Match Transformation

The Match transformation receives the customer data and performs an identity match.

Configure the Match transformation for the Clusters-Match All output type. The Match transformation returns matching records in clusters. Each record in a cluster must match at least one other record in the cluster with a score greater than or equal to the match threshold. The match threshold is .75.

Select the Division matching strategy on the Match transformation **Strategies** tab. The Division strategy is a predefined matching strategy that identifies an organization based on the address fields. On the Match

transformation **Strategies** tab, choose the input ports to examine in a match. Configure the strategy weight as .5.

The following figure shows the Division strategy configuration for the Match transformation:

Match Strategy	Custom Name	Weight	Match Fields	Properties
Division	Division1	0.5	ADDR1_1, ADDR1_2, COMPANY_1, COMPANY_2, ADDR2_1, ADDR2_2, ADDR4_1, ADDR4_2	Population: usa, Match Level: TYPICAL

The Match transformation adds cluster information to each output record. The transformation also adds a unique RowID to each record.

Duplicate Record Exception Input Groups

The Duplicate Record Exception transformation has two input groups. The transformation has a Data group that receives the customer data. The transformation has the Control group that contains the match score for the row, the row identifier, and the cluster ID.

The following figure shows the input groups in the Exception transformation:

Name	Type	Precision	Scale	Input	Output	Default	Description
Inputs							
Data (11)							
1	CUST_ID	decimal	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	COMPANY	string	200	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3	CONTACT	string	200	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
4	TITLE	string	200	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5	ADDR1	string	200	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
6	ADDR2	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
7	ADDR3	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
8	ADDR4	string	50	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
9	COUNTRY	string	50	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
10	PHONE	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
11	EMAIL	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Control (3)							
1	Score	double	15	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	Row_Identifier	string	25	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3	Cluster_ID	integer	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

The Data group contains the customer data. The customer data includes a customer ID, contact, title, and address fields. The Control group is the additional metadata that the Match transformation added for each customer record. The Control group contains the match score, the rowID, and the cluster ID.

Duplicate Record Exception Example Configuration View

Define the upper and lower thresholds on the **Configuration** view. Identify where the transformation writes the duplicate customer records, the possible duplicate records, and the unique customer records.

The following figure shows the Duplicate Record Exception transformation **Configuration** view:

Manual Review Thresholds

Lower Threshold :

Upper Threshold :

Data Routing Options

Type	Standard Output	Duplicate Record Table
Automatic Consolidation (...)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Manual Consolidation (Be...)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Unique Records (Below lo...)	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Create separate output group for unique records

The following table describes the configuration settings:

Option	Setting
Lower threshold	.80
Upper threshold	.95
Automatic Consolidation	Standard Output table
Manual Consolidation	Duplicate Record table
Unique Records	Standard Output table

Click **Generate duplicate record table** to create the duplicate record table. Do not create a separate table for the unique records. The transformation writes the unique records to the Standard Output table.

Standard Output Table Records

The Write_GoodMatches_Customer target table receive rows from the Standard Output group. The table receives unique records and it receives records that are duplicates. These records do not require manual review.

The following figure shows the Standard Output records that the Exception transformation returns:

Output										
Name: lexc_Consolidate_Good_Records										
Score	Row_Identifier	Cluster_ID	CUST_ID	COMPANY	CONTACT	TITLE	ADDR1	ADDR2	ADDR3	
1	1	1	1001590	SHOP'N SAVE	MR DANIEL COWLEY	SR COMPUTER ...	1755 WABASH...	SPRINGFIELD	<null>	
2	1	1	1001599	SHOP'N SAVE	MS VERENE TEKAUTZ	NETWORK SYS ...	45 GRAVOIS BL...	FENTON	MO	
3	1	1	1001604	SHOP'N SAVE	MR REUBEN HENDRICKS	GENERAL MAN...	800 CARLYLE A.	BELEVILLE	IL	
4	1	1	1001622	SHOPRITE	MS JACKY DAGENHART	INFO SYSTEMS...	60 BEAVER BR...	LINCOLN PARK	NJ	
5	1	1	7121564	SPAR	MR BRADLY THOMAS	COMP SPEC	FORE STR	ST DENNIS	CORNWALL	
6	0.850000000000...	1	7121565	SPAR	MR PEARL GARRISON	SURVEYING TE...	FORE STR	ST AUSTELL	BUGLE, CORN...	
7	0.840000000000...	1	7121567	SPAR	MR EARL WILLIS	TECHNICIAN (C...	FORE STREET,...	BUNGLE	CORNWALL	
8	0.779999999999...	1	7121597	SPAR	MRS JERRY FIGURES	LEAD SYS ANAL...	6 FORE STR	BUDLEIGH SAL...	DEVON	
9	0.77	1	7121590	SPAR	MRS MINNA HASE	SR RESEARCH...	42 FORE STR	NEWTON ABBOT	BOVEY TRACEY.	
10	1	1	7121570	SPAR	MR GERMAYNE HANKS	SENIOR PROJE...	1 2 CHRISTINA...	TOTNESS	DEVON	
11	1	1	7121571	SPAR	MRS CHERIE ALBERSON	CONTROLLER	1 N STR	ASHBURTON	DEVON	
12	0.96	1	7121608	SPAR	MRS CHERIE ALBERSON	CONTROLLER	N STR	ASHBURTON	DEVON	
13	1	1	1001658	SPARTAN STOR...	MS MARTHA CHASSE	PROJECT MANA...	5161 W MAIN S...	KALAMAZOO	MI	
14	1	1	1001660	SPARTAN STOR...	MR DESMOND WINTERS	OWNER / CON...	1824 PORTAGE...	KALAMAZOO	MI	
15	1	1	1001659	SPARTAN STOR...	MR DESMOND GRINDLEY	SYSTEMS ANAL...	1824 PORTAGE...	KALAMAZOO	MI	
16	1	1	1001691	SPARTAN STOR...	MS SUZY TRAPANI	MGR ENG. SAL...	2545 W CADILA...	FARWELL	MI	
17	1	1	1001664	SPARTAN STOR...	MR HARDEN DALTON	OWNER / CON...	438 LINCOLN A...	IONIA	MI	
18	1	1	1001694	SPARTAN STOR...	MR REAMONN PELZER	CHIEF DBA	24445 DRAKE RD	FARMINGTON...	MI	
19	1	1	1001729	SUN MART	MS ZELDA DECHICK	MIS MANAGER	1100 13TH AVE E	WEST FARGO	ND	
20	1	1	1001724	SUN MART	MS MINERVA BAKER	COMPUTER SPE...	1921 W A STR	NORTH PLATTE	NE	
21	1	1	1001732	SUPER FRESH	MS MARGA JANOWSKI	PRESIDENT	2105 PHILADEL...	CLAYMONT	DE	
22	1	1	1001736	SUPER FRESH	MS BETTY EASTIP	ASSISTANT	2465 5 BROAD...	HAMILTON TO...	NJ	
23	1	1	1001738	SUPER FRESH	MS MINICA HOCH	MGR COORD	4330 48TH STR...	WASHINGTON	DC	
24	1	1	1001758	SUPER STOP &...	MR ANCLIN SAMSON	AUTOMATED S...	150 NEW PK AVE	HARTFORD	CT	
25	1	1	1001767	SUPERPETZ	MR EZARRAS DELAHANTY	LEAD PROGRA...	250 SHOUP MIL...	DAYTON	OH	
26	1	1	1001796	SUPERPETZ	MR RUSTICUS WHITACRE	DATABASE AD...	1275 W PATRIC...	FREDERICK	MD	
27	1	1	1001798	SUPERPETZ	MS TIMMIE NICE	NETWORK SYS...	2420 EASTERN...	YORK	PA	

The record contains the following fields:

Score

Match score that indicates the degree of similarity between a record and another record in the cluster. Records with a match score of one are duplicate records that do not need review. A cluster in which any record has a match score below the lower threshold is not a duplicate cluster.

Row_Identifier

A row number that uniquely identifies each row in the table. For this example, the row identifier contains the customer ID.

Cluster ID

A unique identifier for a cluster. Each record in a cluster receives the same cluster ID. The first four records in the sample output data are unique. Each record has a unique cluster ID. Rows five to nine belong to cluster five. Each record in this cluster is a duplicate record because of similarities in the address fields.

Source Data Fields

The Standard Output table group also receives all the source data fields.

Cluster Output

The Target_CustomerConsolidate table receives records from the Cluster Output group. The Cluster Output group returns records that might be duplicate records. The records in the Target_CustomerConsolidate table need manual review in the Analyst tool.

The following image shows some of the records and the fields in the Target_CustomerConsolidate table:

Output

Name: [lexc_Consolidate_Cluster_Data](#)

Ro...	Sequential...	Cluster_ID	Score	Is_Master	Workflow...	CUST_ID	COMPANY	CONTACT	TITLE	ADDR1	ADDR2
1	0	8	1	Y	DummyW...	7121580	SPAR	MR MICHAEL AVELINO	CONTROLLER	219 VIRGEN BELLA FLOR...	LEPE
2	1	8	1	N	DummyW...	7121580	SPAR	MR MICHAEL AVELINO	CONTROLLER	219 VIRGEN BELLA FLOR...	LEPE
3	2	8	0.81	N	DummyW...	7121585	SPAR	MR ROBERT ADAMS	MANAGER OF DBA	3 VIRGEN BELLA FDL	LEPE
4	3	26	1	Y	DummyW...	1001921	THE CORNER SHOP	MR DENIS LEE	MANAGER (\$\$ OF DBA	102 MID STR S NUTFIELD	REDHILL
5	4	26	1	N	DummyW...	1001921	THE CORNER SHOP	MR DENIS LEE	MANAGER (\$\$ OF DBA	102 MID STR S NUTFIELD	REDHILL
6	5	26	0.88	N	DummyW...	7121633	THE CORNER SHOP	MR BRADLY WYATT	MGR, DBA	102 MID STR S NUTFIELD	REDHILL
7	6	26	0.81	N	DummyW...	7121634	THE CORNER SHOP	MR BRADLY LOPEZ	OWNER/CONSULTANT	971 MID STR S NUTFIELD	REDHILL
8	7	74	1	Y	DummyW...	1001922	TOPS MARKETS	MS SARAD SACKRIDER	PRESIDENT	22777 ROCKSIDE RD	BEDFORD
9	8	74	1	N	DummyW...	1001922	TOPS MARKETS	MS SARAD SACKRIDER	PRESIDENT	22777 ROCKSIDE RD	BEDFORD
10	9	74	0.860000000000...	N	DummyW...	7121640	TOPS MARKETS	MS SARAH ALLEN	INDUSTRIAL ENGINEER	77 ROCKSIDE RD	BEDFORD
11	10	81	1	Y	DummyW...	1777777	WILLIAM WRIGLEY...	MR JUAN WISNIEWSKI	<null>	C. MIS 410 N MICHIGAN...	CHICAGO
12	11	81	1	N	DummyW...	1777777	WILLIAM WRIGLEY...	MR JUAN WISNIEWSKI	<null>	C. MIS 410 N MICHIGAN...	CHICAGO
13	12	81	0.93	N	DummyW...	1002174	WILLIAM WRIGLEY...	MR JUAN WISNIEWSKI	LEAD PROGRAMMER A...	CORPORATE MIS 410 N M...	CHICAGO
14	13	81	0.93	N	DummyW...	1002153	WILLIAM WRIGLEY...	MR JOHN WISNIEWSKI	MGR COORD	CORPORATE MIS 410 N M...	CHICAGO
15	14	81	1	N	DummyW...	1002211	WILLIAM WRIGLEY...	MR JOHN WISNIEWSKI	MGR COORD	C. MIS 410 N MICHIGAN...	CHICAGO
16	15	81	0.93	N	DummyW...	1002210	WILLIAM WRIGLEY...	MR JOHNNY WISNIEWSKI	BUSINESS ANALYST	CORPORATE MIS 410 N M...	CHICAGO
17	16	81	0.93	N	DummyW...	1002142	WILLIAM WRIGLEY...	MR JOHN WISNIEWSKI	MGR COORD	CORPORATE MIS 410 N M...	CHICAGO
18	17	106	1	Y	DummyW...	1000051	A&P	MS SARA ONEAL	BUSINESS MANAGER	120 MAIN RD	MANVILLE
19	18	106	1	N	DummyW...	1000051	A&P	MS SARA ONEAL	BUSINESS MANAGER	120 MAIN RD	MANVILLE
20	19	106	0.850000000000...	N	DummyW...	1000089	A&P	MR BOBBY SMYTHE	PARTNER	120 N MAYNE	MANVILLE
21	20	106	0.99	N	DummyW...	1000096	A&P	MS JENNY BEASLEY	LEAD SYSTEM ANALYST	120 N MAIN RD	MANVILLE
22	21	135	1	Y	DummyW...	1001628	SIMS DELTEC INC	MR KEN YOUNGQUIST	CONTROLLER	1265 GRAY FOX STR	SAINT PAUL
23	22	135	1	N	DummyW...	1001628	SIMS DELTEC INC	MR KEN YOUNGQUIST	CONTROLLER	1265 GRAY FOX STR	SAINT PAUL

The record contains the following fields:

Row_Identifier

A number that uniquely identifies each row in the table.

Sequential Cluster ID

A sequential identifier for each cluster to review in a Human task. The Duplicate Record Exception transformation adds the sequential cluster ID to records in the Cluster Data output group.

Cluster ID

A unique identifier for a cluster. The Match transformation assigns a cluster ID to all the output records. Duplicate records and possible duplicate records share a cluster ID. A unique record receives a cluster ID, but the record does not share the ID number with any other record.

Score

Match score that indicates the degree of similarity between a record and another record in the cluster. Records that require manual review have match scores less than .95 and greater than .80.

Is Master

Indicates whether the record is the preferred record in the cluster.

WorkflowID

The WorkflowID is DummyWorkflowID because the transformation is not in a workflow.

Record Fields

The other fields in the record contain the customer source data.

Creating a Duplicate Record Exception Transformation

When you configure a Duplicate Record Exception transformation, configure the input ports. Define the upper and lower thresholds for determining matches. Configure where to write the duplicate records and the unique records.

1. Create a reusable or a nonreusable Duplicate Record Exception transformation.

- To create a reusable transformation, select **File > New > Transformation** and select a Duplicate Record Exception transformation.
 - To create a nonreusable transformation, open a mapping and add the transformation to the mapping canvas. Select a Duplicate Record Exception transformation from the wizard.
2. Click **Next**, or click **Finish**.
If you click **Finish**, you can update the default threshold values and the data routing options before you create the transformation.
 3. In the Configuration view, configure the upper and lower match score thresholds.
 4. In the **Data Routing Options** section, configure the standard output and exception table properties to set where the transformation writes each type of record.
Optionally, modify where to write the duplicate records, duplicate records to review, and unique records.
 5. Optionally, generate a unique records table. Enter the database connection and table name information for the new table. When you generate a unique records table, the transformation creates a database object in the Model repository.
 6. Configure the input ports. When you add an input port, the Developer tool adds the same port name to the output groups.
 - If you create a reusable transformation, select the **Ports** tab, and add ports for the data you want to connect to the transformation.
 - If you create a nonreusable transformation, add other objects to the mapping canvas and drag input ports to the transformation.
 7. Connect the transformation output ports to one or more data targets. Connect the output ports to the data objects that correspond to the output options you set on the **Configuration** view.
 - If you create a reusable transformation, add the transformation to a mapping and connect the output ports.
 - If you create a nonreusable transformation, the transformation connects the ports to the Cluster Data table. You must connect output ports to the other data targets.

CHAPTER 16

Expression Transformation

This chapter includes the following topics:

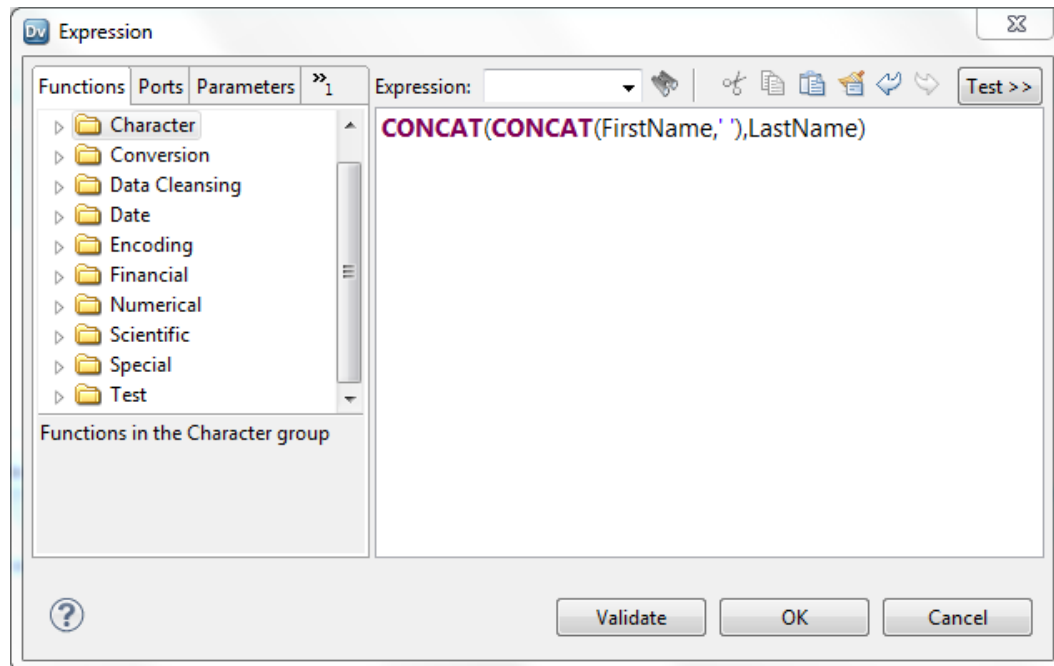
- [Expression Transformation Overview, 266](#)
- [Expression Transformation Ports, 267](#)
- [Test Expressions, 268](#)
- [Port Selectors, 269](#)
- [Windowing, 272](#)
- [Dynamic Expressions, 276](#)
- [Flatten a Dynamic Struct, 280](#)
- [Expression Transformation Advanced Properties, 281](#)
- [Expression Transformation in a Non-native Environment, 281](#)

Expression Transformation Overview

The Expression transformation is a passive transformation that you can use to perform calculations or to test conditional statements in a row. In non-reusable Expression transformations, you can define a mapping output expression to aggregate when you define mapping outputs.

In a single row, you might need to create an expression to adjust employee salaries, to concatenate first and last names, or to convert strings to numbers.

The following figure shows an expression in an Expression transformation that concatenates the first name, a space, and the last name:



You can enter multiple expressions in an Expression transformation by creating an expression for each output port. For example, you might want to calculate different types of taxes from each employee paycheck, such as the local and the federal income tax. Both tax calculations require the employee salary and a tax rate. Define a separate output port for each calculation. Define a different expression for each output port. You can define pass-through ports for the salary and tax rate because the port values do not change.

Expression Transformation Ports

An Expression transformation has different port types that you can reference when you define expressions.

An Expression transformation has the following port types:

Input

Receives data from upstream transformations. If the Expression transformation does not change the port value, you can define a pass-through port instead of an input port.

Output

Contains the return value of the expression. You enter the expression as a configuration option for the output port. You can also configure a default value for each port.

Note: If an expression results in numerical errors, such as division by zero or SQRT of a negative number, it returns an infinite or a NaN value.

Pass-Through

Define a pass-through port to pass the data through the transformation without changing the value. You can reference a pass-through port in a calculation, but you cannot change the data value in the pass-through port.

Variable

Temporarily stores data to use in expressions. You can store data across multiple rows. You can define an expression to return a value to a variable port.

Dynamic Port

Receives or returns ports in a dynamic mapping. A dynamic port can receive one or more columns from an upstream transformation and create a generated port for each column. A dynamic output port can return one or more generated ports. You can define input rules to determine which columns a dynamic port receives. A dynamic output port can contain an expression that generates multiple output ports.

Generated Port

A port that represents a single column within a dynamic port. The generated ports in the Expression transformation might change based on the columns that the Expression transformation receives from an upstream transformation.

Test Expressions

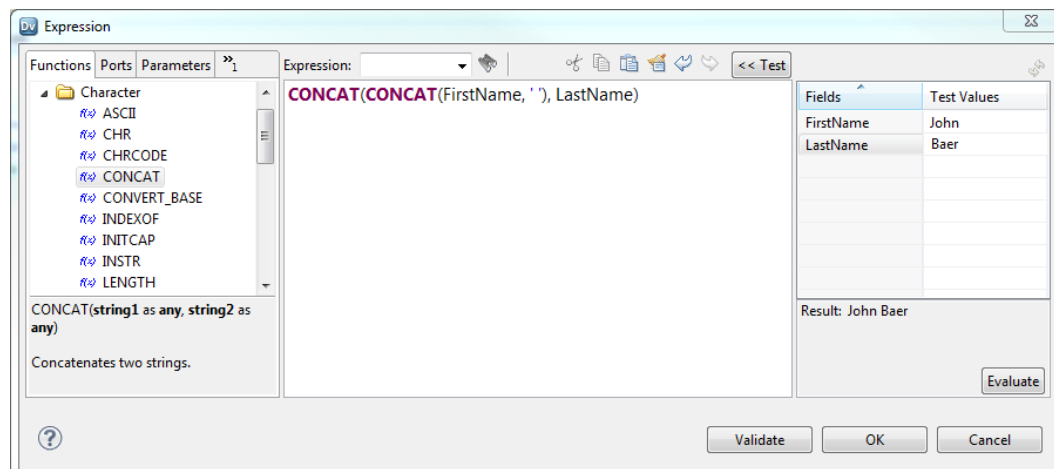
You can test expressions that you configure in the Expression Editor. When you test an expression, you enter sample data and then evaluate the expression.

You can test expressions when you configure expressions in the following ways:

- In an output or variable port in the Expression transformation
- In the Mapping Outputs view of an Expression transformation after adding the transformation to a mapping

For example, after configuring an expression that concatenates the first name, a space, and the last name, you can enter sample data for the ports, and then evaluate the expression to verify the result.

The following image shows the results of an expression that concatenates a sample first name and last name:



Date Format Strings for Sample Data

When you test an expression that uses a port with the Date/Time or Timestamp with Time Zone data type, you must enter sample data for the port using the required date format string.

To enter sample data for a port with the Date/Time data type, use the format MM/DD/YYYY HH24:MI:SS. When you evaluate the expression, the Expression Editor displays the result using the format that you specify in the expression. If you omit the format string in the expression, the Expression Editor displays the result using the same format MM/DD/YYYY HH24:MI:SS.

To enter sample data for a port with the Timestamp with Time Zone data type, use the format MM/DD/YYYY HH24:MI:SS TZR. When you evaluate the expression, the Expression Editor displays the result using the format YYYY-MM-DD HH24:MI:SS.NS TZR.

Testing an Expression

Test an expression in the Expression Editor to evaluate the expression and verify the result.

1. Open the Expression Editor in one of the following ways:
 - In an Expression transformation, click the **Open** button (🔑) in the **Expression** column for an output port or a variable port.
 - Select an Expression transformation included in a mapping. In the **Mapping Outputs** view, click the **Open** button (🔑) in the **Expression** column for an output.
2. Configure the expression.
3. Click **Test >>** to open the testing panel.
4. Enter sample data for each field in the **Test Values** column.

You can enter test values for each port or parameter included in the expression.
5. Click **Evaluate**.

The expression result displays at the bottom of the testing panel.

Port Selectors

When a transformation has generated ports, you need to configure the transformation to run successfully when the generated ports change. You can use a port selector to determine which ports to use in a dynamic expression, a lookup condition, or a joiner condition.

A port selector is an ordered list of ports that you can reference in an expression. When the generated ports change in a dynamic mapping, the port selector can contain different ports.

For example, the following expression references a generated port in a dynamic mapping:

```
Salary * 12
```

You configure the mapping to use dynamic sources, but the column that contains salary information in each source file has a different name. The column names are `Salary`, `Monthly_Salary`, or `Base_Salary`.

You perform the following tasks to accommodate the different column names:

1. Create a port selector named "Salary_PortSelector."
2. Create a selection rule to accept any port name with the suffix "Salary."

- Change the expression to include the port selector name instead of the salary column name. The expression has the following syntax:

```
Salary_PortSelector * 12
```

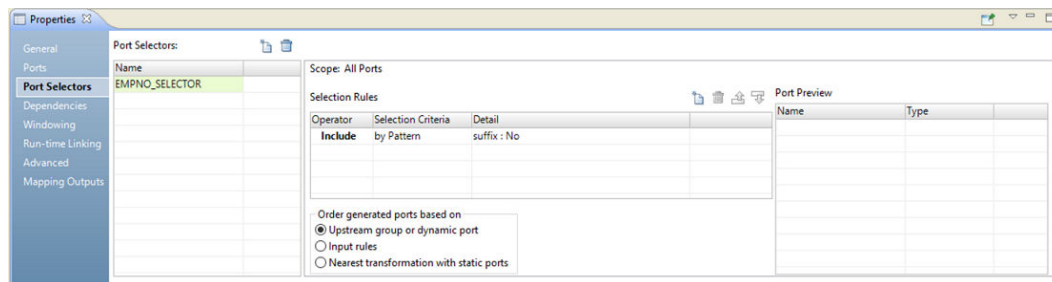
The expression runs successfully with any of the salary port names.

Port Selector Configuration

When you configure a port selector, you define selection rules to determine which generated ports to include. The selection rules are similar to the input rules that you can configure for dynamic ports.

A port selector can include static or generated ports. Configure a port selector on the **Port Selector** tab.

The following image shows the **Port Selector** tab:



Configure the following properties for a port selector:

Name

Identifies the port selector. You can create multiple port selectors in a transformation and reference them in expressions.

Scope

Identifies a group of ports that the port selector applies to. You must choose the scope when you create a port selector for a Joiner or a Lookup transformation. These transformations have multiple input groups. The Joiner transformation has a Master or a Detail scope. The Lookup transformation has an Import or a Lookup scope. The Expression transformation has one input group. The scope is always All Ports.

Selection Rules

Determines the ports to include in the port selector. When you create the selection rules, the **Port Preview** panel shows the ports that qualify from the current input ports. These ports might change. Configure the selection rules to accommodate ports from different sources.

Selection Rules

The selection rules associated with a port selector determine the ports to include in the port selector.

When you create the selection rules, the **Port Preview** panel shows the ports that qualify from the current input ports. These ports might change. Configure the selection rules to accommodate ports from different sources.

Create selection rules based on the following criteria:

Operator

Includes or excludes the ports that selection rules return. Default is include. You must include ports before you can exclude ports.

Selection Criteria

The type of selection rule you want to create. You can create a rule based on the column name, port type, pattern, or complex data type definition. To include ports based on the column name, search for specific names or search for a pattern of characters in the name.

Detail

The values to apply to the selection criteria. If the selection criteria is by column name, configure the string or name to search for. If the selection criteria is by port type, select the port types to include.

The following table describes the selection criteria and how to specify the details for the criteria:

Selection Criteria	Description	Detail
All	Includes all ports.	No details required.
Name	Filters ports based on the port name.	Select the port names from a list of values or use a parameter of type Port or Port List.
Type	Filters ports based on the data type of each port.	Select data types from a list.
Pattern	Filters ports by a string of characters in the name or by a regular expression.	Choose prefix, suffix, or regular expression as the pattern type for the port name. Then, enter a value for the pattern or use a parameter of type String.
Complex Data Type Definition	Filters ports by a complex data type definition.	Choose prefix, suffix, or regular expression as the pattern type for the complex data type definition. Then, enter a value for the pattern or use a parameter of type String.

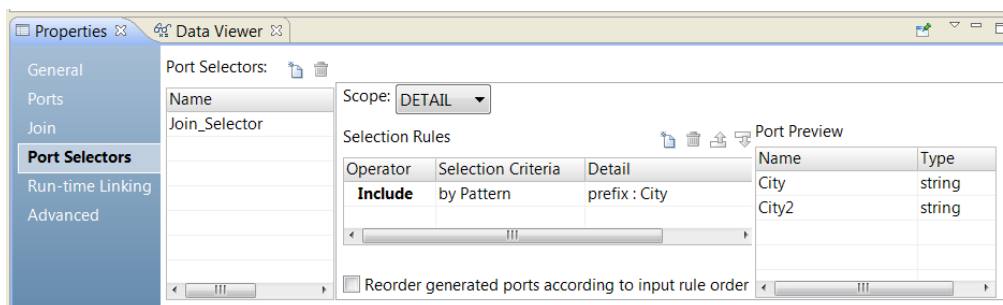
Creating a Port Selector

Create a port selector to determine which ports to use in a dynamic expression, a lookup condition, or a joiner condition.

1. Click the **Port Selectors** tab.
2. In the **Port Selectors** area, click **New**.
The Developer tool creates a port selector with a default selection rule that includes all ports.
3. In the **Port Selectors** area, change the port selector name to a unique name.
4. If you are working on the Joiner transformation or the Lookup transformation, choose the scope.
The available ports change based on the group of ports that you choose.
5. In the **Selection Rules** area, select an **Operator**.
 - Include. Create a rule that includes ports for the port selector. You must include ports before you can exclude ports.
 - Exclude. Create a rule that excludes specific ports from the port selector.
6. Choose the **Selection Criteria**.
 - By Name. Select specific ports by name. You can select the port names from a list of ports in the scope.
 - By Type. Select ports by type. You can select one or more data types.

- By Pattern. Select ports by a pattern of characters in the port name. You can search with specific characters or you can create a regular expression.

The following image shows the Port Selector tab:



7. Click the **Detail** column.

The **Input Rule Detail** dialog box appears.

8. Select the values to filter ports by.

- By Name. Choose to create a port list by value or by a parameter. Click **Choose** to select the ports in the list.
- By Type. Select one or more data types from a list. The **Port Preview** area shows ports of the types that you select.
- By Pattern. Choose to search the prefix or suffix of the port name for a specific pattern of characters. Or, choose to create a regular expression to search with. Configure a parameter or configure the pattern to search with.

The **Port Preview** area shows the ports in the port selector as you configure the rules.

9. To reorder the ports in the port selector, select **Reorder generated ports according to the input rule order**.

Windowing

When a transformation contains a window function, you need to configure the windowing properties. Windowing is available for transformations on the Spark engine only.

Window functions operate on a group of rows and calculate a return value for every input row.

Before you define a window function in an Expression transformation, you need to describe the window by configuring the windowing properties. Windowing properties include a frame specification, partition keys, and order keys. The frame specification states which rows are included in the overall calculation for the current row. The partition keys determine which rows are in the same partition. The order keys determine how rows in a partition are ordered.

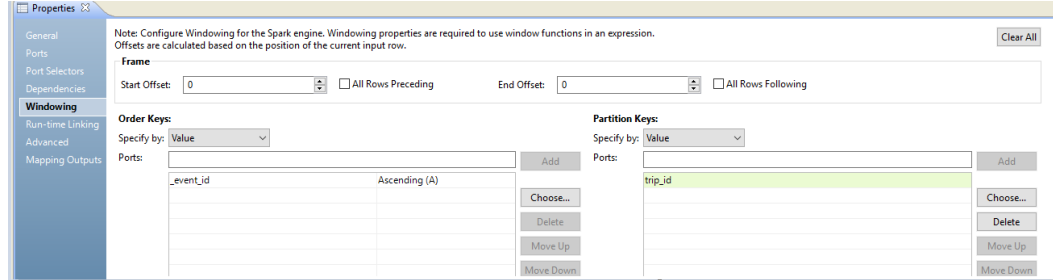
After you configure windowing properties, you define a window function in the Expression transformation. Informatica supports the window functions LEAD and LAG. You can also use aggregate functions as window functions in an Expression transformation.

Windowing Configuration

When you include a window function in an Expression transformation, you configure the windowing properties associated with the function. Windowing properties define the partitioning, ordering, and frame boundaries associated with a particular input row.

Configure a transformation for windowing on the Windowing tab.

The following image shows the Windowing tab:



You configure the following groups of properties on the Windowing tab:

Frame

Defines the rows that are included in the frame for the current input row, based on physical offsets from the position of the current input row.

You configure a frame if you use an aggregate function as a window function. The window functions LEAD and LAG reference individual rows and ignore the frame specification.

Partition Keys

Separate the input rows into different partitions. If you do not define partition keys, all rows belong to a single partition.

Order Keys

Define how rows in a partition are ordered. The ports you choose determine the position of a row within a partition. The order key can be ascending or descending. If you do not define order keys, the rows have no particular order.

Frame

The frame determines which rows are included in the calculation for the current input row, based on their relative position to the current row.

If you use an aggregate function instead of LEAD or LAG, you must specify a window frame. LEAD and LAG reference individual row and ignore the frame specification.

The start offset and end offset describe the number of rows that appear before and after the current input row. An offset of "0" represents the current input row. For example, a start offset of -3 and an end offset of 0 describes a frame including the current input row and the three rows before the current row.

The following image shows a frame with a start offset of -1 and an end offset of 1:

Type	Category	Revenue
Action	Video game	1000
Arcade	Video game	1000
Sports	Video game	2000
Adventure	Video game	3000
Strategy	Video game	4000

Current input row →

← 1 PRECEDING

← 1 FOLLOWING

For every input row, the function performs an aggregate operation on the rows inside the frame. If you configure an aggregate expression like SUM with the preceding frame, the expression calculates the sum of the values within the frame and returns a value of 6000 for the input row.

You can also specify a frame that does not include the current input row. For example, a start offset of 10 and an end offset of 15 describes a frame that includes six total rows, from the tenth to the fifteenth row after the current row.

Note: The start offset must be less than or equal to the end offset.

Offsets of **All Rows Preceding** and **All Rows Following** represent the first row of the partition and the last row of the partition. For example, if the start offset is All Rows Preceding and the end offset is -1, the frame includes one row before the current row and all rows before that.

The following figure illustrates a frame with a start offset of 0 and an end offset of All Rows Following:

Genre	Recordings	Revenue
Jazz	233	5000
Gospel	214	1000
Country	145	2000
Ethnic	154	9000
Pop	317	4000
Rock	237	2100
Classical	221	3200
EDM	153	950
Hip Hop	839	2300
Punk	415	7650

Current input row →

All Rows Following

Partition and Order Keys

Configure partition and order keys to form groups of rows and define the order or sequence of rows within each partition.

Use the following keys to specify how to group and order the rows in a window:

Partition keys

Configure partition keys to define partition boundaries, rather than performing the calculation across all inputs. The window function operates across the rows that fall into the same partition as the current row.

You can specify the partition keys by value or parameter. Select **Value** to use port names. Choose **Parameter** to use a sort key list parameter. A sort key list parameter contains a list of ports to sort by. If you do not specify partition keys, all the data is included in the same partition.

Order keys

Use order keys to determine how rows in a partition are ordered. Order keys define the position of a particular row in a partition.

You can specify the order keys by value or parameter. Select **Value** to use port names. Choose **Parameter** to use a sort key list parameter. A sort key list parameter contains a list of ports to sort by. You must also choose to arrange the data in ascending or descending order. If you do not specify order keys, the rows in a partition are not arranged in any particular order.

Example

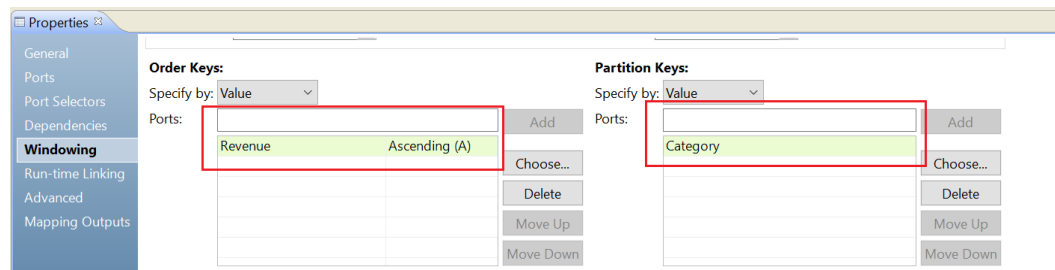
You are the owner of a coffee and tea shop. You want to calculate the best-selling and second best-selling coffee and tea products.

The following table lists the products, the corresponding product categories, and the revenue from each product:

Product	Category	Revenue
Espresso	Coffee	600
Black	Tea	550
Cappuccino	Coffee	500
Americano	Coffee	600
Oolong	Tea	250
Macchiato	Coffee	300
Green	Tea	450
White	Tea	650

You partition the data by category and order the data by descending revenue.

The following image shows the properties you configure on the Windowing tab:



The following table shows the data grouped into two partitions according to category. Within each partition, the revenue is organized in descending order:

Product	Category	Revenue
Espresso	Coffee	600
Americano	Coffee	600
Cappuccino	Coffee	500
Macchiato	Coffee	300
White	Tea	650
Black	Tea	550
Green	Tea	450
Oolong	Tea	250

Based on the partitioning and ordering specifications, you determine that the two best-selling coffees are espresso and Americano, and the two best-selling teas are white and black.

Rules and Guidelines for Windowing Configuration

Certain guidelines apply when you configure a transformation for windowing.

Consider the following rules and guidelines when you define windowing properties for a window function:

- When you configure a frame, the start offset must be less than or equal to the end offset. Otherwise, the frame is not valid.
- Configure a frame specification if you use an aggregate function as a window function. LEAD and LAG operate based on the offset value and ignore the frame specification.
- You cannot use complex ports as partition or order keys.
- Assign unique port names to partition and order keys to avoid run-time errors.
- The partition and order keys cannot use both a dynamic port and one or more generated ports of the same dynamic port. You must select either the dynamic port or the generated ports.

Dynamic Expressions

When you configure an expression in a dynamic output port, the expression becomes a dynamic expression. A dynamic expression can generate multiple output ports.

You can reference a port selector or a dynamic port in a dynamic expression. When the port selector or dynamic port contains multiple ports, the dynamic expression runs against each port.

When you configure a dynamic expression, the Developer tool does not validate if the generated ports are valid types for the expression. For example, if you reference a port selector that contains decimal type ports in an expression that requires string types, the expression appears as valid at design time.

Example

An Expression transformation has the following generated input ports:

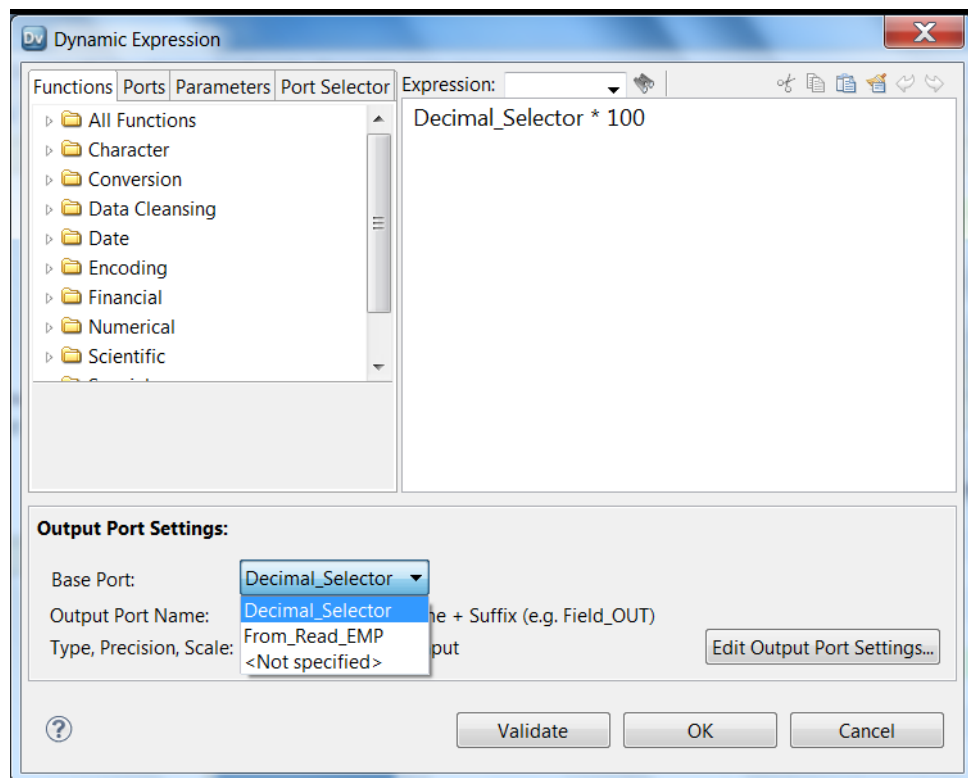
```
EMPNO    Decimal
NAME     String
SALARY   Decimal
DEPTNO   Decimal
```

The transformation contains a dynamic output port called MyDynamicPort. The output port returns the results of a dynamic expression. The dynamic expression multiplies the value of each port in a port selector by 100. The expression runs one time for each port in the port selector. Each instance can return a different result. The Expression transformation generates a separate output port for each result.

The Decimal_Selector port selector has a selection rule that includes the ports that are of decimal data type:

```
EMPNO    Decimal
SALARY   Decimal
DEPTNO   Decimal
```

The following image shows a dynamic expression that references the Decimal_Selector port selector:



Edit the output port settings to change output port names and output port properties. You can also choose the base port.

Output Port Settings

You can indicate which ports you want to use as input to a dynamic expression. Select the ports in the **Base Port** area.

If you select the Decimal_Selector port selector as the base port, the dynamic expression returns decimal type ports. The dynamic expression does not generate a port for the NAME port because it is a string.

The following image shows the generated ports in the transformation:

Name	Type	Precis...	Scale	Input	Output	Varia...	Expression
1 From_Read_EMP	dynamic		0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	From_Read_EMP
1 EMPNO	decimal	3	0				
2 NAME	string	10	0				
3 SALARY	decimal	4	0				
4 DEPTNO	decimal	4	0				
2 MyDynamicPort	dynamic		0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Decimal_Selector * 100
1 EMPNO_OUT	decimal	3	0				
2 SALARY_OUT	decimal	4	0				
3 DEPTNO_OUT	decimal	4	0				

Although the From_Read_Emp dynamic port is an input-output port, the transformation returns just the ports in the MyDynamicPort dynamic output port.

You can configure how you want to name the output ports. The default output port name is the input port name and the suffix `_OUT`.

You can change the base port to a port selector.

The following image shows the output port settings in the Expression Editor:

Output Port Settings:

Base Port:

Output Port Name: Primary input port name + Suffix (e.g. Field_O

Type, Precision, Scale: Inherit from primary input

If you configure the base port as From_Read_EMP, you select the dynamic port that contains all the generated input ports. The Data Integration Service runs the dynamic expression against all the ports in From_Read_EMP.

The following image shows the generated output ports based on the From_Read_Emp input:

Name	Type	Precis...	Scale	Input	Output	Varia...	Expression
1 From_Read_EMP	dynamic		0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	From_Read_EMP
1 EMPNO	decimal	3	0				
2 NAME	string	10	0				
3 SALARY	decimal	4	0				
4 DEPTNO	decimal	4	0				
2 MyDynamicPort	dynamic		0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Decimal_Selector * 100
1 EMPNO_OUT	decimal	3	0				
2 NAME_OUT	string	10	0				
3 SALARY_OUT	decimal	4	0				
4 DEPTNO_OUT	decimal	4	0				

The generated output ports include an output port called NAME_OUT, which is a string type.

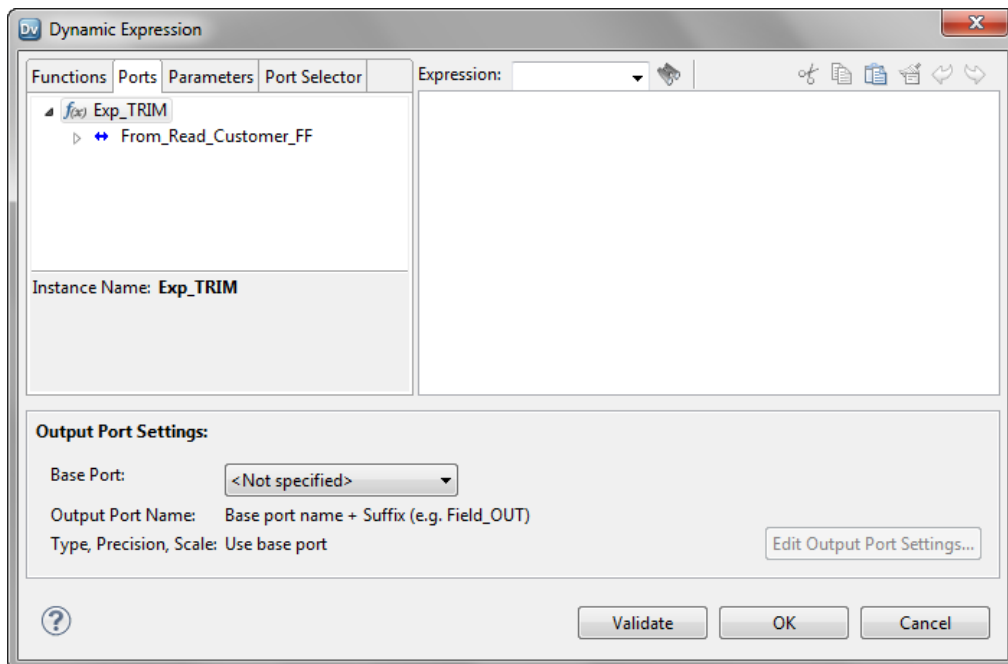
The Data Integration Service generates output ports for each dynamic expression. If you create a dynamic expression that generates 15 ports and you define another dynamic expression that generates 5 ports, the Data Integration Service generates 20 output ports. Each dynamic output port generates a different group of ports.

Creating a Dynamic Expression

Create a dynamic expression in an Expression transformation to run the expression one time for each port in a dynamic port or a port selector. The dynamic expression returns the results to a separate generated port for each instance.

1. In the Expression transformation, go to the **Properties** view and click the **Ports** tab.
2. Click **New Dynamic Port**.
The Developer tool creates a dynamic port with default properties.
3. Rename the dynamic port and disable the input option.
The dynamic port must be an output port.
4. In the **Expression** column for the dynamic output port, click the **Open** button (🔗).

The **Dynamic Expression** dialog box appears:

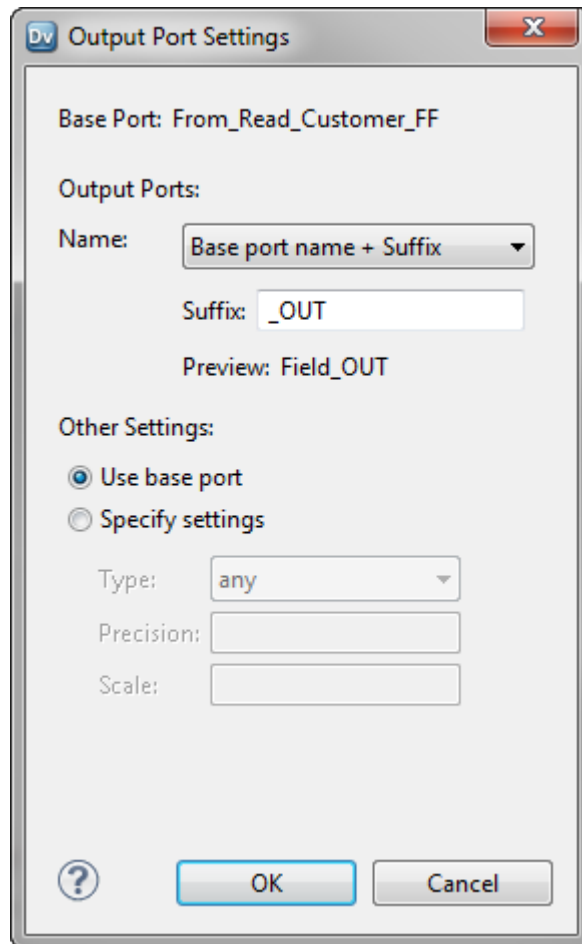


5. In the Expression editor, enter an expression. The expression can include a port selector or a dynamic port.
For example, `LTRIM(RTRIM(Dynamic_Customer))`, where `Dynamic_Customer` is a dynamic port.
6. Click **Validate** to validate the expression.
7. Click **OK** to exit the **Validate Expression** dialog box.
8. In the **Output Port Settings** area, select the dynamic output port from the **Base Port** list or choose a port selector that you referenced in the expression.

The Developer tool generates output ports based on what you select.

9. Use the following steps to rename the output ports:
 - a. Click **Edit Output Port Settings**.

The **Output Port Settings** dialog box appears.



- b. In the **Name** list, select one of the options and enter a value for the prefix or suffix. If you selected **Fixed string + Auto-number**, enter the text for output port name. For example, if you enter TRIM for the output port name, the output port names appear as TRIM1, TRIM2, TRIM3.
 - c. Optionally, choose **Specify settings** in the **Other Settings** area to change the type, precision, and scale for the output ports. By default, the output ports use the settings of the base ports.
 - d. Click **OK**.
10. Click **OK** to exit the **Dynamic Expression** editor.

Flatten a Dynamic Struct

Use the `EXTRACT_STRUCT` function to flatten dynamic structs in an Expression transformation.

Use the dot operator in expressions to extract elements of a struct.

For example, you can use the EXTRACT_STRUCT function to extract all data from an address without specifying each address field individually.

Port	Name	Type	Type Configuration	Precisi...	Scale	Input	Output	Variable	Expression	Default Value	Description	Input Rules
1	AddressInput_RegularStruct	struct	(Address)			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	AddressInput_RegularStruct			
1	Street	string	N/A	10	0							
2	City	string	N/A	10	0							
3	State	string	N/A	10	0							
4	Zip	string	N/A	10	0							
2	AddressInput_DynamicStruct	dynamic struct	N/A			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	AddressInput_DynamicStruct			Include All
1	Street	string	N/A	10	0							
2	City	string	N/A	10	0							
3	State	string	N/A	10	0							
4	Zip	string	N/A	10	0							
3	City_ExtractFromRegularStruct	string	N/A	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	AddressInput_RegularStruct.City		ERROR[transf...	
4	City_ExtractFromDynamicStruct	string	N/A	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	AddressInput_DynamicStruct.City		ERROR[transf...	
5	ExtractAll_FromDynamicStruct	dynamic	N/A			<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	EXTRACT_STRUCT(AddressInput_DynamicStruct)		ERROR[transf...	
1	Street_OUT	string	N/A	10	0							
2	City_OUT	string	N/A	10	0							
3	State_OUT	string	N/A	10	0							
4	Zip_OUT	string	N/A	10	0							

Expression Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Expression transformation.

Configure the following advanced properties for an Expression transformation:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Maintain Row Order

Maintain the row order of the input data to the transformation. Select this option if the Data Integration Service should not perform any optimization that can change the row order.

When the Data Integration Service performs optimizations, it might lose an order established earlier in the mapping. You can establish order in a mapping with a sorted flat file source, a sorted relational source, or a Sorter transformation. When you configure a transformation to maintain row order, the Data Integration Service considers this configuration when it performs optimizations for the mapping. The Data Integration Service performs optimizations for the transformation if it can maintain the order. The Data Integration Service does not perform optimizations for the transformation if the optimization would change the row order.

Expression Transformation in a Non-native Environment

The Expression transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Expression Transformation on the Blaze Engine

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

An Expression transformation with a user-defined function returns a null value for rows that have an exception error in the function.

Expression Transformation on the Spark Engine

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Note: If an expression results in numerical errors, such as division by zero or SQRT of a negative number, it returns a null value and rows do not appear in the output. In the native environment, the expression returns an infinite or a NaN value.

Expression Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

You cannot use the EXTRACT_STRUCT function for streaming mappings.

Expression Transformation on the Databricks Spark Engine

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Note: If an expression results in numerical errors, such as division by zero or SQRT of a negative number, it returns a null value and rows do not appear in the output. In the native environment, the expression returns an infinite or a NaN value.

CHAPTER 17

Filter Transformation

This chapter includes the following topics:

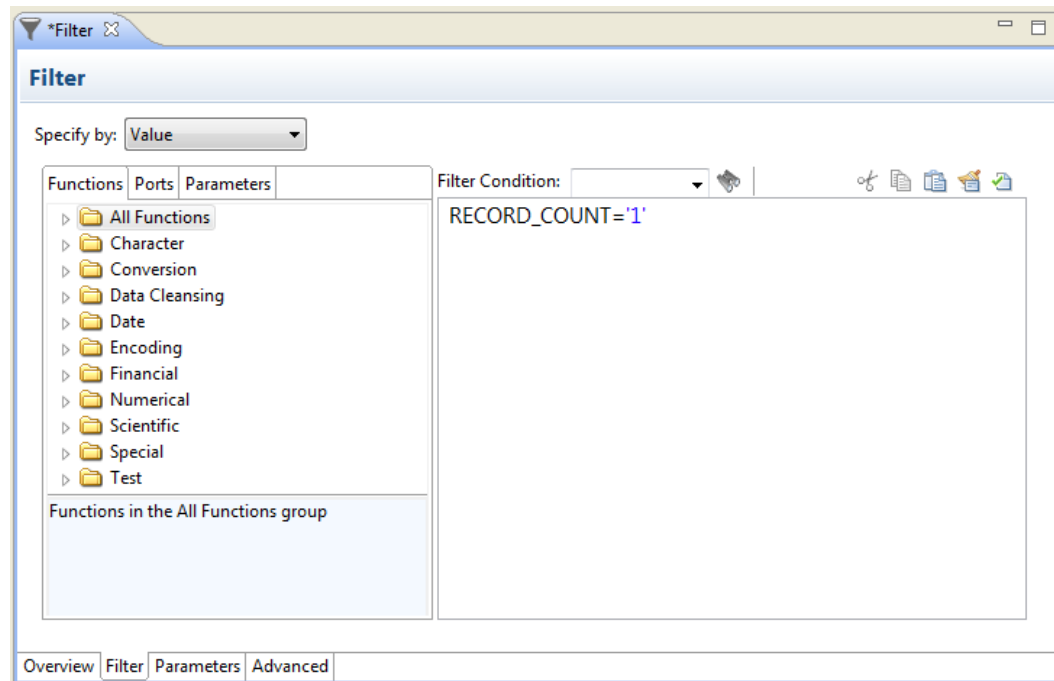
- [Filter Transformation Overview, 283](#)
- [Filter Transformations in Dynamic Mappings, 284](#)
- [Filter Condition, 285](#)
- [Filter Transformation Advanced Properties, 287](#)
- [Filter Transformation Performance Tips, 287](#)
- [Filter Transformation in a Non-native Environment, 287](#)

Filter Transformation Overview

Use the Filter transformation to filter out rows in a mapping. As an active transformation, the Filter transformation may change the number of rows passed through it.

The Filter transformation allows rows that meet the specified filter condition to pass through. It drops rows that do not meet the condition. You can filter data based on one or more conditions.

The following image shows a filter condition in a Filter transformation:



A filter condition returns TRUE or FALSE for each row that the Data Integration Service evaluates, based on whether a row meets the specified condition. For each row that returns TRUE, the Data Integration Services pass through the transformation. For each row that returns FALSE, the Data Integration Service drops and writes a message to the log.

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.

Filter Transformations in Dynamic Mappings

You can use an Filter transformation in a dynamic mapping. You can configure dynamic ports in the transformation and reference the generated ports in the filter condition.

You can parameterize the complete filter condition. Configure an expression parameter with a default value that contains the entire expression. The Developer tool does not validate a filter condition in a parameter default value.

You can reference a dynamic port in a filter condition. The dynamic port can contain multiple generated ports. The Data Integration Service expands the filter condition to contain each generated port. Each generated port must be a valid type to include in the expression.

You can reference a generated port in a filter condition. However, if the generated port does not exist at run time, the mapping fails.

Filter Condition

The filter condition is an expression that returns TRUE or FALSE.

Enter conditions in the Expression editor. The filter condition is case sensitive.

You can use any expression that returns a single value as a filter. For example, if you want to filter out rows for employees whose salary is less than or equal to \$30,000, 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, enter the following condition:

```
SALARY > 30000 AND SALARY < 100000
```

You can use ports, parameters, dynamic ports, and generated ports in the filter condition. Select the ports and the parameters in the Expression editor.

If you use a dynamic port in the filter condition, the filter condition expands to include all the generated ports in the dynamic port. For example, the dynamic port, MyDynamicPort, contains three decimal ports:

```
Salary  
Bonus  
Stock
```

If you configure the following filter condition:

```
MyDynamicPort > 100
```

The filter condition expands to the following expression:

```
Salary > 100 AND Bonus > 100 AND Stock > 100
```

You can 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, the transformation contains a port named NUMBER_OF_UNITS with a numeric data type. You configure a filter condition to return FALSE if the value of NUMBER_OF_UNITS equals zero. Otherwise, the condition returns TRUE.

Note: You cannot use a single port selector or dynamic port as a boolean value.

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 FALSE.

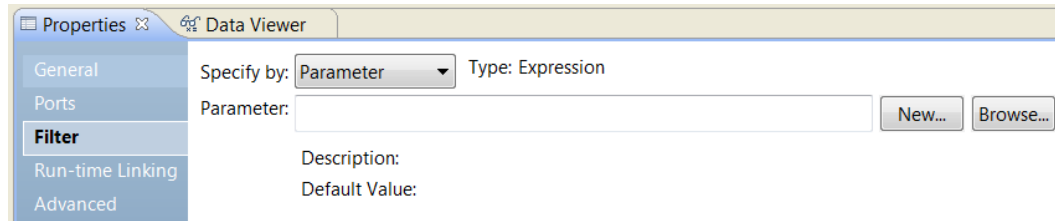
Parameterize the Filter Condition

You can configure an expression parameter to define the filter condition. An expression parameter contains the entire expression.

You might need to parameterize the filter condition when the Filter transformation is in a dynamic mapping. The filter condition might change based on the generated ports in the transformation at run time.

To use an expression parameter for the filter condition, choose **Specify by Parameter** on the **Filter** tab of the Filter transformation properties.

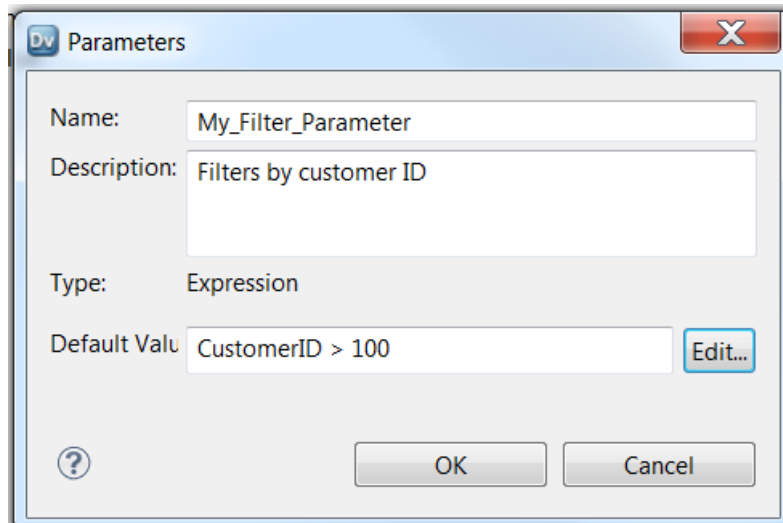
The following image shows the **Filter** tab when you specify the filter condition with a parameter:



You can browse for and select an expression parameter that you already created. Or, you can create an expression parameter.

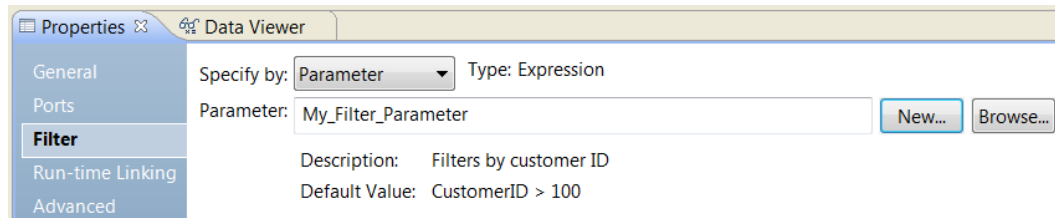
To create an expression parameter, click **New**. Enter a name for the parameter, a description, and the default expression value.

The following image shows where you enter the parameter:



You can enter a default expression on the Parameters dialog box. If you want to use an Expression editor, click Edit. When you use the Expression editor, you can select functions and ports to use in the expression. You can validate the expression.

The following image shows the Filter tab with a parameter for the filter condition:



The expression parameter is a mapping parameter. You can override the parameter in a parameter set or a parameter file at run time.

Filtering Rows with Null Values

To filter 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 NULL value 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.

Filter Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Filter transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Filter Transformation Performance Tips

Use tips to increase Filter transformation performance.

Use the Filter transformation early in the mapping.

Keep the Filter transformation as close as possible to the sources in the mapping. Instead of 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.

Filter Transformation in a Non-native Environment

The Filter transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Filter Transformation on the Blaze Engine

When a mapping contains a Filter transformation on a partitioned column of a Hive source, the Blaze engine can read only the partitions that contain data that satisfies the filter condition. To push the filter to the Hive source, configure the Filter transformation to be the next transformation in the mapping after the source.

CHAPTER 18

Hierarchical to Relational Transformation

This chapter includes the following topics:

- [Hierarchical to Relational Transformation Overview, 289](#)
- [Example - Hierarchical to Relational Transformation, 289](#)
- [Output Relational Ports and the Overview View, 291](#)
- [Hierarchical to Relational Transformation Ports, 292](#)
- [Schema References, 293](#)
- [Port Configuration, 293](#)
- [Hierarchical to Relational Transformation Development, 293](#)

Hierarchical to Relational Transformation Overview

The Hierarchical to Relational transformation processes XML or JSON hierarchical input and transforms it into relational output. A Hierarchical to Relational transformation reads hierarchical input from input ports and transforms the data to relational output at the transformation output ports. To transform hierarchical input to relational output, use a schema file to define the hierarchical data.

You can use the Hierarchical to Relational transformation wizard to automatically map the data. You can configure the mapping to the relational output ports on the transformation **Overview** view.

After the wizard generates the transformation, you can pass the data from the relational output ports to another transformation in a mapping.

Note: The Hierarchical to Relational transformation can process up to 10,000 schema elements in an .xsd file. To process more than 10,000 elements, split the data into multiple files.

Example - Hierarchical to Relational Transformation

The Logistics department of the Harrinder Shipping company must process shipment data. They need to transform inventory and customer data from hierarchical format into relational data that they can store in database tables.

They need to create a mapping that transforms hierarchical data into relational data. The organization inventory system generates shipment inventory data in hierarchical format. The mapping needs to use a Hierarchical to Relational transformation that inputs shipment data and outputs the details in a usable relational format.

The Shipments input is in hierarchical format. The Shipment element contains sub-elements with customer and inventory data for each shipment:

```

Shipments
  Shipment
    Items
      Item_Name
      Inventory_ID
    Customer
      Customer_Name
      Customer_ID
      Customer_Address

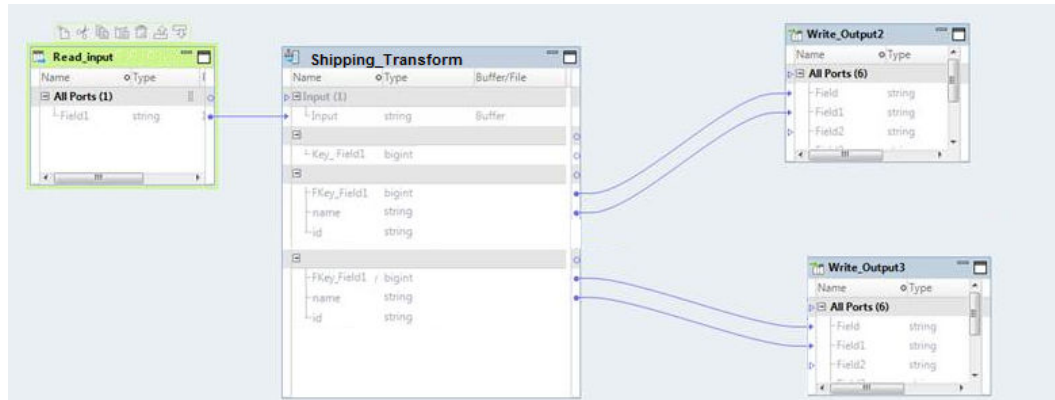
```

In the relational output, the Customer_ID element is a primary key in the Customer table, and is a Foreign key in the Shipment table.

Customer_ID	Customer_Name	Customer_Address
3543766	Tony Birch	6 Moby Drive
6342562	Sujita Man	22 Dan Street
6471862	Dwayne Horace	7 Jafendar Boulevard
7265204	Carmela Perez	23 Dan Street
4559672	Delilah Soraya	28 Jafendar Boulevard

Shipment_ID	Inventory_Item	Customer_ID
9173327437	908274	7265204
9174562342	553439	7265204
8484526471	546584	3543766
7023847265	908274	3543766
9174596725	553439	3543766

The following image shows the mapping in this example:



The mapping contains the following objects:

Read_input

The source that contains the path to the file with hierarchical data. Reads billing data from an XML file.

Shipping_Transform

A Hierarchical to Relational transformation that transforms XML input into relational output.

Write_Output2

A target that stores part of the transformed data, the Customer table, in relational format.

Write_Output3

A second target that stores another part of the transformed data, the Shipment table, in relational format.

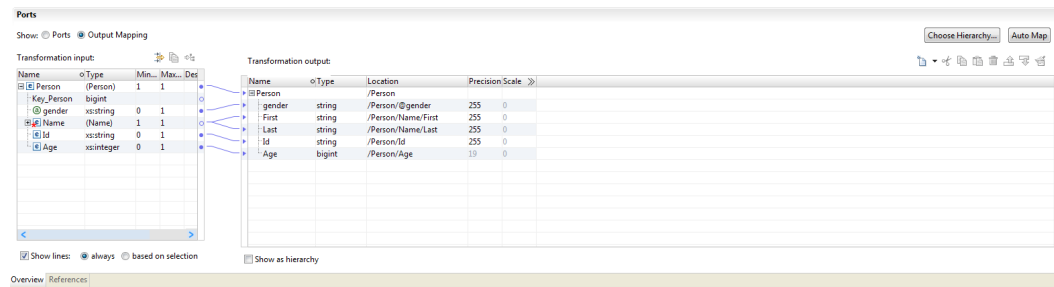
The mapping uses the **Read_input** flat file to input the target path for the hierarchical input. The mapping processes and transforms the data with the **Shipping_Transform** transformation. Then the mapping stores the output in the two output targets.

Output Relational Ports and the Overview View

To convert hierarchical data to relational output in the Hierarchical to Relational transformation, the wizard generates links between hierarchical nodes to relational ports. Use the **Overview** view to view the links between relational ports to hierarchical ports. You can also create groups of output ports by linking nodes from the hierarchical output to groups of ports.

To view the mapping for relational groups, use the **Overview** view. Select **Output Mapping**. The **Ports** panel appears in the **Overview** view.

The following image shows the **Ports** panel:



The **Transformation input** area, which shows the hierarchical schema, is to the left. The **Transformation output** area, which shows the relational output ports, is to the right.

You can define relational output ports in the **Transformation output** area and link nodes from the schema to the ports. You can also drag the pointer from a node in the schema to an empty field in the **Transformation output** area to create a port. When you drag a node from the output schema to a port, the Developer tool shows a link between them.

Hierarchical to Relational Transformation Ports

The Hierarchical to Relational transformation ports are defined on the transformation **Overview** view.

A Hierarchical to Relational transformation can read input from a file or buffer. The output ports return relational data from the transformation.

When you create an Hierarchical to Relational transformation, the Developer tool creates a default input port. The input type determines the type of data that the Data Integration Service passes to the Hierarchical to Relational transformation. The input type determines whether the input is data or a source file path.

Configure one of the following input types:

Buffer

The Hierarchical to Relational transformation receives rows of source data in the Input port. Use the buffer input type when you configure the transformation to receive data from an Informatica transformation.

File

The Hierarchical to Relational transformation receives the source file path in the Input port. The Hierarchical to Relational transformation opens the source file. You can also use the File input type for large files that might require a lot of system memory to process with a buffer input port.

When you create the transformation with the New Transformation wizard you can define an example input file. An example input file is a small sample of the input file. Reference an example input file when you create an Hierarchical to Relational. You also use the example input file when you test the transformation in the **Data Viewer** view.

The transformation contains one or more groups of ports that return relational data.

Schema References

A Hierarchical to Relational transformation requires a hierarchical schema to define the input hierarchy in the transformation. To use the schema in the transformation, you define a schema reference.

You can define transformation schema references in the transformation **References** view.

The Hierarchical to Relational transformation references schema objects in the Model repository. The schema objects can exist in the repository before you create the transformation. You can also import schemas from the transformation **References** view.

A schema can reference additional schemas. The **References** view shows the namespace and prefix for each schema that the Hierarchical to Relational transformation references. When you reference multiple schemas with empty namespaces the transformation is not valid.

Port Configuration

On the **Ports** panel, the transformation shows the mapping between the hierarchical schema nodes to the relational ports. The transformation uses a schema to define the hierarchical input. If the schema has more than one element that can be a root element, choose a node to be the root element.

The wizard generates links between hierarchical schema nodes and relational ports. If you want to change the generated links, you can use the **Ports** panel to add, delete, or edit links. You can link nodes to ports and create a port.

When you link nodes to the **Transformation output** area, the Developer tool updates the location field with the location of the node in the hierarchy. If you manually create ports, you must map a node to the port. Update the **Location** column and select a node from the list.

When you link a multiple-occurring node to a group that contains the parent element, you can configure the number of child element occurrences to include. Or, you can replace the parent group with the multiple-occurring child group in the transformation output.

To create a group, link a node to an empty column in the **Transformation output** area. If you link a multiple-occurring child node to an empty input or output column, the Developer tool asks you to relate the group to other output groups. When you select a group, the Developer tool creates keys to relate the groups.

Configure related groups of output ports in the **Transformation output** area. When the Developer tool prompts you to relate output groups, it adds the keys to the groups. You can also manually add ports to represent keys.

Hierarchical to Relational Transformation Development

Use the New Transformation wizard to auto-generate a Hierarchical to Relational transformation. Choose a schema or hierarchical sample file to define the input hierarchy.

1. Create the transformation in the Developer tool.
2. Configure the input port and mapping.

3. Test the transformation.

Creating the Hierarchical to Relational Transformation

1. In the Developer tool, click **File > New > Transformation**.
2. Select the Hierarchical to Relational transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model repository location to put the transformation and click **Next**.
4. To select a schema, select one of the following methods:
 - To use a schema from the Model repository to define the input hierarchy, near the **Schema Object** field, browse to select the schema file from the repository.
 - To import a schema file, click **Create a new schema object**. In the **New schema object** window, you can browse to and select a schema file, or you can select to create a schema from a sample hierarchical file.
5. Choose the root for the output hierarchy. In the **Hierarchy root** dialog box, select the element in the schema that is the root element for the output hierarchical file. To help select the root object, you can add a sample hierarchical file. To add a sample file, near the **Sample File** field, browse for and select the file from the file system.
6. Click **Finish**.

The wizard creates the transformation in the repository.

Configuring the Ports and Mapping

Configure the input and output ports in the **Overview** view.

1. Select the input port datatype, port type, precision and scale.
2. To view the mapping, in the **Overview** view **Ports** area, select **Output Mapping**.
3. Expand the trees in the **Ports** grid. To the left, the **Transformation input** panel shows the expected hierarchical input, and to the right, the **Transformation output** panel shows the relational output.
4. To define a node as a root, click **Choose Hierarchy**.

The Developer tool displays only the nodes from the root level and below the root level in the **Transformation input** area.
5. To view lines that connect the ports with the hierarchical nodes, click **Show Lines**. Select to view all the connection lines or just the lines for selected ports.
6. To add an input group or port to the **Transformation output** area, use one of the following methods:
 - Drag a simple or complex element in the **Transformation input** area to an empty column in the **Transformation output** area. If the node is a group node, the Developer tool adds a relational group without ports.
 - To add a relational group, select a row and right-click to select **New > Group**.
 - To add a relational port, right-click to select **New > Field**.
7. To clear the hierarchical node settings for locations of ports, use one of the following methods:
 - Select one or more nodes in the **Transformation input** area, right-click and select **Clear**.
 - Select one or more lines that connect the relational ports to the hierarchical nodes, right-click and select **Delete**.

8. To display the output ports in a hierarchy, click **Show As Hierarchy**. Each child group appears underneath the parent group.

Testing the Transformation

Test the Hierarchical to Relational transformation in the **Data Viewer** view.

Before you test the transformation, verify that you defined the file input location. You define the input location on the Data Integration Service machine in the Input Location column of the **Ports** panel in the **Overview** view.

1. Open the **Data Viewer** view.
2. Click **Run**.

The Developer tool validates the transformation. If there is no error, the Developer tool shows the hierarchical file content in the **Output** panel.

CHAPTER 19

Java Transformation

This chapter includes the following topics:

- [Java Transformation Overview, 296](#)
- [Designing a Java Transformation, 300](#)
- [Java Transformation Ports, 300](#)
- [Java Transformation Advanced Properties, 301](#)
- [Developing Java Code, 304](#)
- [Java Transformation Java Properties, 308](#)
- [Filter Optimization with the Java Transformation, 310](#)
- [Creating a Java Transformation, 313](#)
- [Compiling a Java Transformation, 314](#)
- [Troubleshooting a Java Transformation, 315](#)
- [Converting to Struct Data Example, 316](#)
- [Java Transformation in a Non-native Environment, 319](#)

Java Transformation Overview

Use the Java transformation to extend Developer tool functionality.

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 define simple or moderately complex transformation functionality without advanced knowledge of the Java programming language or an external Java development environment. The Java transformation is an active or passive transformation.

The Developer tool uses the Java Development Kit (JDK) to compile the Java code and generate byte code for the transformation. The Developer tool stores the byte code in the Model repository.

The Data Integration Service uses the Java Runtime Environment (JRE) to run generated byte code at run time. When the Data Integration Service runs a mapping with a Java transformation, the Data Integration Service uses the JRE to run the byte code and process input rows and generate output rows.

Create Java transformations by writing Java code snippets that define transformation logic. Define transformation behavior for a Java transformation based on the following events:

- The transformation receives an input row.
- The transformation processed all input rows.

In mappings that run on the Spark engine, you can use complex data types in Java transformations to process hierarchical data. With complex data types, the Spark engine directly reads, processes, and writes hierarchical data in Avro, Parquet, and JSON complex files.

Reusable and Non-Reusable Java Transformations

You can create a reusable or non-reusable Java transformation.

Reusable transformations can exist in multiple mappings. Non-reusable transformations exist within a single mapping.

The views in the editor where you define properties and create Java code differ based on whether you are creating a reusable or non-reusable Java transformation.

Active and Passive Java Transformations

A Java transformation generates output rows differently based on whether the transformation is active or passive.

After you create the transformation, you cannot change whether the transformation is active or passive.

Active Java Transformation

An active transformation can change the number of rows that pass through it.

To define the number of rows in the output, call the `generateRow()` method in the code to generate each output row. You might choose to generate multiple output rows from a single input row or generate a single output row from multiple input rows. For example, if the transformation contains two input ports that represent a start date and an end date, you can call the `generateRow()` method to generate an output row for each date between the start date and the end date.

Passive Java Transformation

A passive transformation cannot change the number of rows that pass through the transformation. The transformation calls the `generateRow()` method to generate an output row after processing each input row.

Data Type Conversion

A Java transformation converts Developer tool data types to Java data types, based on the Java transformation port type.

When a Java transformation reads input rows, it converts input port data types to Java data types.

When a Java transformation writes output rows, it converts Java data types to output port data types.

For example, the following processing occurs for an input port with the integer data type in a Java transformation:

1. The Java transformation converts the integer data type of the input port to the Java primitive `int` data type.
2. In the transformation, the transformation treats the value of the input port as the Java primitive `int` data type.
3. When the transformation generates the output row, it converts the Java primitive `int` data type to the integer data type.

The following table shows how the Java transformation maps Developer tool data types to Java primitive and complex data types:

Developer Tool Data Type	Java Data Type
array*	java.util.List
bigint	long
binary	byte[]
date/time	With nanoseconds processing enabled, BigDecimal with nanosecond precision With nanoseconds processing disabled, long with millisecond precision (the number of milliseconds since January 1, 1970 00:00:00.000 GMT)
decimal	With high precision processing disabled, double with precision 15 With high precision processing enabled, BigDecimal
double	double
integer	int
map*	java.util.Map
string	String
struct*	Customized JavaBean class with getters and setters for the struct field elements
text	String
* Supported only on the Spark engine.	

In Java, the java.util.List, java.util.Map, String, byte[], and BigDecimal data types are complex data types. The double, int, and long data types are primitive data types.

In the Developer tool, array, struct, and map data types are complex data types.

The Java transformation sets null values in primitive data types to zero. You can use the isNull and the setNull API methods on the **On Input** tab to set null values in the input port to null values in the output port. For an example, see [“setNull” on page 330](#).

Note: The decimal data type maps to BigDecimal when high precision is enabled. BigDecimal cannot be used with some operators, such as the + operator. If the Java code contains an expression that uses a decimal port or a complex port with an element of a decimal data type and the port is used with one of the operators, the Java code fails to compile.

Complex Data Type Conversion on the Spark Engine

You can add complex ports to the Java transformation in mappings that run on the Spark engine. A complex port is a port that is assigned a complex data type. The Java transformation converts complex data types to comparable Java complex data types. It also converts elements of complex data types to comparable Java data types, which are a boxed version of primitive data types.

Array Data Type

When a Java transformation reads input rows, it converts array data type to Java List data type. The transformation converts the data type of array elements to a boxed version of Java data types.

For example, the transformation converts the Developer tool complex data type `array<integer>` to a Java data type `List<Integer>`.

When a Java transformation writes output rows, it converts Java List data type to array data type. The transformation converts the data type of List elements to an unboxed version of the Developer tool data types.

Struct Data Type

When a Java transformation reads input rows of struct data type, it generates an equivalent Java bean class. The name of the Java bean class is the same as the name of the struct data type. The transformation converts the data type of struct elements to a boxed version of Java data types.

If the struct data type uses special characters or Java reserved keywords, such as `final` or `private`, the Java transformation replaces the special characters with an underscore (`_`) in the class name. You can open the full code in the Java view of the transformation properties tab to see the generated classes.

The Java transformation generates class member field names prefixed with an underscore (`_`). It also generates getters and setters for the member fields.

The generated Java bean class has the namespace of the type definition library name as an outer class. The name of the outer class is the same as the name of the type definition library. The Java data type name of the struct port is of the following format:

```
type_library_name.struct_type_name
```

For example, the type library name is `m_Type_Definition_Library`. The complex data type definition for the struct port is:

```
Customer {
  name string
  age integer
}
```

The Java transformation generates the Java bean classes with getters and setters for the member fields. The following code snippet shows the outer class and inner class:

```
public static final class m_Type_Definition_Library {
  public static final class Customer implements Serializable {
    private String _name;
    private Integer _age;

    public Customer() {}
```

The following code snippet shows the getter and setter for the struct element name of type string:

```
public String get_name() {
  return _name;
}

public void set_name(String _name) {
  this._name = _name;
}
```

When a Java transformation writes output rows of struct data type, it converts Java bean class to struct data type. The transformation converts the member fields of the class to struct elements. The data type of member fields are converted to an unboxed version of the Developer tool data types.

Map Data Type

When a Java transformation reads input rows, it converts map data type to Java Map data type. The transformation converts the data type of map elements to a boxed version of Java data types.

For example, the Developer tool complex data type `map<string, bigint>` is converted to a Java data type `Map<String, long>`.

When a Java transformation writes output rows, it converts Java Map data type to map data type. The transformation converts the data type of Map elements to an unboxed version of the Developer tool data types.

Designing a Java Transformation

When you design a Java transformation, you must consider factors, such as the type of transformation that you want to create.

When you design a Java transformation, consider the following questions:

- Do you need to create an active or passive Java transformation?

A passive Java transformation generates one output row for each input row in the transformation.

An active Java transformation generates multiple output rows for each input row in the transformation.

- Do you need to define any functions in the Java transformation? If so, which expressions do you want to include in each function?

For example, you can define a function that invokes an expression to look up the values of input or output ports or to look up the values of Java transformation variables.

- Do you want to create a reusable or non-reusable Java transformation?

A reusable transformation can exist in multiple mappings.

A non-reusable transformation can exist within a single mapping.

Java Transformation Ports

A Java transformation can have input and output ports.

To create and edit ports for a non-reusable Java transformation, use the **Ports** tab in the editor. To create and edit ports for a reusable Java transformation, use the **Overview** view in the editor.

You can specify default values for ports. After you add ports to a transformation, you can use the port names as variables in Java code snippets.

Creating Ports

When you create a Java transformation, it includes one input group and one output group.

When you create a port, the Developer tool adds it below the currently selected row or group.

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 based on the datatype of the port.

Input and Output Ports

The Java transformation initializes the value of unconnected input ports or output ports that do not have an assigned value in the Java code snippets.

Java transformation initializes ports based on the following Java data types:

Primitive data type

If you define a default value for the port that is not equal to null, 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 data type

If you define a default value for the port, the transformation creates a new object, and initializes the object to the default value. Otherwise, the transformation initializes the port variable to null. For example, if you define a default value for a string port, the transformation creates a new String object, and initializes the String object to the default value.

Note: If you access an input port variable with a null value in the Java code, a `NullPointerException` occurs.

You can enable an input port as a partition key and a sort key, and you can assign a sort direction. The Data Integration Service partitions the data and sorts the data in each partition by the sort key and sort direction. The Partition Key and Sort Key are valid when the transformation scope is set to All Input.

Use the following properties for partitioning and sorting data:

Partition Key

Input port that determines the rows of data to group into the same partition.

You can enable one or more input rows as partition keys. The Data Integration Service uses the partition keys to repartition the data before the code runs. If you do not select an input row as a partition key, the data is processed using its default partitioning scheme.

Sort Key

Input port that determines the sort criteria within each partition.

Direction

Ascending or descending order. Default is ascending.

Java Transformation Advanced Properties

The Java transformation includes advanced properties for both the transformation code and the transformation.

When you use the transformation in a mapping, you can override the transformation properties.

You can define the following advanced properties for the Java transformation on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Partitionable

The transformation can be processed with multiple threads. Clear this option if you want the Data Integration Service to use one thread to process the transformation. The Data Integration Service can use multiple threads to process the remaining mapping pipeline stages.

Disable partitioning for a Java transformation when the Java code requires that the transformation be processed with one thread.

Enable high precision

Processes a decimal data type port with a precision less than or equal to 38 as a Java BigDecimal data type port.

Disable high precision to process a decimal data type port as a Java Double data type port.

The following table shows how a Java transformation treats a value in a decimal data type input port based on whether you have enabled or disabled the high precision option:

Example	High Precision Processing Enabled	High Precision Processing Disabled
A decimal type input port receives a value of 40012030304957666903.	The Java transformation leaves the value as is.	The Java transformation converts the value to the following value: $4.00120303049577 \times 10^{19}$

If the Java transformation contains a decimal port or a complex port with an element of a decimal data type, the transformation must use the same precision mode as the mapping. For example, if you enable high precision in the Java transformation, you must enable high precision in the mapping.

Use nanoseconds in Date/Time

Converts date/time datatype ports to Java BigDecimal datatype ports with nanosecond precision.

Disable nanosecond processing so that the generated Java code converts date/time datatype ports to Java Long datatype ports with millisecond precision.

Classpath

Sets the classpath for jar or class file directories that are associated with non-standard Java packages that you import on the **Imports** tab.

The jar or class file directories must be accessible on the Developer tool client machine for compilation of the Java code.

Based on the operating system, separate classpath entries as follows:

- On UNIX, use a colon to separate classpath entries.
- On Windows, use a semicolon to separate classpath entries.

For example, if you import the Java converter package on the **Imports** tab and you define the package in converter.jar, you must add the location of the converter.jar file to the classpath before you compile the Java code for the Java transformation.

Note: You do not need to set the classpath for built-in Java packages. For example, because java.io is a built-in Java package, you do not need to set the classpath for java.io.

Is Active

Indicates whether the transformation is an active transformation. An active transformation can change the number of rows that pass through it.

You cannot change this property after you create the transformation. If you need to change this property, create a new transformation.

Transformation Scope

Defines the method that the Data Integration Service uses to apply 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.
- **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.
- **All Input.** Applies the transformation logic to all incoming data. when you choose All Input, the Data Integration Service drops transaction boundaries. Choose All Input when the results of the procedure depend on all rows of data in the source.

Stateless

Maintain the row order of the input data to the transformation. Select this option if the Data Integration Service should not perform any optimization that can change the row order.

When the Data Integration Service performs optimizations, it might lose an order established earlier in the mapping. You can establish order in a mapping with a sorted flat file source, a sorted relational source, or a Sorter transformation. When you configure a transformation to maintain row order, the Data Integration Service considers this configuration when it performs optimizations for the mapping. The Data Integration Service performs optimizations for the transformation if it can maintain the order. The Data Integration Service does not perform optimizations for the transformation if the optimization would change the row order.

Configuring the Classpath for the Developer Tool Client

You can add jar files or class file directories to the Developer tool client classpath.

To set the classpath for the machine where the Developer tool client runs, complete one of the following tasks:

- Configure the CLASSPATH environment variable. Set the CLASSPATH environment variable on the Developer tool client machine. This applies to all java processes run on the machine.
- For a non-reusable Java transformation, configure the the classpath in the Java transformation advanced properties. This applies to mappings that include this Java transformation. The Developer tool client includes files within the classpath when it compiles the java code.

To add jar or class file directories to the classpath in a Java transformation, complete the following steps:

1. On the **Advanced** tab, click the down arrow icon in the **Value** column next to **Classpath**.
The **Edit Classpath** dialog box appears.
2. To add a classpath, complete the following steps:
 - a. Click **Add**.
The **Save As** window appears.
 - b. In the **Save As** window, navigate to the directory where the jar file is located.
 - c. Click **OK**.
The classpath appears in the **Edit Classpath** dialog box.
3. To remove a jar file or class file directory, select the jar or class file directory and click **Remove**.
The directory disappears from the list of directories.

Configuring the Classpath for the Data Integration Service

You can add jar or class file directories that are required at run time to the classpath on the Data Integration Service node.

Place the jar files that are required during runtime in the following directory on the Data Integration Service node:

```
$INFA_HOME/services/shared/jars
```

The jar files in this location are loaded dynamically. Any class files that are required by individual mappings at run time are found and loaded from this directory.

Note: The Java transformation adds any jar files in this directory to the mapping-level classpath.

Developing Java Code

Use the code entry tabs in the **Java** view to write and compile Java code that defines transformation behavior for specific transformation events.

You can develop code snippets on the code entry tabs in any order. You can view, but not edit, the full Java code on the **Full Code** tab.

After you develop code snippets, you can compile the code snippets or the full Java code and view the results of the compilation in the **Results** window in the **Compilation** properties in the **Java** view.

Each code entry tab contains components that you use to write, view, and compile Java code:

Code properties

Provides controls that let you view and enter Java code, including Java transformation API methods. The following table describes the controls that are available in **Code** properties:

Control	Description
Navigator	<p>Shows input ports, output ports, and callable Java transformation API methods.</p> <p>Click an item in the navigator to display a description of the item.</p> <p>Double-click an item to add it to the Java code window. Alternatively, you can drag an item from the navigator into the Java code window.</p> <p>The navigator is available on the following code entry tabs:</p> <ul style="list-style-type: none">- Helpers- On Input- At End
Java code window	<p>Enables you to view or enter Java code for the transformation. The Java code window displays Java code by using the basic Java syntax highlighting.</p> <p>Note: On the Full Code tab, you can view but not edit the full class code for the Java transformation.</p> <p>The Java code window is available on the following code entry tabs:</p> <ul style="list-style-type: none">- Imports- Helpers- On Input- At End- Functions- Optimizer Interfaces- Full Code
New Function command	<p>Opens the Define Function dialog box that you use to define functions that invoke Java expressions.</p> <p>The Function command is available on the Functions tab.</p>
Editing toolbar	<p>Enables you to click tool icons, such as cut, copy, and paste, to edit Java code.</p> <p>The editing toolbar is available on the following code entry tabs:</p> <ul style="list-style-type: none">- Imports- Helpers- On Input- At End- Functions- Optimizer Interfaces

Compilation properties

Provides controls to compile and debug Java code. The following table describes the controls in the **Compilation** properties:

Control	Description
Compile command	Compiles the Java code for the transformation.
Results window	Displays the compilation results for the Java transformation class and enables you to find the source of errors in the code. To find an error in the code, right-click an error message in the Results window and select to view the error in the snippet code or in the full code. You can also double-click an error message in the Results window to find its source.

Creating Java Code Snippets

To create Java code snippets to define transformation behavior, use the **Java Code** window on the code entry tabs.

1. Click the appropriate code entry tab.

The following table describes the tasks that you can complete on the code entry tabs in the **Java** view:

Tab	Description
Imports	Imports third-party, built-in, and custom Java packages for an active or passive Java transformation. After you import packages, you can use them on the other code entry tabs.
Helpers	Declares user-defined variables and methods for the Java transformation class in an active or passive Java transformation. After you declare variables and methods, you can use them in any other code entry tab except the Imports tab.
On Input	Defines how an active or passive Java transformation behaves when it receives an input row. The Java code that you define on this tab runs one time for each input row. On this tab, you can also access and use input and output port data, variables, and Java transformation API methods.
At End	Defines how an active or passive Java transformation behaves after it processes all input data. On this tab, you can also set output data for active transformations, and call Java transformation API methods.
Functions	Defines functions that invoke expressions in a Java transformation with the Java programming language. For example, you can define a function that invokes an expression that looks up the values of input or output ports or looks up the values of Java transformation variables. On the Functions tab, you can manually define functions or click New Function to invoke the Define Function dialog box, which enables you to easily define a function.

Tab	Description
Optimizer Interfaces	Defines early selection or push-into filter optimization. Select the optimization method in the navigator. Update the code snippets to enable the optimization. Define the input ports and associated output ports to push filter logic through.
Full Code	Read-only. On this tab, you can view and compile the full class code for the Java transformation.

- To access input or output column variables in the snippet, expand the **Input** or **Output** list in the navigator and double-click the name of the port.
- To call a Java transformation API in the snippet, expand the **Callable APIs** list in the navigator and double-click the name of the method. If necessary, configure the appropriate input values for the method.
- Write appropriate Java code, based on the code entry tab type.
View the full class code for the Java transformation in the **Java code** window on the **Full Code** tab.

Importing Java Packages

On the **Imports** tab, you can import Java packages for active or passive Java transformations.

You can import third-party, built-in, or custom Java packages. After you import Java packages, you can use the imported packages on the other code entry tabs.

Note: On the **Imports** tab, you cannot declare or use static variables, instance variables, or user methods.

In the Developer tool, when you export or import metadata that contains a Java transformation, the jar or class files that contain the third-party or custom packages required by the Java transformation are not included in the export or import.

If you import metadata that contains a Java transformation, you must copy the jar or class files that contain the required third-party or custom packages to the Developer tool client and Data Integration Service node.

For example, to import the Java I/O package, enter the following code on the **Imports** tab:

```
import java.io.*;
```

When you import non-standard Java packages, add the package or class to the classpath in the Java transformation.

Defining Helper Code

On the **Helpers** tab, you can declare user-defined variables and methods for the Java transformation class in active or passive Java transformations.

After you declare variables and methods on the **Helpers** tab, you can use the variables and methods on any code entry tab except the **Imports** tab.

On the **Helpers** tab, you can declare the following types of code, variables, and methods:

- Static code and static variables.

Within a static block, you can declare static variables and static code. All instances of a reusable Java transformation in a mapping share static code and variables. Static code runs 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;
```

Use this variable to store the error threshold for the transformation and access it from all instances of the Java transformation in a mapping.

Note: You must synchronize static variables in a reusable Java transformation.

- Instance variables.

Multiple instances of a reusable Java transformation in a mapping 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 static or instance methods.

Extends the functionality of the Java transformation. Java methods declared on the **Helpers** tab can use or modify output variables or locally declared instance variables. You cannot access input variables from Java methods on the **Helpers** tab.

For example, use the following code on the **Helpers** tab to declare a function that adds two integers:

```
private int myTXAdd (int num1,int num2)
{
    return num1+num2;
}
```

Java Transformation Java Properties

Use the code entry tabs in the **Java** view to write and compile Java code that defines transformation behavior for specific transformation events.

The following tabs are code entry tabs:

- **Imports**
- **Helpers**
- **On Input**
- **At End**
- **Functions**
- **Optimizer Interfaces**

View the full class code for the Java transformation on the **Full Code** tab.

Imports Tab

On the **Imports** tab, you can import third-party, built-in, or custom Java packages for active or passive Java transformations.

To import a Java package, enter the code to import the package in the **Java code** window in the **Code** properties on the **Imports** tab.

For example, you might enter the following code to import the java.io package:

```
import java.io.*;
```

To compile the code that imports Java packages, click **Compile** in the **Compilation** properties on the **Imports** tab. The results of the compilation appear in the **Results** window on the **Imports** tab.

After you import Java packages, you can use them on the other code entry tabs.

Helpers Tab

On the **Helpers** tab, you can declare user-defined variables and methods for the Java transformation class in an active or passive Java transformation.

To declare user-defined variables and methods, enter the code in the **Java code** window in the **Code** properties on the **Helpers** tab.

To compile the helper code for the Java transformation, click **Compile** in the **Compilation** properties on the **Helpers** tab. The results of the compilation appear in the **Results** window on the **Helpers** tab.

After you declare variables and methods, you can use them in any other code entry tab except the **Imports** tab.

On Input Tab

On the **On Input** tab, you define how an active or passive Java transformation behaves when it receives an input row. On this tab, you can also access and use input and output port data, variables, and Java transformation API methods.

The Java code that you define on this tab runs one time for each input row.

To define how a Java transformation behaves when it receives an input row, enter the code in the **Java code** window in the **Code** properties on the **On Input** tab.

From the navigator on the **On Input** tab, you can access and define the following variables and API methods:

- Input port and output port variables. 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 on the **On Input** 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 on the **Helpers** tab.

For example, an active Java transformation has two input ports, BASE_SALARY and BONUSSES, with an integer datatype, and a single output port, TOTAL_COMP, with an integer datatype. You create a user-defined method on the **Helpers** tab, myTXAdd, that adds two integers and returns the result. Use the following Java code in the **On Input** tab to assign the total values for the input ports to the output port and generate an output row:

```
TOTAL_COMP = myTXAdd (BASE_SALARY, BONUSSES);
generateRow();
```

When the Java transformation receives an input row, it adds the values of the BASE_SALARY and BONUSSES 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.

To compile the code for the Java transformation, click **Compile** in the **Compilation** properties on the **On Input** tab. The results of the compilation appear in the **Results** window on the **On Input** tab.

At End Tab

On the **At End** tab, you define how an active or passive Java transformation behaves after it processes all input data. On this tab, you can also set output data for active transformations, and call Java transformation API methods.

To define how a Java transformation behaves after it processes all input data, enter the code in the **Java code** window in the **Code** properties on the **At End** tab.

You can access and define the following variables and API methods on the **At End** tab:

- Output port variables. You can use the names of any output ports that you defined on the **Ports** tab as variables, or set output data for active Java transformations.
- Instance variables and user-defined methods. Use any instance variables or user-defined methods you declared on the **Helpers** tab.
- Java transformation API methods. Call API methods provided by the Java transformation.

For example, use the following Java code to write information to the log when the end of data is reached:

```
logInfo("Number of null rows for partition is: " + partCountNullRows);
```

To compile the code for the Java transformation, click **Compile** in the **Compilation** properties on the **At End** tab. The results of the compilation appear in the **Results** window on the **At End** tab.

Functions Tab

On the **Functions** tab, you define functions that invoke expressions in a Java transformation with the Java programming language.

For example, you can define a function that invokes an expression that looks up the values of input or output ports or looks up the values of Java transformation variables.

To define a function, you can manually define functions in the **Java code** window in the **Code** properties on the **Functions** tab, or you can click **New Function** to invoke the **Define Function** dialog box, which enables you to easily define a function.

To compile the code, click **Compile** in the **Compilation** properties on the **Functions** tab. The results of the compilation appear in the **Results** window on the **Functions** tab.

Full Code Tab

On the **Full Code** tab, you can view, but not edit, the full class code for the Java transformation and compile the code.

You can view the full class code in the **Java code** window in the **Code** properties.

To compile the full code for the Java transformation, click **Compile** in the **Compilation** properties on the **Full Code** tab. The results of the compilation appear in the **Results** window on the **Full Code** tab.

Filter Optimization with the Java Transformation

The Data Integration Service can apply filter optimization to active Java transformations. When you define the Java transformation, you add code for filter optimization on the Java transformation **Optimizer Interfaces** tab.

Early Selection Optimization with the Java Transformation

You can enable an active or passive Java transformation for early selection optimization if the Java transformation has no side effects. The optimizer passes the filter logic through the Java transformation and modifies the filter condition as required.

To view the code snippets for early selection optimization, choose `PredicatePushOptimization` in the navigator of the **Optimizer Interfaces** tab.

`allowPredicatePush`

Boolean. Enables early selection. Change the function to return a true result and message in order to enable early selection. Default is false, and the function returns a message that optimization is not supported.

```
public ResultAndMessage allowPredicatePush(boolean ignoreOrderOfOp) {
    // To Enable PredicatePushOptimization, this function should return true
    //return new ResultAndMessage(true, "");
    return new ResultAndMessage(false, "Predicate Push Optimization Is Not
Supported");
}
```

`canGenerateOutputFieldEvalError`

Boolean. Indicates whether or not the Java transformation can return an output field error, such as a division by zero error. Change the function to return false if the Java transformation does not generate output field errors. When the Java transformation can generate field errors, then the Data Integration Service cannot use early selection optimization.

```
public boolean canGenerateOutputFieldEvalError() {
    // If this Java transformation can never generate an output field evaluation error,
    // return false.
    return true;
}
```

`getInputExpr`

Returns an Informatica expression that describes which input values from input fields comprise an output field. The optimizer needs to know which input fields comprise an output field in order to push the filter logic through the transformation.

```
public InfaExpression getInputExpr(TransformationField field,
TransformationDataInterface group) {
    // This should return an Informatica expression for output fields in terms of input
fields
    // We will only push predicate that use fields for which input expressions are
defined.
    // For example, if you have two input fields in1 and in2 and three output fields
out0, out1, out2
    // out0 is the pass-through of in1, out2 is sum of in1 and in2, and out3 is unknown,
the code should be:
    //if (field.getName().equals("out0"))
    //    return new InfaExpression("in0", instance);
    //else if (field.getName().equals("out1"))
    //    return new InfaExpression("in0 + in1", instance);
    //else if (field.getName().equals("out2"))
    //    return null;
    return null;
}
```

For example, a mapping contains a filter expression, `"out0 > 8"`. Out0 is the value of the out0 output port in the Java transformation. You can define the value of out0 as the value of the in0 input port + 5. The optimizer can push the following expression `"(in0 + 5) > 8"` past the Java transformation with early selection optimization. You can return NULL if an output field does not have input field expression. The optimizer does not push filter expressions past output fields with no input expression.

You might include the following code:

```
if (field.getName().equals("out0"))
    return new InfaExpression("in0 + 5", instance);
else if (field.getName().equals("out2"))
    return null;
```

inputGroupsPushPredicateTo

Returns a list of groups that can receive the filter logic. The Java transformation has one input group. Do not modify this function for the Java transformation.

```
public List<TransformationDataInterface> inputGroupsPushPredicateTo(
    List<TransformationField> fields) {
    // This functions returns a list of input data interfaces to push predicates to.
    // Since JavaTx only has one input data interface, you should not have to modify
    this function
    AbstractTransformation tx = instance.getTransformation();
    List<DataInterface> dis = tx.getDataInterfaces();
    List<TransformationDataInterface> inputDIs = new
    ArrayList<TransformationDataInterface>();
    for (DataInterface di : dis){
        TransformationDataInterface tdi = (TransformationDataInterface) di;
        if (tdi.isInput())
            inputDIs.add(tdi);
    }
    if(inputDIs.size() == 1)
        return inputDIs;
    else
        return null;
}
```

Push-Into Optimization with the Java Transformation

You can enable an active Java transformation for push-into optimization if it has no side effects and the optimization does not affect the mapping results.

When you configure push-into optimization for the Java transformation, you define a way for the Java transformation to store the filter condition that it receives from the optimizer. Add code that examines the filter condition. If the Java transformation can absorb the filter logic, then the Java transformation passes a true condition back to the optimizer. The optimizer removes the Filter transformation from the optimized mapping.

When you configure the Java transformation you write the code that stores the filter condition as transformation metadata during optimization. You also write the code to retrieve the filter condition at run-time and to drop the rows according to the filter logic.

When you define the Java transformation, you add code for push-into optimization on the Java transformation **Optimizer Interfaces** tab. To access the code snippets for push-into optimization, choose FilterPushdownOptimization in the navigator of the transformation **Optimizer Interfaces** tab.

The Developer tool displays code snippets to enable push-into optimization and to receive the filter condition from the optimizer. Update the code snippets to enable optimization and to save the filter logic as transformation metadata.

isFilterSupported

Returns true to enable push-into optimization. Returns false to disable push-into optimization. Change the function to return true in order to enable push-into optimization.

```
public ResultAndMessage isFilterSupported() {
    // To enable filter push-into optimization this function should return true
    // return new ResultAndMessage(true, "");
    return new ResultAndMessage(false, "Filter push-into optimization is not supported");
}
```


pushFilter

Receives the filter condition from the optimizer.

Add code to examine the filter and determine if the filter logic can be used in the transformation. If the transformation can absorb the filter, then use the following method to store the filter condition as transformation metadata:

```
storeMetadata(String key, String data)
```

The key is an identifier for the metadata. You can define any string as a key. The data is the data you want to store in order to determine which rows to drop at run time. For example, the data might be the filter condition that the Java transformation receives from the optimizer.

```
public ResultAndMessage pushFilter(InfaExpression condition) {  
    // Add code to absorb the filter  
    // If filter is successfully absorbed return new ResultAndMessage(true, ""); and the  
optimizer  
    // will remove the filter from the mapping  
    // If the filter is not absorbed, return new ResultAndMessage(false, msg);  
    return new ResultAndMessage(false, "Filter push-into optimization is not supported");  
}
```

Creating a Java Transformation

In the Developer tool, you can create a reusable or non-reusable Java transformation.

Creating a Reusable Java Transformation

Reusable transformations can exist within multiple mappings.

Create a reusable Java transformation in the Developer tool.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
The **New** dialog box appears.
3. Select the Java transformation.
4. Click **Next**.
5. Enter a name for the transformation.
6. To create an active transformation, select the **Create as active** option.
7. Click **Finish**.

The transformation appears in the editor.

8. In the **Ports** view, click the **New** button to add a port to the transformation.
9. Edit the port to set the name, datatype, and precision.
Use port names as variables in Java code snippets.
10. In the **Java** view, use the code entry tabs to write and compile the Java code for the transformation.
11. In the **Java** view, use the **Functions** tab to define functions that invoke expressions.
12. On any code entry tab, double-click error messages in the **Results** window in the **Compilation** properties to locate and fix compilation errors in the Java code for the transformation.

13. In the **Advanced** view, edit the transformation properties.

Creating a Non-Reusable Java Transformation

Non-reusable transformations exist within a single mapping.

Create a non-reusable Java transformation in the Developer tool.

1. In a mapping or maplet, drag a Java transformation from the Transformation palette to the editor.
2. In the **New Java Transformation** dialog box, enter a name for the transformation.
3. To create an active transformation, select the **Create as active** option.
4. Click **Finish**.

The transformation appears in the editor.

5. On the **General** tab, edit the transformation name and the description.
6. On the **Ports** tab, click the **New** button to add a port to the transformation.
7. Edit the port to set the name, datatype, and precision.
Use port names as variables in Java code snippets.
8. In the **Java** view, use the code entry tabs to write and compile the Java code for the transformation.
9. In the **Java** view, use the **Functions** tab to define functions that invoke expressions.
10. On any code entry tab, double-click error messages in the **Results** window in the **Compilation** properties to locate and fix compilation errors in the Java code for the transformation.
11. In the **Advanced** view, edit the transformation properties.

Compiling a Java Transformation

The Developer tool 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 **Results** window in the **Compilation** properties on the code entry tabs. The Java compiler installs with the Developer tool in the `java/bin` directory.

To compile the full code for the Java transformation, click **Compile** in the **Compilation** properties on the **Full 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 on the code entry tabs.

When you compile a Java transformation, the Developer tool 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 Developer tool 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 **Results** window. Use the results of the compilation to identify and locate Java code errors.

Troubleshooting a Java Transformation

In the **Results** window in the **Compilation** properties on any code entry tab, you can find and fix Java code errors.

Errors in a Java transformation can occur due to an error in code on a code entry tab or in the full code for the Java transformation class.

To troubleshoot a Java transformation, complete the following high-level steps:

1. Find the source of the error in the Java snippet code or in the full class code for the transformation.
2. Identify the type of error. Use the results of the compilation in the **Results** window and the location of the error to identify the type of error.
3. Fix the Java code on the code entry tab.
4. Compile the transformation again.

Finding the Source of Compilation Errors

To find the source of compilation errors, use the results of the compilation displayed in the **Results** window in the **Compilation** properties on a code entry tab or the **Full Code** tab.

When you double-click an error message in the **Results** window, the source code that caused the error is highlighted in the **Java code** window on the code entry tab or on the **Full Code** tab.

You can find errors on the **Full Code** tab, but you cannot edit Java code on the **Full Code** tab. To fix errors that you find on the **Full Code** tab, modify the code on the appropriate code entry tab. You might need to use the **Full Code** tab to view errors caused by adding user code to the full class code for the transformation.

Finding an Error on a Code Entry Tab or the Full Code Tab

You can find compilation errors on a code entry tab or the **Full Code** tab.

1. In the **Results** window in the **Compilation** properties on any code entry tab or the **Full Code** tab, right-click an error message.
2. Click either **Show In > Snippet** or **Show In > Full Code Tab**.

The Developer tool highlights the source of the error on the selected tab.

Note: You can view but you cannot correct errors on the **Full Code** tab. To correct errors, you must navigate to the appropriate code entry tab.

Identifying the Source of Compilation Errors

Compilation errors can appear as a result of errors in the user code.

Errors in the user code might also generate an error in the non-user code for the class. Compilation errors occur in user and non-user code for the Java transformation.

User Code Errors

Errors can occur in the user code on the code entry tabs. User code errors include standard Java syntax and language errors.

User code errors might also occur when the Developer tool 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 on the **On Input** tab, the Java compiler issues an error for a redeclaration of a variable. To fix the error, rename the variable on the **On Input** tab.

Non-User Code Errors

User code on the code entry tabs can 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** 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 Developer tool adds the code from the **On Input** 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 end prematurely, and the Java compiler issues an error.

Converting to Struct Data Example

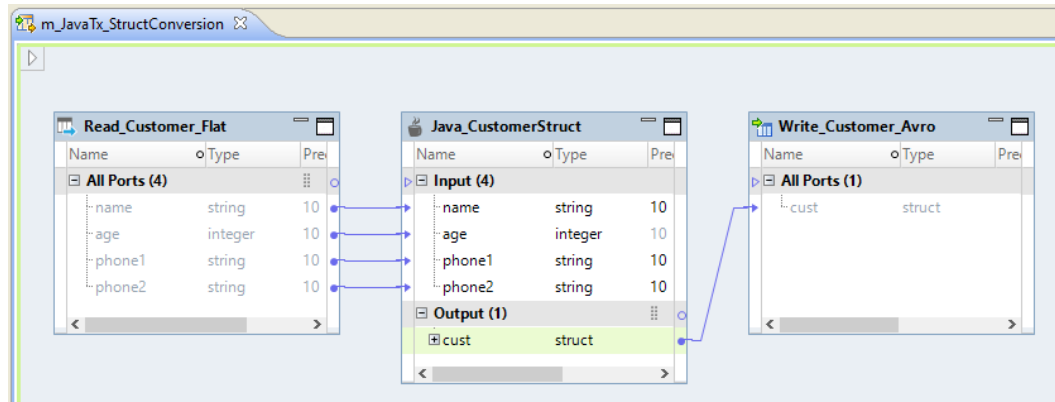
Your organization needs to convert a large volume of customer data in a flat file to struct data and write it to an Avro file. The input file contains customer details such as name, age, and phone numbers. If the customer name is null in the input file, you do not want to add customer details to the output file.

You can develop a mapping with a Java transformation to define the transformation functionality. In the Hadoop environment, run the mapping on the Spark engine to transform the data and write the struct data to an Avro file.

Create a mapping and configure the following transformations:

- Read transformation that reads customer information from a flat file source
- Java transformation as an active transformation that converts flat data to struct data and removes inconsistent data
- Write transformation that writes the struct data to an Avro file

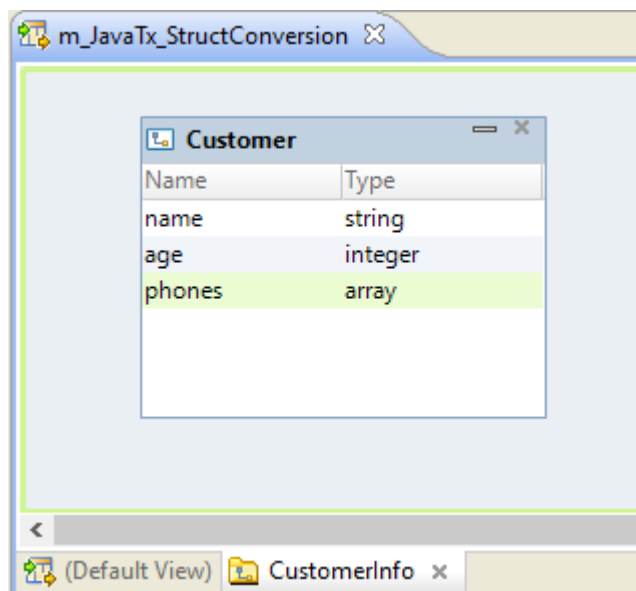
The following image shows the mapping with a Read transformation, a Java transformation, and a Write transformation.



On the type definition library tab of the mapping editor, create a complex data type definition Customer. The complex data type definition represents the schema of the struct data. Rename the type definition library to CustomerInfo. Add the following elements to the complex data type definition:

- name of type string
- age of type integer
- phones of type array with string elements

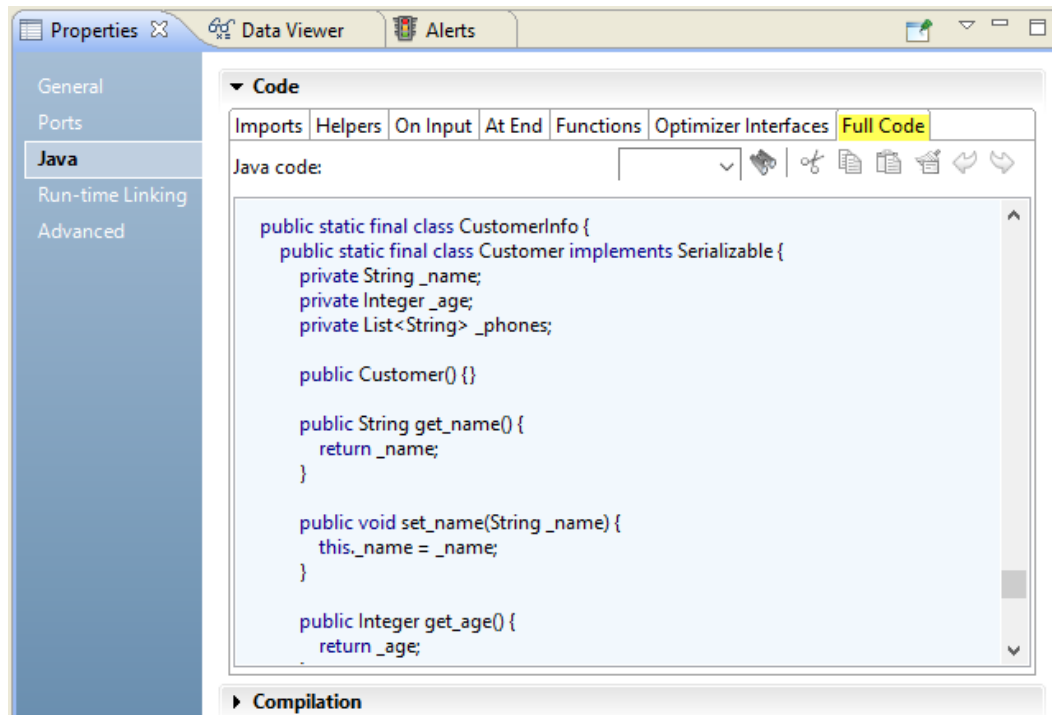
The following image shows the complex data type definition in the type definition library:



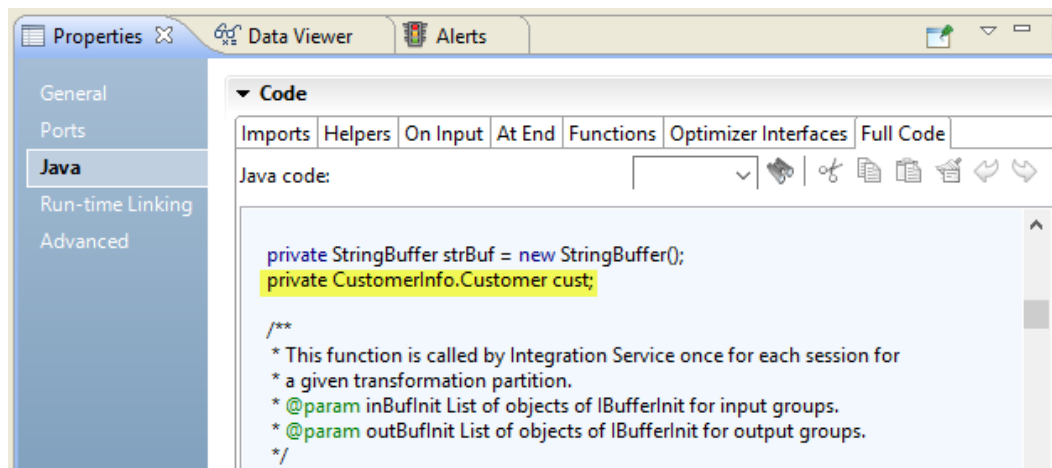
In the Java transformation, add a struct output port and specify the type configuration of the port to reference the complex data type definition that you created. The Java transformation generates a class Customer with setters and getters to read and set the member fields. The class contains the following member fields:

- _name
- _age
- _phones

The following image shows the class created for the struct port in the **Full Code** tab of the **Java** view:

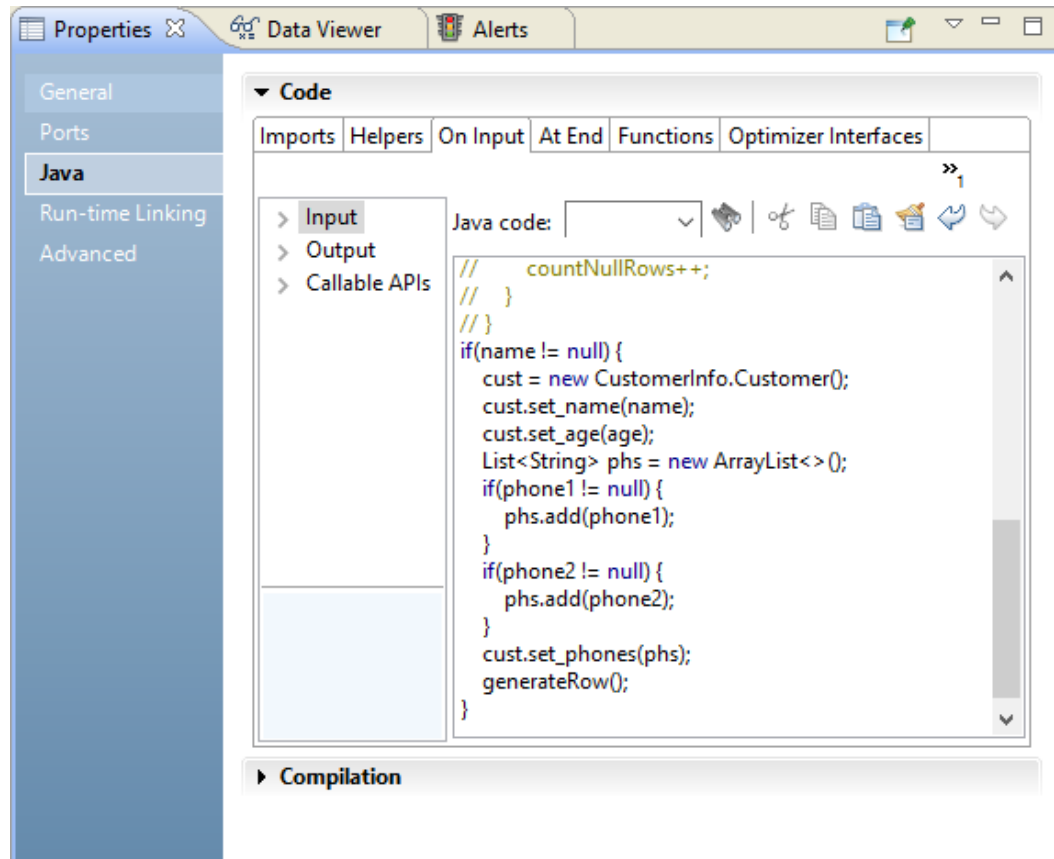


The Java data type for the struct port uses the name of the type definition library and complex data type definition. The following image shows the Java data type name `CustomerInfo.Customer` for the `cust` field in the generated code:



In the **Java** view of the Java transformation, import any third-party, built-in, or custom Java packages that the transformation requires. Write and compile the Java code to convert the flat data into struct data and to remove the customer row if the customer name is null.

The following image shows the code in the **On Input** tab:



Validate the mapping and run the mapping on the Spark engine to write the transformed data to the Avro file output.

Java Transformation in a Non-native Environment

The Java transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Not supported.

Java Transformation on the Blaze Engine

To use external .jar files in a Java transformation, perform the following steps:

1. Copy external .jar files to the Informatica installation directory in the Data Integration Service machine at the following location: <Informatica installation directory>/services/shared/jars. Then recycle the Data Integration Service.

2. On the machine that hosts the Developer tool where you develop and run the mapping that contains the Java transformation:
 - a. Copy external .jar files to a directory on the local machine.
 - b. Edit the Java transformation to include an import statement pointing to the local .jar files.
 - c. Update the classpath in the Java transformation.
 - d. Compile the transformation.

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Partitioning

- The following restrictions apply to the Transformation Scope property:
 - The value Transaction for transformation scope is not valid.
 - If you enable an input port for partition key, the transformation scope must be set to All Input.
 - Stateless must be enabled if the transformation scope is row.

Java Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

General Restrictions

The Java transformation is supported with the following restrictions on the Spark engine:

- The Java code in the transformation cannot write output to standard output when you push transformation logic to Hadoop. The Java code can write output to standard error which appears in the log files.
- For date/time values, the Spark engine supports the precision of up to microseconds. If a date/time value contains nanoseconds, the trailing digits are truncated.

Partitioning

The Java transformation has the following restrictions when used with partitioning:

- The Partitionable property must be enabled in the Java transformation. The transformation cannot run in one partition.
- The following restrictions apply to the Transformation Scope property:
 - The value Transaction for transformation scope is not valid.
 - If you enable an input port for partition key, the transformation scope must be set to All Input.
 - Stateless must be enabled if the transformation scope is row.

Mapping Validation

Mapping validation fails in the following situations:

- You reference an unconnected Lookup transformation from an expression within a Java transformation.
- You select a port of a complex data type as the partition or sort key.
- You enable nanosecond processing in date/time and the Java transformation contains a port of complex data type with an element of a date/time type. For example, a port of type `array<data/time>` is not valid if you enable nanosecond processing in date/time.

The mapping fails in the following situations:

- The Java transformation and the mapping use different precision modes when the Java transformation contains a decimal port or a complex port with an element of a decimal data type.

Even if high precision is enabled in the mapping, the mapping processes data in low-precision mode in some situations, such as when the mapping contains a complex port with an element of a decimal data type, or the mapping is a streaming mapping. If high precision is enabled in both the Java transformation and the mapping, but the mapping processes data in low-precision mode, the mapping fails.

- Binary null characters are passed to an output port. To avoid a mapping failure, you can add code to the Java transformation that replaces the binary null characters with an alternative character before writing the data to the output ports.

Using External .jar Files

To use external .jar files in a Java transformation, perform the following steps:

1. Copy external .jar files to the Informatica installation directory in the Data Integration Service machine at the following location:
`<Informatica installation directory>/services/shared/jars`
2. Recycle the Data Integration Service.
3. On the machine that hosts the Developer tool where you develop and run the mapping that contains the Java transformation:
 - a. Copy external .jar files to a directory on the local machine.
 - b. Edit the Java transformation to include an import statement pointing to the local .jar files.
 - c. Update the classpath in the Java transformation.
 - d. Compile the transformation.

Java Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

The Data Integration Service ignores the following properties:

- Partitionable. The Data Integration Service ignores the property and can process the transformation with multiple threads.
- Stateless. The Data Integration Service ignores the property and maintains the row order of the input data.

CHAPTER 20

Java Transformation API Reference

This chapter includes the following topics:

- [Java Transformation API Methods Overview, 322](#)
- [defineJExpression, 323](#)
- [failSession, 324](#)
- [generateRow, 324](#)
- [getInRowType, 325](#)
- [getMetadata, 326](#)
- [incrementErrorCount, 326](#)
- [invokeJExpression, 327](#)
- [isNull, 328](#)
- [logError, 328](#)
- [logInfo, 329](#)
- [resetNotification, 329](#)
- [setNull, 330](#)
- [storeMetadata, 331](#)

Java Transformation API Methods Overview

On the code entry tabs in the **Java** view in the editor, you can add API methods to the Java code to define transformation behavior.

To add an API method to the code, expand the **Callable APIs** list in the navigator on the code entry tab, and then double-click the name of the method that you want to add to the code.

Alternatively, you can drag the method from the navigator into the Java code snippet or manually enter the API method in the Java code snippet.

You can add the following API methods to the Java code in a Java transformation:

defineJExpression

Defines a Java expression.

failSession

Throws an exception with an error message and fails the mapping.

generateRow

Generates an output row for active Java transformations.

getInRowType

Returns the input type of the current row in the transformation.

incrementErrorCount

Increments the error count for the mapping.

invokeJExpression

Invokes a Java expression that you have defined by using the `defineJExpression` method.

isNull

Checks for a null value in an input column.

logError

Writes an error message to the log.

logInfo

Writes an informational message to the log.

resetNotification

If the Data Integration Service machine runs in restart mode, resets variables that you use in the Java code after a mapping run.

setNull

Sets the value of an output column in an active or passive Java transformation to null.

defineJExpression

Defines an expression, including the expression string and input parameters. Arguments for the `defineJExpression` method include an array of `JExprParamMetadata` objects that contains the input parameters and a string value that defines the expression syntax.

Use the following syntax:

```
defineJExpression(
    String expression,
    Object[] paramMetadataArray
);
```

The following table describes the parameters:

Parameter	Type	Datatype	Description
expression	Input	String	String that represents the expression.
paramMetadataArray	Input	Object[]	Array of <code>JExprParaMetadata</code> objects that contain the input parameters for the expression.

You can add the `defineJExpression` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

To use the `defineJExpression` method, 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 a parameter to the `defineJExpression` method.

For example, the following Java code creates an expression to look up the value of two strings:

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
params[1] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
defineJExpression(":lkp.mylookup(x1,x2)",params);
```

Note: You must number the parameters that you pass to the expression consecutively and start the parameters with the letter `x`. For example, to pass three parameters to an expression, name the parameters `x1`, `x2`, and `x3`.

failSession

Throws an exception with an error message and fails the mapping.

Use the following syntax:

```
failSession(String errorMessage);
```

The following table describes the parameter:

Parameter	Parameter Type	Datatype	Description
<code>errorMessage</code>	Input	String	Error message string.

Use the `failSession` method to end the mapping. Do not use the `failSession` method in a try/catch block on a code entry tab.

You can add the `failSession` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

The following Java code shows how to test the `input1` input port for a null value and fail the mapping if it is null:

```
if(isNull("input1")) {
    failSession("Cannot process a null value for port input1.");
}
```

generateRow

Generates an output row for active Java transformations.

Use the following syntax:

```
generateRow();
```

When you call the `generateRow` method, 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 the `generateRow` method more than once for each input row. If you do not use the `generateRow` method in an active Java transformation, the transformation does not generate output rows.

You can add the `generateRow` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

You can call the `generateRow` method in active transformations only. If you call the `generateRow` method in a passive transformation, the Data Integration Service generates an error.

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.

Use the following syntax:

```
rowType getInRowType();
```

The following table describes the parameter:

Parameter	Parameter Type	Datatype	Description
rowType	Output	String	Returns the update strategy type, which is one of the following values: <ul style="list-style-type: none">- DELETE- INSERT- REJECT- UPDATE

You can add the `getInRowType` method to the Java code on the **On Input** code entry tab.

You can use the `getInRowType` method in active transformations configured to set the update strategy. If you call this method in an active transformation that is not configured to set the update strategy, the Data Integration Service generates an error.

getMetadata

Retrieves Java transformation metadata at runtime. The `getMetadata` method retrieves metadata that you save with the `storeMetadata` method, such as a filter condition that the optimizer passes to the Java transformation in the `pushFilter` function.

Use the following syntax:

```
getMetadata (String key);
```

The following table describes the parameters:

Parameter	Parameter Type	Datatype	Description
key	Input	String	Identifies the metadata. The <code>getMetadata</code> method uses the key to determine which metadata to retrieve.

You can add the `getMetadata` method to the Java code on the following code entry tabs:

- Helper
- On Input
- At End
- Optimizer Interfaces
- Functions

You can configure the `getMetadata` method to retrieve filter conditions for push-into optimization. The `getMetadata` method can retrieve each filter condition that you store from the optimizer.

```
// Retrieve a filter condition  
String mydata = getMetadata ("FilterKey");
```

incrementErrorCount

Increments the error count. If the error count reaches the error threshold, the mapping fails.

Use the following syntax:

```
incrementErrorCount(int nErrors);
```

The following table describes the parameter:

Parameter	Parameter Type	Datatype	Description
nErrors	Input	Integer	Number by which to increment the error count.

You can add the `incrementErrorCount` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

The following Java code shows how 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;
}
```

invokeJExpression

Invokes an expression and returns the value for the expression.

Use the following syntax:

```
(datatype)invokeJExpression(
    String expression,
    Object[] paramMetadataArray);
```

Input parameters for the invokeJExpression method are a string value that represents the expression and an array of objects that contain the expression input parameters.

The following table describes the parameters:

Parameter	Parameter Type	Datatype	Description
expression	Input	String	String that represents the expression.
paramMetadataArray	Input	Object[]	Array of objects that contain the input parameters for the expression.

You can add the invokeJExpression method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

Use the following rules and guidelines when you use the invokeJExpression method:

- Return datatype. The return datatype of the invokeJExpression method 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 the invokeJExpression method is INSERT.

To use a different row type for the return value, use the advanced interface.

- Null values. If you pass a null value as a parameter or the return value for the invokeJExpression method 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 the String datatype.

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 datatype.

The following example concatenates the strings "John" and "Smith" and returns the string, "John Smith":

```
(String)invokeJExpression("concat(x1,x2)", new Object [] { "John ", "Smith" });
```

Note: You must number the parameters that you pass to the expression consecutively, and start the parameter with the letter x. For example, to pass three parameters to an expression, name the parameters x1, x2, and x3.

isNull

Checks the value of an input column for a null value.

Use the following syntax:

```
Boolean isNull(String strColName);
```

The following table describes the parameter:

Parameters	Parameter Type	Datatype	Description
strColName	Input	String	Name of an input column.

You can add the isNull method to the Java code on the **On Input** code entry tab.

The following Java code shows how to check whether the value of the SALARY input column is null before adding it to the totalSalaries instance variable:

```
// if value of SALARY is not null
if (!isNull("SALARY")) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

Alternatively, use the following Java code to achieve the same results:

```
// if value of SALARY is not null
String strColName = "SALARY";
if (!isNull(strColName)) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

logError

Writes an error message to the log.

Use the following syntax:

```
logError(String msg);
```


The following table describes the parameter:

Parameter	Parameter Type	Datatype	Description
msg	Input	String	Error message string.

You can add the `logError` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

The following Java code shows how to log an error when the input port is null:

```
// check BASE_SALARY
if (isNull("BASE_SALARY")) {
    logError("Cannot process a null salary field.");
}
```

When the code runs, the following message appears in the log:

```
[JTX_1013] [ERROR] Cannot process a null salary field.
```

logInfo

Writes an informational message to the log.

Use the following syntax:

```
logInfo(String msg);
```

The following table describes the parameter:

Parameter	Parameter Type	Datatype	Description
msg	Input	String	Information message string.

You can add the `logInfo` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

The following Java code shows how to write a message to the log after the Java transformation processes a message threshold of 1000 rows:

```
if (numRowsProcessed == messageThreshold) {
    logInfo("Processed " + messageThreshold + " rows.");
}
```

resetNotification

If the Data Integration Service machine runs in restart mode, resets variables that you use in the Java code after a mapping run.

In restart mode, the Data Integration Service is not deinitialized but is reset after a request so that the Data Integration Service can process the next request.

For a Java transformation, use the `resetNotification` method to reset variables in the Java code after a mapping run.

Use the following syntax:

```
public int resetNotification(IGroup group) {  
    return EStatus.value;  
}
```

The following table describes the parameters:

Parameter	Parameter Type	Datatype	Description
int	Output	EStatus.value	Return value, where <i>value</i> is one of the following values: <ul style="list-style-type: none">- SUCCESS. Success.- FAILURE. Failure.- NOIMPL. Not implemented.
group	Input	IGroup	The input group.

You can add the `resetNotification` method to the Java code on the code entry tab on the **Helpers** tab.

The `resetNotification` method does not appear in the Callable APIs list.

For example, assuming that the Java code declares a static variable named `out5_static` and initializes it to 1, the following Java code resets the `out5_static` variable to 1 after the next mapping run:

```
public int resetNotification(IGroup group) {  
    out5_static=1;  
    return EStatus.SUCCESS;  
}
```

This method is not required. However, if the Data Integration Service runs in restart mode and the mapping contains a Java transformation that does not implement the `resetNotification` method, the `JSDK_42075` warning message appears in the log.

setNull

Sets the value of an output column to null in an active or passive Java transformation.

Use the following syntax:

```
setNull(String strColName);
```

The following table describes the parameter:

Parameter	Parameter Type	Datatype	Description
strColName	Input	String	Name of an output column.

The `setNull` method sets the value of an output column in an active or passive Java transformation to null. After you set an output column to null, you cannot modify the value until you generate an output row.

You can add the `setNull` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

The following Java code shows how 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");
}
```

Alternatively, you can use the following Java code achieve the same results:

```
// check value of Q3RESULTS input column
String strColName = "Q3RESULTS";
if(isNull(strColName)) {
    // set the value of output column to null
    setNull(strColName);
}
```

storeMetadata

Stores Java transformation metadata that you can retrieve at runtime with the `getMetadata` method.

Use the following syntax:

```
storeMetadata (String key String data);
```

The following table describes the parameters:

Parameter	Parameter Type	Datatype	Description
key	Input	String	Identifies the metadata. The <code>storeMetadata</code> method requires a key to identify the metadata. Define the key as any string.
data	Input	String	The data that you want to store as Java transformation metadata.

You can add the `storeMetadata` method to the Java code to the following code entry tabs:

- Helper
- On Input
- At End
- Optimizer Interfaces
- Functions

You can configure the `storeMetadata` method in an active transformation to accept filter conditions for push-into optimization. The `storeMetadata` method stores a filter condition that the optimizer pushes from the mapping to the Java transformation.

```
// Store a filter condition
storeMetadata ("FilterKey", condition);
```

CHAPTER 21

Java Expressions

This chapter includes the following topics:

- [Java Expressions Overview, 332](#)
- [Using the Define Function Dialog Box to Define an Expression, 333](#)
- [Working with the Simple Interface, 335](#)
- [Working with the Advanced Interface, 337](#)
- [JExpression Class API Reference, 341](#)

Java Expressions Overview

You can invoke 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 API methods to invoke the expression. You invoke the expression and use the result of the expression on 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.

The following table describes the methods that you can use to create and invoke expressions in a Java transformation:

Method	Description
Define Function dialog box	Enables you to create a function that invokes an expression and generate the code for an expression.
Simple interface	Enables you to call a single API method to invoke an expression and get the result of the expression.
Advanced interface	Enables you to define the expression, invoke the expression, and use the result of the expression. If you are familiar with object-oriented programming and want more control over invoking the expression, use the advanced interface.

Expression Function Types

You can create expressions for a Java transformation by using the **Define Function** 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 Function** dialog box, you can validate the expression before you use it in a Java transformation.

You can invoke the following types of expression functions in a Java transformation:

Expression Function Type	Description
Transformation language functions	SQL-like functions designed to handle common expressions.
User-defined functions	Functions that you create in the Developer tool based on transformation language functions.
Custom functions	Functions that you create with the Custom Function API.

You can also use unconnected transformations and built-in variables in expressions. For example, you can use an unconnected lookup transformation in an expression.

Using the Define Function Dialog Box to Define an Expression

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 Function** dialog box.

To create an expression function and use the expression in a Java transformation, complete the following high-level tasks:

1. Configure the function that invokes the expression, including the function name, description, and parameters. You use the function parameters when you create the expression.
2. Create the expression syntax and validate the expression.
3. Generate the Java code that invokes the expression.

The Developer places the code on the **Functions** code entry tab.

After you generate the Java code, call the generated function on the appropriate code entry tab to invoke an expression or get a JExpression object, based on whether you use the simple or advanced interface.

Note: To validate an expression when you create the expression, you must use the **Define Function** 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 the Date datatype to an expression, use the String datatype for the input parameter.

If an expression returns the Date datatype, you can use the return value as the String datatype in the simple interface and the String or long datatype in the advanced interface.

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 Function** dialog box.

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 Developer places the generated Java code on the **Functions** code entry tab. Use the generated Java code to call the functions that invoke the expression in the code entry tabs. 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 create the expression again.

Creating an Expression and Generating Java Code by Using the Define Function Dialog Box

You can create a function that invokes an expression in the **Define Function** dialog box.

Complete the following steps to create a function that invokes an expression:

1. In the Developer, open a Java transformation or create a new Java transformation.
2. On the **Java Code** tab, click **New Function**.
The **Define Function** dialog box appears.
3. Enter a function name.
4. Optionally, enter a description for the expression.
Enter up to 2,000 characters.
5. Create the arguments for the function.
When you create the arguments, configure the argument name, datatype, precision, and scale.
6. On the **Expression** tab, create an expression with the arguments that you created.
7. To validate the expression, click **Validate**.

8. Optionally, enter the expression in the **Expression** box. Then, click **Validate** to validate the expression.
9. To generate Java code by using the advanced interface, select the **Generate Advanced Code** option. Then, click **Generate**.

The Developer generates the function to invoke the expression on the **Functions** 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 based on the template for the expression.

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 by 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.

Use the following syntax:

```
(datatype) invokeJExpression(
    String expression,
    Object[] paramMetadataArray);
```

Input parameters for the `invokeJExpression` method are a string value that represents the expression and an array of objects that contain the expression input parameters.

The following table describes the parameters:

Parameter	Parameter Type	Datatype	Description
expression	Input	String	String that represents the expression.
paramMetadataArray	Input	Object[]	Array of objects that contain the input parameters for the expression.

You can add the `invokeJExpression` method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

Use the following rules and guidelines when you use the `invokeJExpression` method:

- Return datatype. The return datatype of the `invokeJExpression` method 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 the `invokeJExpression` method is `INSERT`.

To use a different row type for the return value, use the advanced interface.

- Null values. If you pass a null value as a parameter or the return value for the `invokeJExpression` method 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 the `String` datatype.

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` datatype.

Note: You must number the parameters that you pass to the expression consecutively, and start the parameter 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 method on the **Helpers** and **On Input** code entry tabs.

The following example shows how to complete a lookup on the `NAME` and `ADDRESS` input ports in a Java transformation and assign the return value to the `COMPANY_NAME` output port.

Enter the following code on the **On Input** code entry tab:

```
COMPANY_NAME = (String)invokeJExpression(":lkp.my_lookup(X1,X2)", new Object []
{str1 ,str2} );
generateRow();
```


Working with the Advanced Interface

In the advanced interface, you can use object oriented API methods to define, invoke, and get the result of an expression.

The following table describes the classes and API methods that are available in the advanced interface:

Class or API Method	Description
EDataType class	Enumerates the datatypes for an expression.
JExprParamMetadata class	Contains the metadata for each parameter in an expression. Parameter metadata includes datatype, precision, and scale.
defineJExpression API method	Defines the expression. Includes expression string and parameters.
invokeJExpression API method	Invokes an expression.
JExpression class	Contains the methods to create, invoke, get the metadata and get the expression result, and check the return datatype.

Invoking an Expression with the Advanced Interface

You can define, invoke, and get the result of an expression by using the advanced interface.

1. On the **Helpers** or **On Input** code entry tab, create an instance of JExprParamMetadata class for each argument for the expression and set the value of the metadata. Optionally, you can instantiate the JExprParamMetadata object in the defineJExpression method.
2. Use the defineJExpression method to get the JExpression object for the expression.
3. On the appropriate code entry tab, invoke the expression with the invokeJExpression method.
4. Check the result of the return value with the isResultNull method.
5. You can get the datatype of the return value or the metadata of the return value with the getResultDataType and getResultMetadata methods.
6. Get the result of the expression by using the appropriate API method. You can use the getInt, getDouble, getStringBuffer, and getBytes methods.

Rules and Guidelines for Working with the Advanced Interface

When you work with the advanced interfaces, you must be aware of rules and guidelines.

Use the following rules and guidelines:

- 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 by using the isResultNull method.

- 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 Date datatype as a String or long datatype.

To get the result of an expression that returns a Date datatype as a String datatype, use the getStringBuffer method. To get the result of an expression that returns a Date datatype as a long datatype, use the getLong method.

EDataType Class

Enumerates the Java datatypes used in expressions. Gets 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.

The following table lists the enumerated values for Java datatypes in expressions:

Datatype	Enumerated Value
INT	1
DOUBLE	2
STRING	3
BYTE_ARRAY	4
DATE_AS_LONG	5

The following example Java code shows how to use the EDataType class to assign a datatype of String to a 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 method to set the metadata for the input parameters. You can create an instance of the JExprParamMetadata object on the **Functions** 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);;
```

The following table describes the arguments:

Argument	Argument Type	Argument Datatype	Description
datatype	Input	EDatatype	Datatype of the parameter.
precision	Input	Integer	Precision of the parameter.
scale	Input	Integer	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 an expression, including the expression string and input parameters. Arguments for the defineJExpression method include an array of JExprParamMetadata objects that contains the input parameters and a string value that defines the expression syntax.

Use the following syntax:

```
defineJExpression(
    String expression,
    Object[] paramMetadataArray
);
```

The following table describes the parameters:

Parameter	Type	Datatype	Description
expression	Input	String	String that represents the expression.
paramMetadataArray	Input	Object[]	Array of JExprParamMetadata objects that contain the input parameters for the expression.

You can add the defineJExpression method to the Java code on any code entry tab except the **Imports** and **Functions** tabs.

To use the defineJExpression method, 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 a parameter to the defineJExpression method.

For example, the following Java code creates an expression to look up the value of two strings:

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
params[1] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
defineJExpression(":lkp.mylookup(x1,x2)",params);
```

Note: You must number the parameters that you pass to the expression consecutively and start the parameters with the letter x. For example, to pass three parameters to an expression, name the parameters x1, x2, and x3.

JExpression Class

Contains methods to create and invoke an expression, return the value of an expression, and check the return datatype.

The following table lists the methods in the JExpression class:

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.

Advanced Interface Example

You can use the advanced interface to create and invoke a lookup expression in a Java transformation.

The following example 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 on the **Helpers** tab:

```
JExpression 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 on the **On Input** tab to invoke the expression and return the value of the ADDRESS port:

```
...
if(!isJExprObjCreated)
```

```

{
    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 Class API Reference

The JExpression class contains API methods that let you create and invoke an expression, return the value of an expression, and check the return datatype.

The JExpression class contains the following API methods:

- `getBytes`
- `getDouble`
- `getInt`
- `getLong`
- `getResultDataType`
- `getResultMetadata`
- `getStringBuffer`
- `invoke`
- `isResultNull`

getBytes

Returns the value of an expression result as a `byte[]` datatype. Gets the result of an expression that encrypts data with the `AES_ENCRYPT` function.

Use the following syntax:

```
objectName.getBytes();
```

Use the following example Java code to get the result of an expression that encrypts the binary data using the `AES_ENCRYPT` function, where `JExprEncryptData` is a JExpression object:

```
byte[] newBytes = JExprEncryptData.getBytes();
```

getDouble

Returns the value of an expression result as a `Double` datatype.

Use the following syntax:

```
objectName.getDouble();
```

Use the following example Java code to get the result of an expression that returns a salary value as a double, where `JExprSalary` is a `JExpression` object:

```
double salary = JExprSalary.getDouble();
```

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();
```

getLong

Returns the value of an expression result as a Long datatype. Gets the result of an expression that uses a Date datatype.

Use the following syntax:

```
objectName.getLong();
```

Use the following example Java code to get the result of an expression that returns a Date value as a Long datatype, where `JExprCurrentDate` is a `JExpression` object:

```
long currDate = JExprCurrentDate.getLong();
```

getResultDataType

Returns the datatype of an expression result. Returns a value of `EDatatype`.

Use the following syntax:

```
objectName.getResultDataType();
```

Use the following example Java 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();
```

getResultMetadata

Returns the metadata for an expression result. 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 a `JExprParamMetadata` object. Use the `getScale`, `getPrecision`, and `getDataType` object methods to retrieve the result metadata.

Use the following syntax:

```
objectName.getResultMetadata();
```

Use the following example 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`.

getStringBuffer

Returns the value of an expression result as a String datatype.

Use the following syntax:

```
objectName.getStringBuffer();
```

Use the following example Java code to get the result of an expression that returns two concatenated strings, where `JExprConcat` is an `JExpression` object:

```
String result = JExprConcat.getStringBuffer();
```

invoke

Invokes an expression. Arguments for `invoke` include an object that defines the input parameters and the row type. You must instantiate a `JExpression` object before you can use the `invoke` method. For the row type, use `ERowType.INSERT`, `ERowType.DELETE`, and `ERowType.UPDATE`.

Use the following syntax:

```
objectName.invoke(  
    new Object[] { param1[, ... paramN ]},  
    rowType  
);
```

The following table describes the arguments:

Argument	Datatype	Input/Output	Description
<code>objectName</code>	<code>JExpression</code>	Input	<code>JExpression</code> object name.
<code>parameters</code>	-	Input	Object array that contains the input values for the expression.

For example, you create a function on the **Functions** 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);
```

isResultNull

Checks the value of an expression result.

Use the following syntax:

```
objectName.isResultNull();
```

Use the following example 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();  
}
```

CHAPTER 22

Joiner Transformation

This chapter includes the following topics:

- [Joiner Transformation Overview, 344](#)
- [Joiner Transformation Advanced Properties, 345](#)
- [Joiner Caches, 346](#)
- [Joiner Transformation Ports, 347](#)
- [Joiner Transformations in Dynamic Mappings, 348](#)
- [Port Selectors in a Joiner Transformation, 348](#)
- [Defining a Join Condition, 351](#)
- [Join Types, 354](#)
- [Sorted Input for a Joiner Transformation, 357](#)
- [Joining Data from the Same Source, 360](#)
- [Blocking the Source Pipelines, 362](#)
- [Joiner Transformation Performance Tips, 363](#)
- [Rules and Guidelines for a Joiner Transformation, 364](#)
- [Joiner Transformation in a Non-native Environment , 364](#)

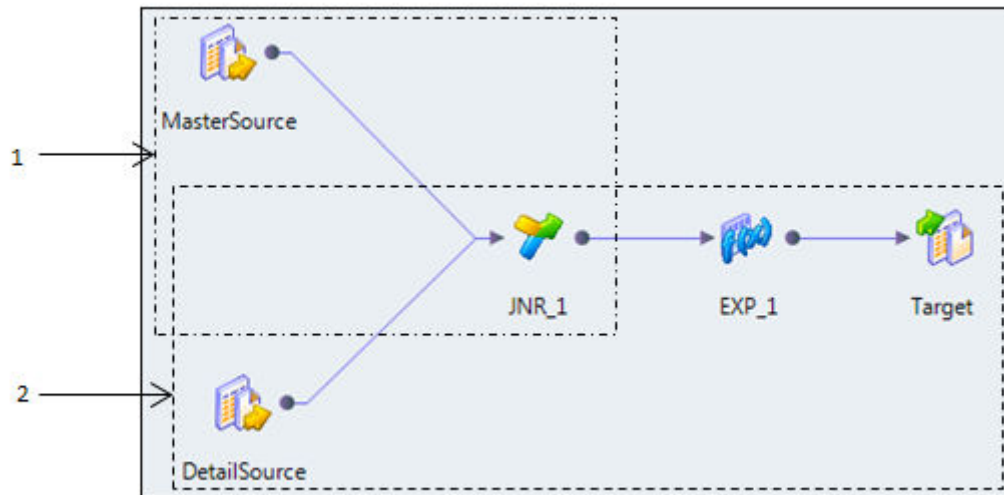
Joiner Transformation Overview

The Joiner transformation joins source data from two related heterogeneous sources from different locations or from different file systems. You can also join data from the same source. The Joiner transformation is a multiple-group active transformation.

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.

The following figure shows the master and the detail pipelines in a mapping with a Joiner transformation:



1. Master Pipeline
2. Detail Pipeline

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.

Joiner Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Joiner Transformation.

Configure the following properties on the **Advanced** tab:

Joiner Data Cache Size

Amount of memory that the Data Integration Service allocates to the data cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.

Joiner Index Cache Size

Amount of memory that the Data Integration Service allocates to the index cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.

Cache Directory

Directory where the Data Integration Service creates the index cache files and data cache files. Verify that the directory exists and contains enough disk space for the cache files.

Enter multiple directories separated by semicolons to increase performance during cache partitioning. Cache partitioning creates a separate cache for each partition that processes the transformation.

Default is the CacheDir system parameter. You can configure another system parameter or user-defined parameter for this property.

Sorted Input

Indicates that input data is presorted by groups. Choose Sorted Input to join sorted data. Using sorted input can increase performance.

Master Sort Order

Specifies the sort order of the master source data. Choose Ascending if the master source data is in ascending order. If you choose Ascending, also enable sorted input. Default is Auto.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

RELATED TOPICS:

- [“Cache Size” on page 71](#)

Joiner Caches

When you run a mapping that uses a Joiner transformation, the Data Integration Service creates an index cache and data cache in memory to run the transformation. If the Data Integration Service requires more space than available in the memory cache, it stores overflow data in cache files.

When you run a mapping that uses a Joiner transformation, the Data Integration Service reads rows from the master and detail sources concurrently and builds index and data caches based on the master rows. The Data Integration Service performs the join based on the detail source data and the cached master data.

The type of Joiner transformation determines the number of rows that the Data Integration Service stores in the cache.

The following table describes the information that the Data Integration Service stores in the caches for different types of Joiner transformations:

Joiner Transformation Type	Index Cache	Data Cache
Unsorted Input	Stores all master rows in the join condition with unique index keys.	Stores all master rows.
Sorted Input with Different Sources	Stores 100 master rows in the join condition with unique index keys.	Stores master rows that correspond to the rows stored in the index cache. If the master data contains multiple rows with the same key, the Data Integration Service stores more than 100 rows in the data cache.
Sorted Input with the Same Source	Stores all master or detail rows in the join condition with unique keys. Stores detail rows if the Data Integration Service processes the detail pipeline faster than the master pipeline. Otherwise, stores master rows. The number of rows it stores depends on the processing rates of the master and detail pipelines. If one pipeline processes its rows faster than the other, the Data Integration Service caches all rows that have already been processed. The service keeps the rows cached until the other pipeline finishes processing its rows.	Stores data for the rows stored in the index cache. If the index cache stores keys for the master pipeline, the data cache stores the data for master pipeline. If the index cache stores keys for the detail pipeline, the data cache stores data for detail pipeline.

Joiner Transformation Ports

A Joiner transformation has different port types that determine how the Data Integration Service performs the join.

A Joiner transformation has the following port types:

Master

Ports that link to the master source in the mapping.

Detail

Ports that link to the detail source in the mapping.

Dynamic Port

Receives or returns ports in a dynamic mapping. A dynamic port can receive one or more columns from an upstream transformation and create a generated port for each column. A dynamic output port can return one or more generated ports. You can define input rules to determine which columns a dynamic port receives.

You can change a port from a master port to a detail port. You can also change a port from a detail port to a master port. When you change the port type of one port, you change the port type of all ports. Therefore, when you change a master port to a detail port, you change all master ports to detail ports and all detail ports to master ports.

Joiner Transformations in Dynamic Mappings

You can use a Joiner transformation in a dynamic mapping. You can reference dynamic ports and generated ports in join conditions.

A dynamic mapping is a mapping in which the sources, targets, and transformation logic can change at run time. You can set parameters and rules to change the structure of the data. When you use a Joiner transformation in a dynamic mapping, the structure of the source can change. The input ports and the ports in the join condition change too.

You can perform the following tasks to configure a Joiner transformation in a dynamic mapping:

Define dynamic ports.

Define dynamic ports and generated ports to accommodate the different input columns from a dynamic source. You can include dynamic ports or generated ports in the join condition.

Define port selectors.

Define a port selector that contains the ports to use in the join condition. Configure selection rules that determine the ports to include in the port selector. You can parameterize the port selector to include specific ports at run time.

Parameterize the join condition.

You can parameterize the entire join condition. Configure an expression parameter for each join condition you might need.

For more information about dynamic mappings, see the *Informatica Developer Mapping Guide*.

Port Selectors in a Joiner Transformation

When the Joiner transformation has generated ports, you need to configure a join condition that is valid when the transformation has different generated ports at run time.

For example, a dynamic mapping includes a Joiner transformation with the following Join condition:

```
CustomerID = CustomerNo
```

CustomerID is a generated port in the Joiner transformation. Since the mapping has a dynamic source, the mapping can run with several different source file formats. The column that contains the customer number has a different name in each source file: `CustomerID`, `CustomerNum`, or `CustNO`.

You can create a port selector in the Joiner transformation to accommodate the different customer column names from the dynamic source. Configure a port selector with a selection rule that includes any port name with the prefix "Cust."

Then, configure the Join condition to include the port selector name instead of the CustomerID column name:

```
Customer_PortSelector = CustomerNo
```

The join condition is valid with any port name that starts with "Cust."

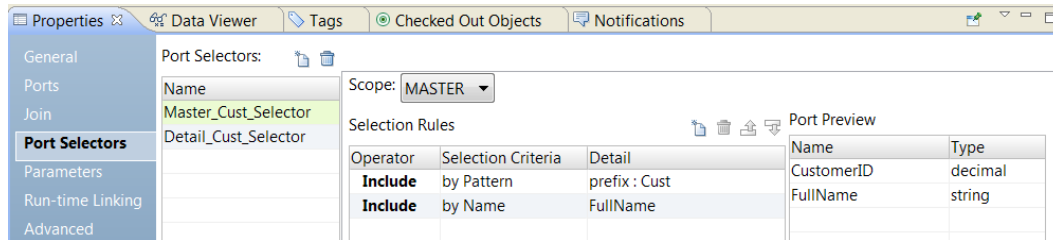
A port selector can have one or more ports in it. The join condition can include multiple ports if the master group and the detail group in the join condition contain the same number of ports.

Selection Rules

When you configure a port selector, you define selection rules to determine which generated ports to include. The selection rules are similar to the input rules that you can configure for dynamic ports.

A port selector can include ports or generated ports. Configure a port selector on the **Port Selector** tab.

The following image shows the **Port Selector** tab:



Configure the following properties for a port selector:

Name

Identifies the port selector. You can create multiple port selectors in a transformation and reference them in expressions.

Scope

Identifies a group of ports that the port selector applies to. Choose a Master or a Detail scope.

Selection Rules

Determines the ports to include in the port selector. When you create the selection rules, the **Port Preview** panel shows the ports that qualify from the current input ports. These ports might change. Configure the selection rules to accommodate ports from different sources.

You can create selection rules based on the following criteria:

Operator

Includes or excludes the ports that selection rules return. Default is include. You must include ports before you can exclude ports.

Selection Criteria

The type of selection rule you want to create. You can create a rule based on the port type or the column name. To include ports based on the column name, search for specific names or search for a pattern of characters in the name.

Detail

The values to apply to the selection criteria. If the selection criteria is by column name, configure the string or name to search for. If the selection criteria is by port type, select the port types to include.

The following table describes the selection criteria and how to specify the details for the criteria:

Selection Criteria	Description	Detail
All	Includes all ports.	No details required.
Name	Filters ports based on the port name.	Select the port names from a list of values or use a parameter of type Port or Port List.

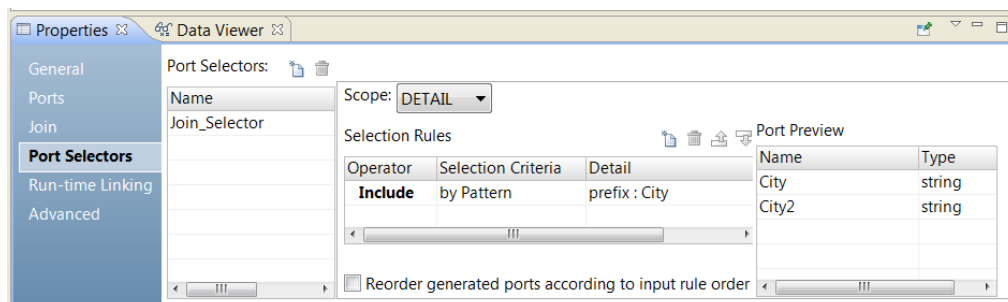
Selection Criteria	Description	Detail
Type	Filters ports based on the data type of each port.	Select data types from a list.
Pattern	Filters ports by a string of characters in the name or by a regular expression.	Choose prefix, suffix, or regular expression as the pattern type for the port name. Then, enter a value for the pattern or use a parameter of type String.

Creating a Port Selector

Create a port selector to determine which ports to use in a dynamic expression, a lookup condition, or a joiner condition.

1. Click the **Port Selectors** tab.
2. In the **Port Selectors** area, click **New**.
The Developer tool creates a port selector with a default selection rule that includes all ports.
3. In the **Port Selectors** area, change the port selector name to a unique name.
4. If you are working on the Joiner transformation or the Lookup transformation, choose the scope.
The available ports change based on the group of ports that you choose.
5. In the **Selection Rules** area, select an **Operator**.
 - Include. Create a rule that includes ports for the port selector. You must include ports before you can exclude ports.
 - Exclude. Create a rule that excludes specific ports from the port selector.
6. Choose the **Selection Criteria**.
 - By Name. Select specific ports by name. You can select the port names from a list of ports in the scope.
 - By Type. Select ports by type. You can select one or more data types.
 - By Pattern. Select ports by a pattern of characters in the port name. You can search with specific characters or you can create a regular expression.

The following image shows the Port Selector tab:



7. Click the **Detail** column.
The **Input Rule Detail** dialog box appears.
8. Select the values to filter ports by.

- By Name. Choose to create a port list by value or by a parameter. Click **Choose** to select the ports in the list.
- By Type. Select one or more data types from a list. The **Port Preview** area shows ports of the types that you select.
- By Pattern. Choose to search the prefix or suffix of the port name for a specific pattern of characters. Or, choose to create a regular expression to search with. Configure a parameter or configure the pattern to search with.

The **Port Preview** area shows the ports in the port selector as you configure the rules.

9. To reorder the ports in the port selector, select **Reorder generated ports according to the input rule order**.

Defining a Join Condition

The join condition contains ports from both input sources that the Data Integration Service uses to join two rows.

Depending on the type of join selected, the Data 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 mapping run, the Data 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.

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 Data Integration Service compares the ports in the order you specify.

If you join Char and Varchar datatypes, the Data 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 Data 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 Data 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.

You can define a simple or advanced condition type. You can also define an expression parameter. An expression parameter is a parameter that contains the join expression. You can change the parameter value at run time with a mapping parameter.

Simple Condition Type

Define a simple condition type for a sorted or unsorted Joiner transformation.

A simple condition includes one or more conditions that compare the specified master and detail sources. A simple condition must use the following format:

```
<master_port> operator <detail_port>
```

For a sorted Joiner transformation, the condition must use the equality operator.

For an unsorted Joiner transformation, the condition can use any of the following operators: =, !=, >, >=, <, <=.

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
```

The Developer tool validates datatypes in a simple condition. Both ports in the 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.

You can configure a list of join conditions in a simple condition. When you configure multiple join conditions, all the conditions must be true to make the join.

For example, you might configure the following statements in a simple condition:

```
StoreID = StoreNO  
Dept    = Department  
Salary  > Commission
```

If you view the same statements as an Advanced Condition, the join condition appears as the following expression:

```
StoreID = StoreNO AND Dept = Department AND (Salary > Commission)
```

Advanced Condition Type

Define an advanced condition type for an unsorted Joiner transformation.

An advanced condition can include any expression that evaluates to a Boolean or numeric value. An advanced condition can include any of the following operators: =, !=, >, >=, <, <=.

You can enter a constant for the join condition. The numeric equivalent of FALSE is zero (0). Any non zero value is the equivalent of TRUE. For example, the transformation contains a port named NUMBER_OF_UNITS with a numeric data type. You configure a filter condition to return FALSE if the value of NUMBER_OF_UNITS equals zero. Otherwise, the condition returns TRUE.

Note: You cannot use a single dynamic port or a port selector as a boolean value for a join condition.

To enter an expression in the join condition, choose the Advanced condition type on the **Join** tab. Use the Expression Editor to include ports, parameters, expressions, port selectors, and operators in the condition. You can use generated ports. You can enter a single port in the Expression Editor if the port type is numeric. However, you cannot enter one port selector as an expression.

For example, you want to join sources by matching an employee's full name. The master source includes a FirstName and a LastName port. The detail source includes a FullName port. Define the following condition to concatenate the master ports and match the full name in both sources:

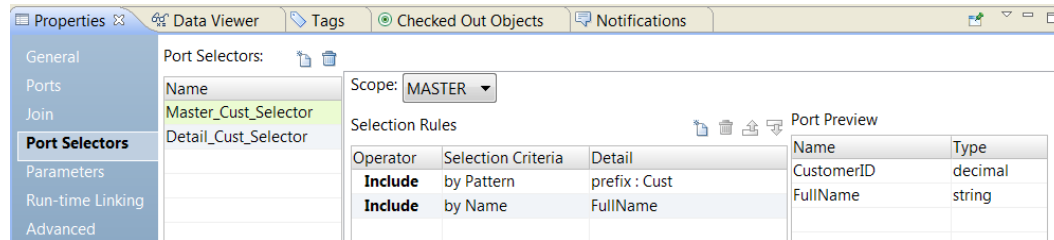
```
CONCAT(FirstName, LastName) = FullName
```


Port Selectors in Join Conditions

You can include port selectors in a join condition. The join condition must reference a port selector from the master group and a port selector from the detail group.

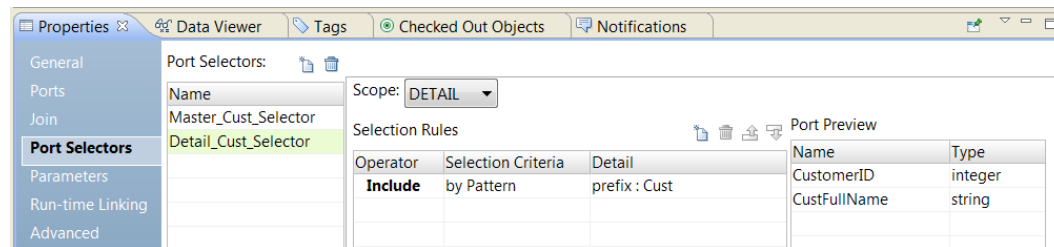
For example, the Joiner transformation has dynamic ports. You might need to compare multiple generated ports in the join condition.

The following image shows the fields in the port selector for the master group:



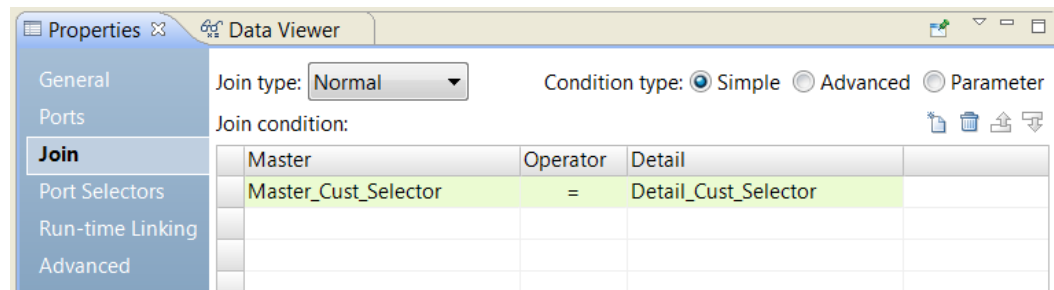
The Master_Cust_Selector contains has the CustomerID and FullName ports.

The following image shows the fields in the port selector for the detail group:



The Detail_Cust_Selector contains has the CustomerNo and CustFullName ports. These ports have the prefix Cust.

Create the following simple join condition:



The join condition compares each port in the Master_Cust_Selector to the Detail_Cust_Selector. The join condition is: `CustomerID = CustomerNo AND FullName = CustFullName`.

Each port selector must contain the same number of ports. The ports must be the same type.

Note: If you change the scope of a port selector and a simple type join condition is no longer valid, the Developer tool might switch the condition type to advanced. You can switch the join condition type back to a simple type on the **Join** tab.

Dynamic Ports in Join Conditions

You can reference a dynamic port in a port selector.

A dynamic port can contain one or more generated ports. If the join condition contains dynamic ports, the number of master ports must be the same as the number of detail ports.

For example a dynamic port A has 2 generated ports:

```
CustomerID
OrderID
```

Dynamic port B also has 2 generated ports:

```
CustomerNo
OrderNo
```

The following join condition is valid:

DynamicPortA = DynamicPortB

The join condition expands to the following expression:

```
CustomerID = CustomerNo AND OrderID = OrderNo
```

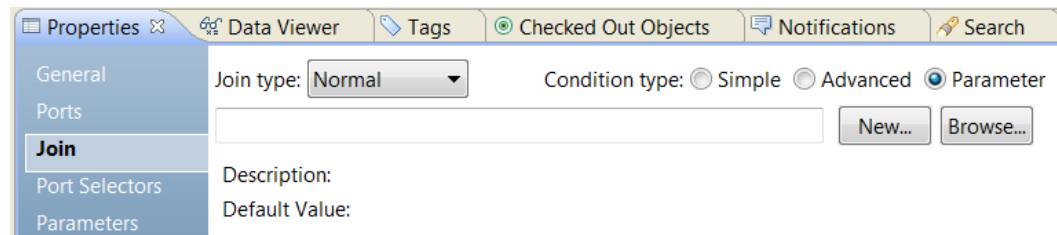
You can reference a port selector and a dynamic port in a join condition if the port selector contains the same number of ports as the dynamic port.

Expression Parameter

You can define an expression parameter that contains a join condition. You can choose the parameter as the join condition in the Joiner transformation.

To use a parameter for the join condition, select the parameter condition type on the Join tab.

The following image shows where to select the parameter condition type:



You can browse for an existing parameter or you can create a parameter. To create a parameter, click **New** and define the parameter. Create the expression in the expression editor.

Note: An expression parameter cannot contain other parameters. If you embed a parameter in an expression parameter, the Data Integration Service issues a runtime validation error.

Join Types

In a Joiner transformation, the join can originate from different types of sources.

The Joiner transformation supports the following types of join:

- 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 returns NULL and you do not want to insert NULLs in the target, you can set a default value for the corresponding port.

Normal Join

With a normal join, the Data Integration Service discards all rows of data from the master and detail source that do not match, based on the condition.

For example, you have two sources of data for auto parts called PARTS_SIZE and PARTS_COLOR.

The PARTS_SIZE data source is the master source and contains the following data:

PART_ID1	DESCRIPTION	SIZE
1	Seat Cover	Large
2	Ash Tray	Small
3	Floor Mat	Medium

The PARTS_COLOR data source is the detail source and contains the following data:

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 Data 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_COLOR.PART_ID2 = PARTS_SIZE.PART_ID1)
```

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 Ash Tray, the Data Integration Service populates the field with NULL.

The following example shows the equivalent SQL statement:

```
SELECT * FROM PARTS_SIZE LEFT OUTER JOIN PARTS_COLOR ON (PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

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 the following data:

PARTED	DESCRIPTION	SIZE	Color
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black

PARTED	DESCRIPTION	SIZE	Color
4	Fuzzy Dice	NULL	Yellow

Because no color is specified for the Ash Tray and no size is specified for the Fuzzy Dice, the Data 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)
```

Sorted Input for a Joiner Transformation

You can increase Joiner transformation performance with the sorted input option. You use sorted input when the data is sorted.

When you configure the Joiner transformation to use sorted data, the Data Integration Service increases performance by minimizing disk input and output. You see the greatest performance increase 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 Data Integration Service can use the sorted data when it processes the Joiner transformation. Complete the following steps to configure the mapping:

1. Configure the sort order of the data you want to join.
2. Add transformations that maintain the order of the sorted data.
3. 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.

Configuring the Sort Order

Configure the sort order to ensure that the Data Integration Service passes sorted data to the Joiner transformation.

To configure the sort order, use 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 relational data object to sort columns from the source database. Configure the order of the sorted ports the same in each relational data object.
- 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.

If you pass unsorted or incorrectly sorted data to a Joiner transformation configured to use sorted data, the mapping run fails. The Data Integration Service logs the error in the log file.

Adding Transformations to the Mapping

Add transformations to the mapping that maintain the order of the sorted data in a Joiner transformation.

You can place the Joiner transformation directly after the sort origin to maintain sorted data.

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:
 - Rank
 - Union
 - Unsorted Aggregator
 - Maplet that contains one the preceding 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.
 - 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.

Rules and Guidelines for Join Conditions

Certain rules and guidelines apply when you create join conditions for a sorted Joiner transformation.

Use the following guidelines when you create join conditions:

- You must define a simple condition type that uses the equality operator.
- 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.
- 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.
- If you join ports with Decimal data types, the precision of each port must belong to the same precision range.

You can use one of the following valid precision ranges:

- Decimal 0-18
- Decimal 19-28
- Decimal 29-38
- Decimal 39 and over

For example, if you define the condition `DecimalA = DecimalB` where `DecimalA` has precision 15 and `DecimalB` has precision 25, the condition is not valid.

Example of a Join Condition and Sort Order

This example shows a Joiner transformation that join the master and detail pipelines with sorted ports.

You configure Sorter transformations in the master and detail pipelines with the following sorted ports:

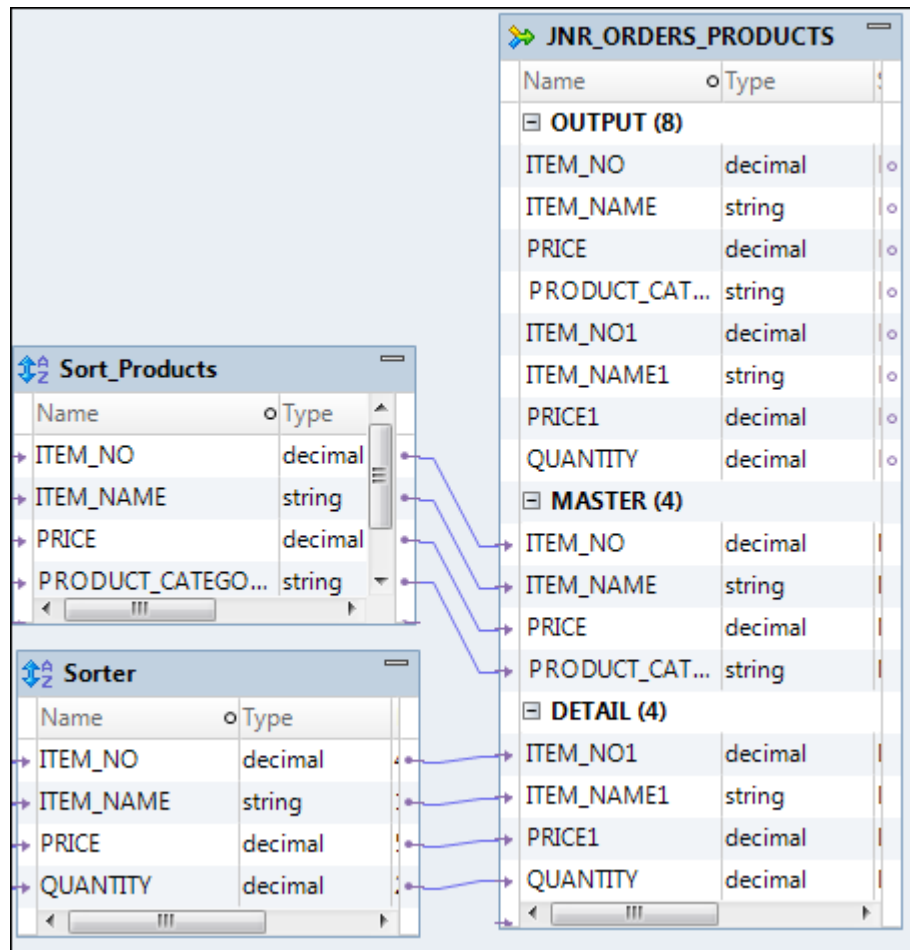
- ITEM_NO
- ITEM_NAME
- PRICE

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 Data Integration Service fails the mapping run.

The following figure shows a mapping configured to sort and join on the ports ITEM_NO, ITEM_NAME, and PRICE:



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 the Same Source

You can 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 data from the same source, 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 mapping input 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.

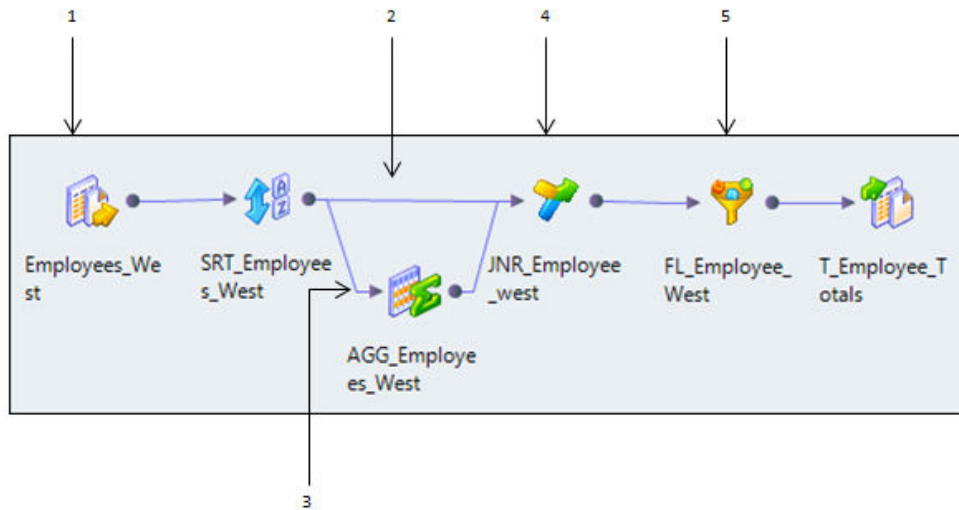
For example, you have a source with the following ports:

- 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. Joins 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.



1. Employees_West Source
2. Pipeline branch 1
3. Pipeline Branch 2
4. Sorted Joiner transformation
5. Filter out employees with less than above average sales

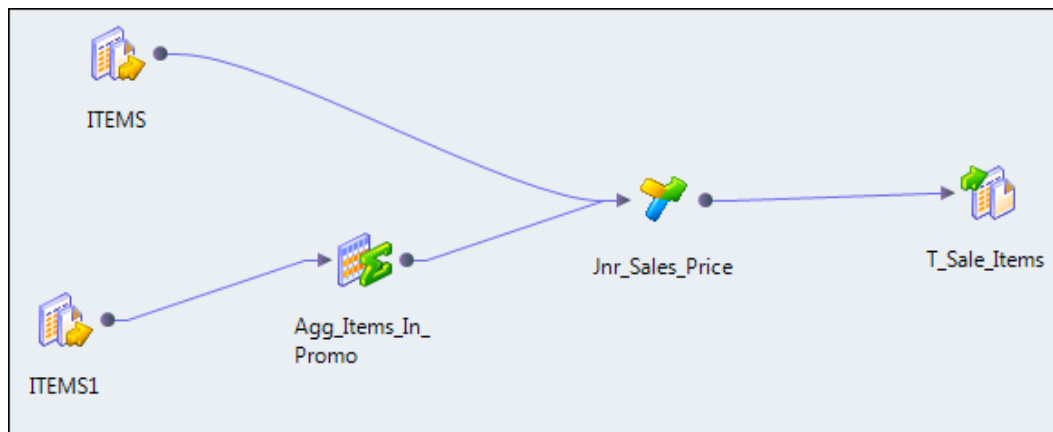
Joining two branches might decrease performance if the Joiner transformation receives data from one branch much later than the other branch. The Joiner transformation caches all the data from the first branch and writes the cache to disk if the cache fills. The Joiner transformation must then read the data from disk when it receives the data from the second branch.

Joining Two Instances of the Same Source

You can join data from the same source 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.

The following figure shows two instances of the same source joined with a Joiner transformation:



When you join two instances of the same source, the Data Integration Service reads the source data for each source instance. Performance can be slower than joining two branches of a pipeline.

Guidelines for Joining Data from the Same Source

Certain guidelines apply when you decide whether to join branches of a pipeline or join two instances of a source.

Use the following guidelines when you decide 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.
- 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 mapping with a Joiner transformation, the Data Integration Service blocks and unblocks the source data based on the mapping configuration and whether you configure the Joiner transformation for sorted input.

Unsorted Joiner Transformation

When the Data Integration Service processes an unsorted Joiner transformation, it reads all master rows before it reads the detail rows. The Data Integration Service blocks the detail source while it caches rows from the master source.

After the Data 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.

Sorted Joiner Transformation

When the Data 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 Data 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 Data 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 Data 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 Data Integration Service then performs the join based on the detail source data and the cache data. The number of rows the Data Integration Service stores in the cache depends on the source data and whether you configure the Joiner transformation for sorted input.

To increase performance for an unsorted Joiner transformation, use the source with fewer rows as the master source. To increase performance for a sorted Joiner transformation, use the source with fewer duplicate key values as the master.

Joiner Transformation Performance Tips

Use tips to increase Joiner transformation performance.

Joiner transformations can slow performance because they need additional space at run time to hold intermediary results. You can view Joiner performance counter information to determine whether you need to optimize the Joiner transformations.

Use the following tips to increase performance with the Joiner transformation:

Designate the master source as the source with fewer duplicate key values.

When the Data Integration Service processes a sorted Joiner transformation, it caches rows for one hundred unique keys at a time. If the master source contains many rows with the same key value, the Data Integration Service must cache more rows, which can decrease performance.

Designate the master source as the source with fewer rows.

The Joiner transformation compares each row of the detail source against the master source. The fewer rows in the master, the fewer iterations of the join comparison occur, which speeds the join process.

Perform joins in a database when possible.

Performing a join in a database is faster than performing a join in during the mapping run. The type of database join that you use can affect performance. Normal joins are faster than outer joins and result in fewer rows. Sometimes, you cannot perform the join in the database, such as joining tables from two different databases or flat file systems.

Join sorted data when possible.

Configure the Joiner transformation to use sorted input. The Data Integration Service increases performance by minimizing disk input and disk output. The greatest performance increase occurs when you work with large data sets. For an unsorted Joiner transformation, designate the source with fewer rows as the master source.

Optimize the join condition.

The Data Integration Service attempts to decrease the size of the data set of one join operand by reading the rows from the smaller group, finding the matching rows in the larger group, and then performing the join operation. Decreasing the size of the data set improves mapping performance because the Data Integration Service no longer reads unnecessary rows from the larger group source. The Data Integration Service moves the join condition to the larger group source and reads only the rows that match the smaller group.

Use the semi-join optimization method.

Use the semi-join optimization method to improve mapping performance when one input group has many more rows than the other and when the larger group has many rows with no match in the smaller group based on the join condition.

Rules and Guidelines for a Joiner Transformation

Certain rules and guidelines apply when you use a Joiner transformation.

The Joiner transformation accepts input from most transformations. However, you cannot use a Joiner transformation when either input pipeline contains an Update Strategy transformation.

Joiner Transformation in a Non-native Environment

The Joiner transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Joiner Transformation on the Blaze Engine

Mapping validation fails in the following situations:

- The transformation contains an inequality join and map-side join is disabled.
- The Joiner transformation expression references an unconnected Lookup transformation.

Map-side join is disabled when the Joiner transformation is configured for detail outer join or full outer join.

Joiner Transformation on the Spark Engine

Mapping validation fails in the following situation:

- The join condition is of binary data type or contains binary expressions.

Joiner Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- A Joiner transformation is downstream from an Aggregator transformation.
- A Joiner transformation is downstream from a Rank transformation.

- A streaming pipeline contains more than one Joiner transformation.
- A Joiner transformation joins data from streaming and non-streaming pipelines.

General Guidelines

To specify a join condition, select the `TIME_RANGE` function from the Advanced condition type on the Join tab and enter a join condition expression. The `TIME_RANGE` function determines the time range for the streaming events to be joined.

Joiner Transformation on the Databricks Spark Engine

Mapping validation fails in the following situation:

- The join condition is of binary data type or contains binary expressions.

CHAPTER 23

Key Generator Transformation

This chapter includes the following topics:

- [Key Generator Transformation Overview, 366](#)
- [Soundex Strategy, 367](#)
- [String Strategy, 367](#)
- [NYSIIS Strategy, 368](#)
- [Key Generator Output Ports, 368](#)
- [Configuring a Grouping Strategy, 369](#)
- [Key Creation Properties, 369](#)
- [Key Generator Transformation Advanced Properties, 370](#)
- [Key Generator Transformation in a Non-native Environment, 370](#)

Key Generator Transformation Overview

The Key Generator transformation is an active transformation that organizes records into groups based on data values in a column that you select. Use this transformation to sort records before passing them to the Match transformation.

The Key Generator transformation uses a grouping strategy to create group keys for the column you select. The strategies are String, Soundex, and NYSIIS. Records with common values in the selected field have a common group key value. The Match transformation processes records with common group key values together. This enables faster duplicate analysis in the Match transformation.

The number of comparison operations that the Match transformation must perform grows exponentially with the number of records in the data set. This exponential growth can consume significant amounts of computing resources. By creating group keys, the Key Generator transformation enables the Match transformation to compare records in smaller groups, which reduces processing time.

When you perform field matching, select a column for group key generation that is likely to provide useful groups for your matching needs. For example, a Surname column is likely to provide more meaningful group key data than a Forename column. But do not use the Surname column if you intend to select that column for duplicate analysis in the Match transformation.

The Key Generator transformation can also create a unique ID for each record. Each record that enters the Match transformation must contain a unique ID. Use the Key Generator transformation to create IDs for your data if none exist.

Soundex Strategy

The Soundex strategy analyzes words and creates group keys from codes that represent word pronunciation.

Soundex codes begin with the first letter in the word and followed by a series of numbers representing successive consonants. Use the Soundex strategy to assign the same code to words that sound similar. Configure the Soundex depth to define the number of alphanumeric characters that the strategy returns.

This strategy focuses on the sound of words instead of spelling, it can group alternate spellings and minor spelling variations. For example, the Soundex codes for `Smyth` and `Smith` are the same.

The Soundex strategy can also group mispronounced words. For example, the Soundex codes for the names `Edmonton` and `Edmonson` are the same.

Soundex Strategy Properties

Configure the Soundex strategy properties to determine the Soundex settings that the Key Generator transformation uses to create a group key.

The following table describes the Soundex strategy properties:

Property	Description
Soundex Depth	Determines the number of alphanumeric characters returned by the Soundex strategy. The default depth is 3. This depth creates a Soundex code consisting of the first letter in the string and two numbers that represent the next two distinct consonant sounds.

RELATED TOPICS:

- [“String Strategy Properties” on page 368](#)
- [“Key Creation Properties” on page 369](#)
- [“Configuring a Grouping Strategy” on page 369](#)

String Strategy

The String strategy creates group keys from substrings in input data.

You can specify the length and location of a substring within the input column. For example, you can configure this strategy to create a key from the first four characters in the input string.

String Strategy Properties

Configure the String strategy properties to determine the substrings that the Key Generator transformation uses to create a group key.

The following table describes the String strategy properties:

Property	Description
Start from left	Configures the transformation to read the input field from left to right.
Start from right	Configures the transformation to read the input field from right to left.
Start position	Specifies the number of characters to skip. For example, if you enter 3 for the Start position , the substring starts at the fourth character in the input field, starting from the side that you specify.
Length	Specifies the length of the string to use as a group key. Enter 0 to use the whole input field.

RELATED TOPICS:

- [“Soundex Strategy Properties” on page 367](#)
- [“Key Creation Properties” on page 369](#)
- [“Configuring a Grouping Strategy” on page 369](#)

NYSIIS Strategy

The NYSIIS strategy analyzes words and creates group keys from letters that represent word pronunciation.

While the Soundex strategy only considers the first vowel in a string, the NYSIIS strategy analyzes vowels throughout a string. The NYSIIS strategy converts all letters to one of six characters, and converts most vowels to the letter A.

Key Generator Output Ports

The Key Generator transformation output ports create IDs and group keys that the Match transformation uses to process records.

The following table describes the output ports for the Key Generator transformation:

Property	Description
SequenceID	Creates an ID that identifies each record in the source data set.
GroupKey	Creates the group keys that the Match transformation uses to process records.

When you create a reusable Key Generator transformation, use the **Overview** view to view the ports. When you add a nonreusable transformation to a mapping, use the **Ports** tab of the **Properties** view to view the ports.

Configuring a Grouping Strategy

To configure a grouping strategy, edit the properties in the **Strategies** view.

Before you configure a Key Generator strategy, add input ports to the Key Generator transformation.

1. Select the **Strategies** view.
2. Click the **New** button.
3. Select a grouping strategy.
4. Click **OK**.
5. In the **Inputs** column, select an input port.
6. Configure the strategy properties by clicking the selection arrow in the properties field.
7. Configure the key creation properties.

RELATED TOPICS:

- [“Soundex Strategy Properties” on page 367](#)
- [“String Strategy Properties” on page 368](#)
- [“Key Creation Properties” on page 369](#)

Key Creation Properties

Configure the key creation properties appropriate for the data you analyze.

The following table describes the key creation properties:

Property	Description
Sort results	Sorts the Key Generator transformation output using the GroupKey field. For field matching operations, you must select this option or verify that you provide the Match transformation with sorted data. Do not select this option for identity matching operations.
Generate sequence key automatically	Generates a sequence key field using the order of input data.
Use field as sequence key	Generates a sequence field for a column that you specify.
Sequence key field	Specifies the name of the sequence key field.

RELATED TOPICS:

- [“Soundex Strategy Properties” on page 367](#)
- [“String Strategy Properties” on page 368](#)
- [“Configuring a Grouping Strategy” on page 369](#)

Key Generator Transformation Advanced Properties

The Key Generator transformation contains advanced properties that determine the cache memory behavior and the tracing level.

You can configure the following advanced properties:

Cache File Directory

Specifies the directory to which the Data Integration Service writes temporary data for the current transformation. The Data Integration Service writes temporary files to the directory when the volume of input data is greater than the available system memory. The Data Integration Service deletes the temporary files after the mapping runs.

You can enter a directory path on the property, or you can use a parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory. The default value is the CacheDir system parameter.

Cache File Size

Determines the amount of system memory that the Data Integration Service uses to sort the input data on the transformation. You can use a parameter to specify the cache file size.

Before it sorts the data, the Data Integration Service allocates the amount of memory that you specify. If the sort operation generates a greater amount of data, the Data Integration Service writes the excess data to the cache directory. If the sort operation requires more memory than the system memory and the file storage can provide, the mapping fails.

If you do not specify a cache file size, the transformation applies the maximum memory value on the execution options of the Data Integration Service.

The transformation reads the value in bytes. The default value is 400,000 bytes. The maximum value is 2,147,483,647 bytes. You can use a parameter to specify the cache file size.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Key Generator Transformation in a Non-native Environment

The Key Generator transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Key Generator Transformation on the Blaze Engine

The Key Generator transformation is supported with the following restriction:

- When you run a mapping with a Key Generator transformation on more than one node, the sequence generation option on the transformation may not produce unique ID values.

Key Generator Transformation on the Spark Engine

The Key Generator transformation is supported with the following restriction:

- When you run a mapping with a Key Generator transformation on more than one node, the sequence generation option on the transformation may not produce unique ID values.

Key Generator Transformation on the Databricks Spark Engine

The Key Generator transformation is supported with the following restriction:

- When you run a mapping with a Key Generator transformation on more than one node, the sequence generation option on the transformation may not produce unique ID values.

CHAPTER 24

Labeler Transformation

This chapter includes the following topics:

- [Labeler Transformation Overview, 372](#)
- [When to Use a Labeler Transformation, 373](#)
- [Reference Data Use in the Labeler Transformation, 374](#)
- [Labeler Transformation Strategies, 376](#)
- [Labeler Transformation Ports, 377](#)
- [Character Labeling Properties, 377](#)
- [Token Labeling Properties, 379](#)
- [Configuring a Character Labeling Strategy, 381](#)
- [Configuring a Token Labeling Strategy, 382](#)
- [Labeler Transformation Advanced Properties, 383](#)
- [Labeler Transformation in a Non-native Environment, 383](#)

Labeler Transformation Overview

The Labeler transformation is a passive transformation that analyzes input port fields and writes text labels that describe the data in each field.

You use a Labeler transformation when you want to understand the types of information that a port contains. Use a Labeler transformation when you do not know the types of information on a port, or when you want to identify records that do not contain the expected types of information on a port.

A label is a string one or more characters that describes an input string. You configure the Labeler transformation to assign labels to input strings based on the data that each string contain.

When you configure the transformation, you specify the types of character or string to search for, and you specify the label that the transformation writes as output when it finds the associated character or string. You enter the character and string types to search for, and the labels to use, when you configure a labeling operation. Or, you use reference data objects to specify the characters, strings, and labels.

You configure the transformation to perform character labeling or token labeling:

Character Labeling

Writes a label that describes the character structure of the input string, including punctuation and spaces. The transformation writes a single label for each row in a column. For example, the Labeler transformation can label the ZIP Code 10028 as "nnnnn," where "n" stands for a numeric character.

Token Labeling

Writes a label that describes the type of information in the input string. The transformation writes a label for each token identified in the input data. For example, you can configure the Labeler transformation to label the string "John J. Smith" with the tokens "Word Init Word."

A token is a delimited value in an input string.

When the Labeler finds a character or string that matches a label that you specify, it writes the label name to a new output port.

The Labeler transformation uses reference data to identify characters and tokens. You select the reference data object when you configure an operation in a Labeler strategy.

When to Use a Labeler Transformation

The Labeler transformation writes a descriptive label for each value on a port.

The following examples describe some of the types of analysis you can perform with a Labeler transformation.

Find records with contact data

Configure the transformation with a reference table that contains a list of first names. Create a token labeling strategy to label any string that matches a value in the reference table. When you review the output data, any record that contains the label is likely to identify a person.

Find business records

Configure the transformation with a token set that contains a list of business suffixes, such as Inc, Corp, and Ltd. Create a token labeling strategy to label any string that matches a value in the reference table. When you review the output data, any record that contains the label is likely to identify a business.

Note: Use a token set of business suffixes you want to identify any business name. You can use a reference table of business names if you are certain that the table contains all the businesses you want to identify. For example, you can use a reference table that lists the corporations on the New York Stock Exchange.

Find telephone number data

Configure the transformation with character set that defines the character structure of a telephone number. For example, you can use a character set that recognizes different patterns of punctuation symbols and digits as United States telephone numbers. You can review the data to find records that do not contain the correct digits for a telephone number.

The character labels may use the following characters to analyze the column data:

```
c=punctuation character n=digit s=space
```

The following table shows sample telephone number structures:

Character Structure	Telephone number
cnnncsnnncnnncnnnnn	(212) 555-1212
nnnnnnnnnn	2125551212
cnnncnnncnnnn	+212-555-1212

Reference Data Use in the Labeler Transformation

Informatica Developer installs with different types of reference data objects that you can use with the Labeler transformation. You can also create reference data objects.

When you add a reference data object to Labeler transformation strategy, the transformation searches the input data on the strategy for values in the reference data object. The transformation replaces any value it finds with a valid value from the reference data object, or with a value that you specify.

The following table describes the types of reference data you can use:

Reference Data Type	Description
Character sets	Identifies different types of characters, such as letters, numbers, and punctuation symbols. Use in character labeling operations.
Probabilistic models	Adds fuzzy match capabilities to token label operations. The transformation can use a probabilistic model to infer the type of information in a string. To enable the fuzzy match capabilities, you compile the probabilistic model in the Developer tool. Use in token labeling operations.
Reference tables	Finds strings that match the entries in a database table. Use in token labeling and character labeling operations.
Regular expressions	Identifies strings that match conditions that you define. You can use a regular expression to find a string within a larger string. Use in token labeling operations.
Token sets	Identifies strings based on the types of information they contain. Use in token labeling operations. Informatica installs with token sets different types of token definitions, such as word, telephone number, post code, and product code definitions.

Character Sets

A character set contains expressions that identify specific characters and character ranges. You can use character sets in Labeler transformations and in Parser transformations that use token parsing mode.

Character ranges specify a sequential range of character codes. For example, the character range "[A-C]" matches the uppercase characters "A," "B," and "C." This character range does not match the lowercase characters "a," "b," or "c."

Use character sets to identify a specific character or range of characters as part of token parsing or labeling operations. For example, you can label all numerals in a column that contains telephone numbers. After labeling the numbers, you can identify patterns with a Parser transformation and write problematic patterns to separate outputs.

Probabilistic Models

A probabilistic model identifies tokens by the types of information that they contain and by the positions that they occupy in an input string.

A probabilistic model contains reference data values and label values. The reference data values represent the data on an input port that you connect to the transformation. The label values describe the types of information that the reference data values contain. You assign a label to each reference data value in the model.

To link the reference data values to the labels in a probabilistic model, you compile the model. The compilation process generates a series of logical associations between the data values and the labels. When you run a mapping that reads the model, the Data Integration Service applies the model logic to the transformation input data. The Data Integration Service returns the label that most accurately describes the input data values.

You create a probabilistic model in the Developer tool. The Model repository stores the probabilistic model object. The Developer tool writes the data values, the labels, and the compilation data to a file in the Informatica directory structure.

Reference Tables

A reference table is a database table that contains at least two columns. One column contains the standard or required version of a data value, and other columns contain alternative versions of the value. When you add a reference table to a transformation, the transformation searches the input port data for values that also appear in the table. You can create tables with any data that is useful to the data project you work on.

Regular Expressions

In the context of labeling operations, a regular expression is an expression that you can use to identify a specific string in input data. You can use regular expressions in Labeler transformations that use token labeling mode.

Labeler transformations use regular expressions to match an input pattern and create a single label. Regular expressions that have multiple outputs do not generate multiple labels.

Token Sets

A token set contains expressions that identify specific tokens. You can use token sets in Labeler transformations that use token labeling mode.

Use token sets to identify specific tokens as part of token labeling operations. For example, you can use a token set to label all email addresses that use that use an "AccountName@DomainName" format. After labeling the tokens, you can use the Parser transformation to write email addresses to output ports that you specify.

The Developer tool includes system-defined token sets that you can use to identify a wide range of patterns. Examples of system-defined token sets include:

- Words
- Numbers
- Phone numbers
- Email addresses
- Postal codes

- National identification numbers, such as Social Security Numbers
- Credit card numbers

Labeler Transformation Strategies

Use labeling strategies to assign labels to input data. To configure a labeling strategy, edit the settings in the **Strategies** view of a Labeler transformation.

When you create a labeling strategy, you add one or more operations. Each operation implements a specific labeling task.

The Labeler transformation provides a wizard that you use to create strategies. When you create a labeling strategy, you choose between character labeling or token labeling mode. You then add operations specific to that labeling mode.

Important: You can change the order of operations and strategies. The order of operations within a strategy can change the output of a strategy, because each operation reads the results of the preceding operation.

Character Labeling Operations

Use character labeling operations to create labels that describe the character patterns in your data.

You can add the following types of operations to a character labeling strategy:

Label Characters using Character Sets

Label characters using predefined character sets, such as digits or alphabetic characters. You can select Unicode and non-Unicode character sets.

Label Characters using Reference Table

Label characters with custom labels from a reference table.

Token Labeling Operations

Use token labeling operations to create labels that describe strings in your data.

The Labeler transformation can identify and label multiple tokens in an input string. For example, you can configure the Labeler transformation to use the US Phone Number and Email Addresses token sets. When the Labeler transformation processes the input string "555-555-1212 someone@somewhere.com," the output string is "USPHONE EMAIL."

You can add the following types of token labeling operations to a labeling strategy:

Label with Reference Table

Label strings that match reference table entries.

Label Tokens with Token Set

Label string patterns that match token set data or probabilistic model data.

Labeler Transformation Ports

You select the input and output ports you need for the labeling operations you configure in the transformation.

Labeler transformations use the following port:

Input Ports

Reads string input from upstream objects.

Labeled Output Ports

Writes the labels defined by transformation operations.

Tokenized Output Ports

Passes input strings that correspond to each label in the output. Select this port if you will add a Parser transformation downstream of the Labeler transformation in a mapplet or mapping, and you will configure the Parser transformation to run in pattern-based parsing mode. The Parser transformation associates the token labeling output with the data on the tokenized output ports.

Score Output Ports

Select to write the score values generated by probabilistic matching techniques in a token labeling operation.

When you run a token labeling operation that uses a probabilistic model, the operation generates a numerical score for each labeled string. The score represents the degree of similarity between the input string and the patterns defined in the probabilistic model.

Character Labeling Properties

Configure properties for character labeling operations on the **Strategies** view in the Labeler transformation.

General Properties

General properties apply to all character labeling operations you define in the strategy. You use the general properties to name the strategy and specify input and output ports.

The following table describes the general properties:

Property	Description
Name	Provides a name for the strategy.
Inputs	Identifies the input ports that the strategy operations can read.
Outputs	Identifies the output ports that the strategy operations can write to.
Description	Provides a text description of the strategy. This is an optional property.

Reference Table Properties

When you define a character labeling strategy, you can add operations to label with character sets and with reference tables. You use the reference table properties to specify how the transformation uses reference tables.

The following table describes the reference table properties:

Property	Description
Name	Provides a name for the operation.
Reference Table	Specifies reference tables that the transformation uses to label characters.
Label	Specifies the replacement text for input characters that match reference table entries.
Override other labels in strategy	Determines whether this labeling operation overrides other labeling operations.

Character Set Properties

When you define a character labeling strategy, you can add operations to label with character sets and with reference tables. You use the character set properties to specify how the transformation uses character sets.

The following table describes the character set properties:

Property	Description
Name	Provides a name for the operation.
Select Character Sets	Specifies the character sets that the transformation uses to label strings. You can override the replacement text for input strings that match the character set. Click the selection arrow in the Label column to enter custom replacement text.
Filter text	Uses characters or wildcards you enter to filter the list of character sets.
Add Character Set	Select to define a custom character set.
Edit	Edit the contents of a custom character set.
Import	Allows you to create nonreusable copies of character sets that are stored in content sets. Changes to the original character set do not update the copies you store in the Labeler transformation.
Remove	Deletes a custom character set.
Specify Execution Order	Sets the order in which the operation applies the token sets to the data. Use the Up and Down arrows to change the order.

Filter Properties

You can specify values to skip during a labeling operation. Use the **Ignore Text** properties to specify values to which labeling operations will not apply.

The following table describes the filter properties:

Property	Description
Search Term	Specifies strings that the transformation filters before performing labeling. Use this feature to specify exceptions to your defined labeling strategy.
Case Sensitive	Determines whether filtered strings must match the case of the search term.
Uppercase	Converts filtered strings to uppercase.
Start	Specifies the character position to start searching for filtered string.
End	Specifies the character position to stop searching for filtered string.

Token Labeling Properties

Configure properties for token labeling operations on the **Strategies** view in the Labeler transformation.

General Properties

General properties apply to all token labeling operations you define in the strategy. You use the general properties to name the strategy, specify input and output ports, and specify whether the strategy enables probabilistic matching techniques.

The following table describes the general properties:

Property	Description
Name	Provides a name for the strategy.
Inputs	Identifies the output port that the strategy operations can read.
Outputs	Identifies the output port that the strategy operations can write to.
Description	Describes the strategy. The property is optional.
Use probabilistic matching techniques	Specifies that the strategy can use a probabilistic model to identify token types.
Reverse Enabled	Indicates that the strategy reads input data from right to left. This property is disabled for probabilistic matching.
Delimiters	Specifies the characters that the transformation uses to evaluate substrings in input data. Default is space. The property is disabled in probabilistic labeling.

Property	Description
Tokenized Output Field	Indicates that the strategy writes multiple labels to an output port. Select this field to create input data for pattern-based parsing in the Parser transformation.
Score Output Field	Identifies the field that contains the score values generated in probabilistic matching. Set the score output field when you select the option to use probabilistic matching techniques.
Output Delimiter	Specifies a character to separate data values on the output port. Default is colon.

Token Set Properties

Token set properties apply when you configure a labeling operation to use token sets.

The following table describes the general properties:

Property	Description
Select Token Sets	Specifies the token sets that the transformation uses to label strings.
Filter text	Filters the list of token sets or regular expressions. Use text characters and wildcard characters as a filter.
Add Token Set	Use to define custom token sets.
Add Regular Expression	Use to define regular expressions that match an input pattern.
Edit	Edits the contents of a custom token set or a regular expression.
Import	Imports a nonreusable copy of a token set or regular expression from a folder in the Model repository. If you update the source object for the token set or regular expression, the Data Integration Service does not update the nonreusable copy.
Remove	Removes a custom token set or a regular expression.
Specify Execution Order	Sets the order in which the operation applies the token sets or regular expressions to the data. Use the Up and Down arrows to change the order.

Custom Label Properties

When you configure a token label operation, you can select the **Custom Label** view to create labels for specific search terms.

The following table describes the custom label properties:

Property	Description
Search Term	Identifies the string to search for.
Case Sensitive	Specifies whether the input data must match the case of the search term.
Custom Label	Identifies the custom label to apply.

Probabilistic Matching Properties

When you select the options to use probabilistic matching techniques, you can add a probabilistic model to the labeling operation. You cannot add a probabilistic model to a strategy that uses a token set or reference table.

The following table describes the properties associated with probabilistic matching:

Property	Description
Name	Provides a name for the operation.
Filter Text	Uses characters or wildcards you enter to filter the list of probabilistic models in the repository.
Probabilistic Model	Identifies the probabilistic model to use in the operation.

Reference Table Properties

Reference table properties apply when you configure a labeling operation to use a reference table.

The following table describes the reference table properties:

Property	Description
Name	Provides a name for the operation.
Reference Table	Specifies the reference table that the operation uses to label tokens.
Label	Specifies the text that the operation writes to a new port when an input string matches a reference table entry.
Case Sensitive	Determines whether input strings must match the case of reference table entries.
Replace Matches with Valid Values	Replaces labeled strings with the entry from the Valid column in the reference table.
Mode	Determines the token labeling method. Select Inclusive to label input strings that match reference table entries. Select Exclusive to label input strings that do not match reference table entries.
Set Priority	Determines whether reference table labeling operations takes precedence over token set labeling operations in a strategy. If you set this property, the transformation performs reference table labeling before token set labeling, and the token set analysis cannot overwrite the reference table label analysis.

Configuring a Character Labeling Strategy

To configure a labeling strategy, edit the settings in the **Strategies** view of the Labeler transformation.

1. Select the **Strategies** view, and click **New** to create a strategy.
The **Strategy** wizard opens.
2. Enter a name for the strategy.

3. Click the **Inputs** and **Outputs** fields to define ports for the strategy.
4. Optionally, enter a description of the strategy.
5. Select character labeling mode.
6. Click **Next**.
7. Select the type of character labeling operation you want to configure. You can configure the following operations:
 - Label characters using reference tables.
 - Label characters using character sets.
8. Click **Next**.
9. Configure the operation properties, and click **Next**.
10. Optionally, configure **Ignore Text** properties.
11. Click **Next** to add more operations to the strategy, or click **Finish**.

You can change the order that the transformation processes strategies and operations. In the **Strategies** view, select a strategy or operation and click **Move Up** or **Move Down**.

Configuring a Token Labeling Strategy

To configure a labeling strategy, edit the settings in the **Strategies** view of the Labeler transformation.

1. Select the **Strategies** view, and click **New** to create a strategy.
The **Strategy** wizard opens.
2. Enter a name for the strategy.
3. Click the **Inputs** and **Outputs** fields to define ports for the strategy.
4. Optionally, enter a description of the strategy.
5. Select token labeling mode.
Verify or edit the properties for the mode you select.
6. Click **Next**.
7. Select the type of token labeling operation you want to configure. You can configure the following operations:
 - Label tokens with a token set.
 - Label tokens with a reference table.
 - Label tokens using probabilistic matching.
8. Click **Next**.
9. Configure the operation properties, and click **Next**.
If you configure the strategy to use probabilistic matching, enter a label to use for tokens identified as residue data.
10. Optionally, configure **Custom Label** properties.
11. Click **Next** to add more operations to the strategy, or click **Finish**.

You can change the order that the transformation processes strategies and operations. In the **Strategies** view, select a strategy or operation and click **Move Up** or **Move Down**.

Labeler Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Labeler transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Labeler Transformation in a Non-native Environment

The Labeler transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 25

Lookup Transformation

This chapter includes the following topics:

- [Lookup Transformation Overview, 384](#)
- [Connected and Unconnected Lookups, 385](#)
- [Developing a Lookup Transformation, 387](#)
- [Lookup Query, 388](#)
- [Lookup Source Filter, 390](#)
- [Lookup Condition, 391](#)
- [Lookup Cache, 394](#)
- [Query Properties, 394](#)
- [Lookup Transformations in Dynamic Mappings, 395](#)
- [Define Dynamic Ports, 395](#)
- [Change the Lookup Source, 396](#)
- [Port Selectors, 400](#)
- [Run-time Properties, 406](#)
- [Advanced Properties, 407](#)
- [Creating a Reusable Lookup Transformation, 408](#)
- [Creating a Non-Reusable Lookup Transformation, 409](#)
- [Creating an Unconnected Lookup Transformation, 410](#)
- [Unconnected Lookup Example, 411](#)
- [Lookup Transformation in a Non-native Environment, 413](#)

Lookup Transformation Overview

The Lookup transformation is a passive or active transformation that looks up data in a flat file, logical data object, reference table, or relational table. The Lookup transformation can return one row or multiple rows from a lookup.

Before you create a Lookup transformation, create the lookup source. Import a flat file or relational database table as a physical data object. Or, create a logical data object or reference table to use as a lookup source. When you create a Lookup transformation, the Developer tool adds the columns from the data object or reference table as lookup ports in the transformation. After you create the transformation, configure one or

more output ports to return the lookup results. Configure the lookup conditions and configure other lookup properties.

When you run a mapping or preview data, the Integration Service queries the lookup source. The Integration Service queries the lookup source based on the lookup ports in the transformation, the lookup properties, and the lookup condition. The Lookup transformation returns the result of the lookup to the target or another transformation.

You can configure a connected or unconnected Lookup transformation. A connected transformation connects to another transformation in the mapping. An unconnected transformation receives input from a :LKP expression in another transformation. If the Lookup transformation performs a lookup on a logical data object, you must configure a connected Lookup transformation. Connect the Lookup transformation input ports to an upstream transformation or to an upstream source. Connect the output ports to a downstream transformation or to a downstream target.

You can use multiple Lookup transformations in a mapping.

You can perform the following tasks with a Lookup transformation:

- Get a related value. Retrieve a value from the lookup source based on a value in the input data. For example, the input data contains an employee ID. Retrieve the employee name from the lookup source by employee ID.
- Retrieve multiple rows from a lookup source.
- Perform a calculation. Retrieve a value from a lookup table and use the value in a calculation. For example, retrieve a sales tax percentage, calculate a tax, and return the tax to a target.
- Perform an unconnected lookup with a :LKP expression in a transformation that accepts expressions. Filter the results with another expression in the transformation.
- Parameterize the lookup source and the lookup condition to use a Lookup transformation in a dynamic mapping.

Connected and Unconnected Lookups

You can configure a connected Lookup transformation or an unconnected Lookup transformation. A connected Lookup transformation is a transformation that has input and output ports that you connect to other transformations in a mapping. An unconnected Lookup transformation appears in the mapping, but is not connected to other transformations.

An unconnected Lookup transformation receives input from the result of a :LKP expression in a transformation such as an Expression transformation or Aggregator transformation. The :LKP expression passes arguments to the Lookup transformation and receives a result back from the Lookup transformation. The :LKP expression can pass lookup results to another expression in the Expression or Aggregator transformation to filter results.

The following table lists the differences between connected and unconnected lookups:

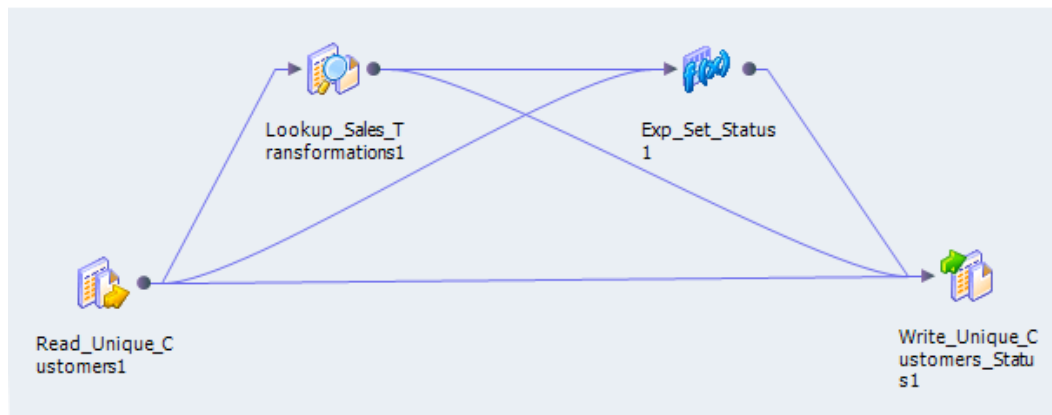
Connected Lookup	Unconnected Lookup
Receives input values directly from the pipeline.	Receives input values from the result of a :LKP expression in another transformation.
Use a dynamic or static cache.	Use a static cache.

Connected Lookup	Unconnected Lookup
Cache includes the lookup source columns in the lookup condition and the lookup source columns that are output ports.	Cache includes all lookup and output ports in the lookup condition and the lookup/return port.
Returns multiple columns from the same row or insert into the dynamic lookup cache.	Returns one column from each row to a return port.
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 the in the cache or leaves the row unchanged.	If a match occurs for the lookup condition, the Integration Service returns the result of the lookup condition to the return port.
Passes multiple output values to another transformation. Link lookup/output ports to another transformation.	Returns one output value to another transformation. The Lookup transformation return port passes the value to the port that contains the :LKP expression in the other transformation.
Supports user-defined default values.	Does not support user-defined default values.

Connected Lookups

A connected Lookup transformation is a Lookup transformation that is connected to a source or target in a mapping.

The following figure shows a mapping with a connected Lookup transformation:



When you run a mapping that contains a connected Lookup transformation, the Integration Service performs the following steps:

1. The Integration Service passes values from another transformation to input ports in the Lookup transformation.
2. For each input row, the Integration Service queries the lookup source or cache based on the lookup ports and the lookup 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 returns data from the query and passes it to the next transformation in the mapping.

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 Lookups

An unconnected Lookup transformation is a Lookup transformation that is not connected to a source or target in the mapping. Call the lookup with a :LKP expression in a transformation that allows expressions.

The syntax for the lookup expression is `:LKP lookup_transformation_name(argument, argument, ...)`

The order in which you list each argument must match the order of the lookup conditions in the Lookup transformation. The Lookup transformation returns the result of the query through the Lookup transformation return port. The transformation that calls the lookup receives the lookup result value in the port that contains the :LKP expression. If the lookup query fails to return a value, the port receives a null value.

When you perform an unconnected lookup, you can perform the same lookup multiple times in a mapping. You can test the results of the lookup in another expression and filter rows based on the results.

When you run a mapping that contains an unconnected Lookup transformation, the Integration Service performs the following steps:

1. An unconnected Lookup transformation receives input values from the result of a :LKP expression in another transformation, such as an Aggregator transformation, Expression transformation, or Update Strategy transformation.
2. The Integration Service queries the lookup source or cache based on the lookup ports and condition in the Lookup transformation.
3. The Integration Service returns a value through the return port of the Lookup transformation.
4. The Integration Service passes the return value to the port that contains the :LKP expression.

Developing a Lookup Transformation

When you develop a Lookup transformation, you need to consider factors such as the lookup source type and the lookup condition.

Consider the following factors when you develop a Lookup transformation:

- Whether you want to create the transformation from a flat file, logical data object, reference table, or relational data object. Before you create a Lookup transformation, create the lookup source. Import a flat file or relational database table as a physical data object. Or, create a logical data object or reference table to use as a lookup source.
- The output ports for the transformation.
- The lookup conditions in the transformation.

- Whether you want the Integration Service to cache lookup data. The Integration Service can cache data for flat files, reference tables, or relational data objects.

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 lookup query when the first row enters the Lookup transformation.

If you use a lookup against a relational table, you can override the lookup query. You can use the override to add a WHERE clause or transform the lookup data before it is cached.

If you configure a SQL override and a filter on the lookup query, the Integration Service ignores the filter.

Default Lookup Query

The default lookup query contains the following statements:

SELECT

The SELECT statement includes all the lookup ports in the mapping. To view the SELECT statement for the lookup query, select the Use Custom Query property.

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.

SQL Override for a Lookup Query

You can override the lookup query for a relational lookup. You can add a WHERE clause and transform the lookup data before it is cached.

You can use reserved words and forward slashes in table names and column names.

You can enter a query to override the default lookup query completely. Or, you view and edit the default lookup query. The default lookup query includes the lookup ports, output ports, and the return port.

When you use an SQL override, the query appends the clause ORDER BY 1. The clause orders the data to reliably provide first and last values for other clauses.

Note: You can manually validate the SQL by running the following query in a Hive command line utility:

```
CREATE VIEW <table name> (<port list>) AS <SQL>
```

where:

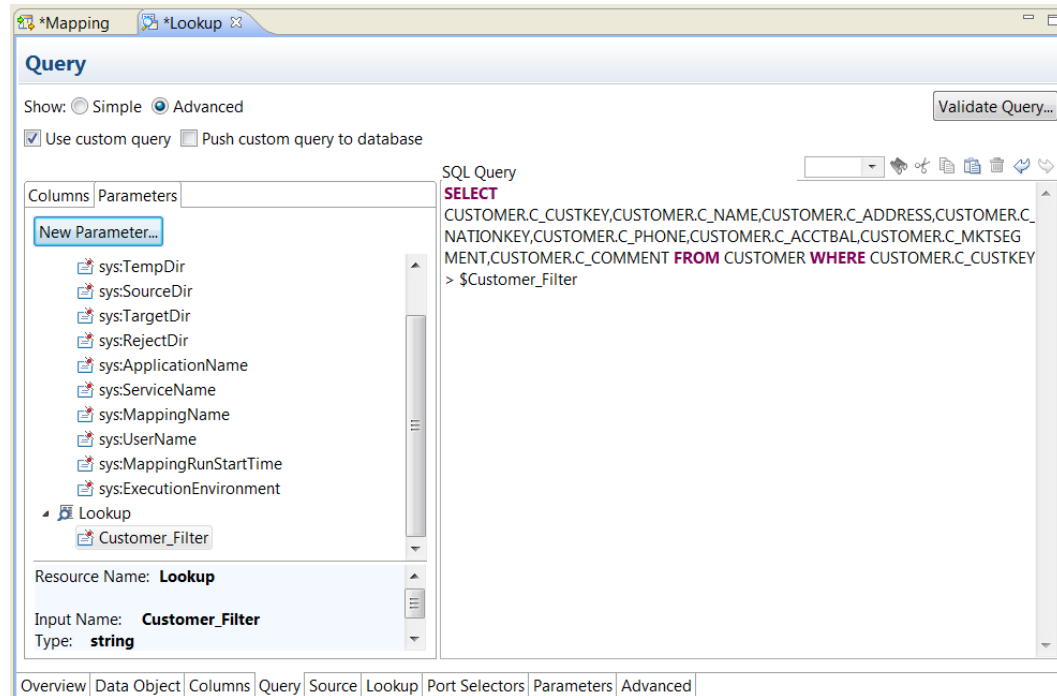
- <table name> is a name of your choice
- <port list> is the comma-delimited list of ports in the source
- <SQL> is the query to validate

Parameters in an SQL Override Query

You can use system parameters or user-defined parameters in the lookup query of a Lookup transformation. The SQL Editor provides a list of the system parameters and user-defined parameters that you can select from.

You can browse for or you can create user-defined parameters in the **Query** tab of a Lookup transformation. Define a default value for each parameter. You can override a default value by binding a mapplet or mapping parameter to the transformation parameter after you add the Lookup transformation to a mapping.

The following image shows the Lookup transformation Query tab:



Reserved Words

If any lookup name or column name contains a database reserved word, such as MONTH or YEAR, the mapping 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 mapping, it searches the reswords.txt file and places quotes around reserved words, and then executes the SQL against source, target, and lookup databases.

You might need to enable some databases, such as Microsoft SQL Server and Sybase, to use SQL-92 standards regarding quoted identifiers. Use environment SQL to issue the command. For example, with Microsoft SQL Server, use the following command:

```
SET QUOTED_IDENTIFIER ON
```

Guidelines for Overriding the Lookup Query

Certain rules and guidelines apply when you override a lookup query.

Consider the following guidelines when you override the lookup SQL query:

- You can override the lookup SQL query for relational lookups.

- Add a source lookup filter to filter the rows that are added to the lookup cache. This ensures that the Integration Service inserts rows in the dynamic cache and target table that match the WHERE clause.
- If multiple Lookup transformations share a lookup cache, use the same lookup SQL override for each Lookup transformation.
- If a table name or column name in the lookup query contains a reserved word, enclose the reserved word in quotes.
- To override the lookup query for an uncached lookup, choose to return any value when the Integration Service finds multiple matches.
- You cannot add or delete any columns from the default SQL statement.
- The Developer tool does not validate the syntax of the SQL query. If the SQL override in an unconnected lookup query is not valid, the mapping fails.

Overriding the Lookup Query

You can override the default lookup SQL query to create a customized query on the lookup source.

1. On the **Properties** view, select the **Query** tab.
2. Select **Advanced**.
3. Select **Use Custom Query**.
4. Edit the lookup query in the SQL Query area.
You can double-click a table name, a column name, or a parameter to add it to the query.
5. Click **Validate Query** to validate the lookup query.
6. Select **Push Custom Query to Database** to run the lookup query in the database.

Lookup Source Filter

You can configure a Lookup source filter for a relational Lookup transformation that has caching enabled. Add the lookup source filter to limit the number of lookups that the Integration Service performs on a lookup source table.

When you configure a Lookup source filter, the Integration Service performs lookups based on the results of the filter statement. For example, you might need to retrieve the last name of every employee whose ID is greater than 510.

You configure the following lookup source filter on the EmployeeID column:

```
EmployeeID >= 510
```

EmployeeID is an input port in the Lookup transformation. When the Integration Service reads the source row, it performs a lookup on the cache when the value of EmployeeID is greater than 510. When EmployeeID is less than or equal to 510, the Lookup transformation does not retrieve the last name.

When you add a lookup source filter to the Lookup query for a mapping configured for pushdown optimization, the Integration Service creates a view to represent the SQL override. The Integration Service runs an SQL query against this view to push the transformation logic to the database.

Filtering Source Rows in a Lookup

You can configure a Lookup source filter for a relational Lookup transformation that has caching enabled. Filter source rows in a lookup to limit the number of lookups that the Integration Service performs on a lookup source table.

1. On the **Properties** view, select the **Query** tab.
2. In the **Filter** option, click **Edit**.
3. In the SQL Editor, select the input ports or enter any Lookup transformation port that you want to filter.
4. Enter a filter condition.

Do not include the keyword WHERE in the filter condition. Enclose string mapping parameters and variables in string identifiers.

5. Click **Validate Query** to validate the syntax of the filter condition.

Lookup Condition

The Data Integration Service looks up data in the lookup source based on a lookup condition. When you configure a lookup condition in a Lookup transformation, you compare the value of one or more columns in the source data with values in the lookup source or cache.

For example, the source data contains an `employee_number`. The lookup source table contains `employee_ID`, `first_name`, and `last_name`. You configure the following lookup condition:

```
employee_ID = employee_number
```

For each `employee_number`, the Data Integration Service returns the `employee_ID`, `last_name`, and `first_name` column from the lookup source.

The Data Integration Service can return more than one row from the lookup source. You configure the following lookup condition:

```
employee_ID > employee_number
```

The Data Integration Service returns rows for all `employee_ID` numbers greater than the source `employee number`.

Null Values in a Data Object Lookup

When an input to the Lookup condition is NULL, a data object Lookup transformation returns a single row with null values for output-only ports, and values from input rows for pass-through ports.

For example, the following lookup condition performs a lookup on a data source that contains one or more rows where the value for `employee_ID` is NULL:

```
employee_ID = employee_number
```

In this example, you use a lookup table with the following data:

EMPLOYEE_ID	LAST_NAME
1294765	Hara
1356356	Carver

EMPLOYEE_ID	LAST_NAME
1407207	NULL
1570348	Draper
NULL	Limonov

You compare the following input values from your data source with the lookup table:

```

EMPLOYEE_NUMBER
-----
1294765
1356356
1407207
1648246
NULL

```

In this example, the Lookup condition produces the following results:

```

1294765, Hara
1356356, Carver
1407207, NULL
NULL, NULL
NULL, NULL

```

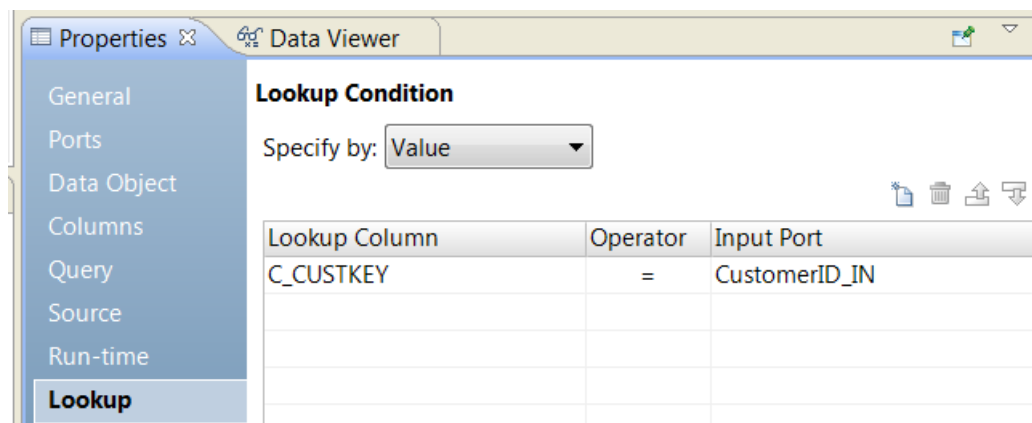
The Lookup condition finds matches between EMPLOYEE_ID and EMPLOYEE_NUMBER for the first two rows. For the third row, the lookup source contains a row with a NULL value that does not participate in the lookup condition. It matches the lookup condition, and returns a result with the NULL value for the non-lookup column.

For the fourth and fifth rows, the Lookup condition does not find a match, and returns NULL for both values. For the fifth row, note that the Lookup condition does not find a match again, because NULL does not match anything, including NULL.

Configure the Lookup Condition

The Lookup condition is an expression that determines which rows to retrieve from the lookup source. Configure the lookup condition on the **Lookup** tab of the **Properties** view.

The following image shows a lookup condition to perform a lookup by customer number:



You can configure the following options on the **Lookup** tab:

Specify by

Choose **Value** to select the lookup column and input port names. Select **Parameter** to configure an expression parameter to define the lookup condition.

Lookup column

The column in the lookup source to match with a column from the input row. You can include multiple columns in the lookup condition.

Operator

The operator that determines the condition to search for between the lookup column and the input port. Operators include =, !=, >, <, >=, <=.

Input Port

The input port that contains the value to search for in the lookup source. You can compare more than one input port to ports in the lookup source.

Rules and Guidelines for Lookup Transformation Conditions

Certain rules and guidelines apply when you enter a condition for a Lookup transformation.

Consider the following rules and guidelines when you enter a condition for a Lookup transformation.

- The datatypes for the columns in a lookup condition must match.
- Use one input port for each lookup port in the lookup condition. You can use the same input port in more than one condition in a transformation.
- If you use a port selector or a dynamic port in a lookup condition, the lookup condition considers all the ports in the expression.
- You can use a dynamic input port or a port selector as the input port of a lookup condition. The number of generated ports in the input port must be equal to the number of ports in the lookup column.
- When processing a Lookup transformation with multiple lookup conditions, the Integration Service returns rows that match all the lookup conditions.
- You can create an expression parameter to parameterize the lookup condition in a nonreusable lookup transformation.
- To increase lookup performance, enter conditions in the following order:
 - Equal to (=)
 - Less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=)
 - Not equal to (!=)
- Use one of the following operators when you create a lookup condition: =, >, <, >=, <=, or !=.
- The Integration Service processes lookup matches differently based on whether you configure the Lookup transformation with a dynamic lookup cache, static lookup cache, or uncached lookup.
- The Integration Service matches null values. For example, the Integration Service considers a lookup port and an input port equal if they both have null values.
- If the columns in a lookup condition are Decimal data types, the precision of each column must belong to the same precision range. Valid precision ranges include:
 - Decimal 0-18
 - Decimal 19-28
 - Decimal 29-38
 - Decimal 39 and over

For example, if you define the condition `DecimalA = DecimalB` where `DecimalA` has precision 15 and `DecimalB` has precision 25, the lookup condition is not valid.

Lookup Cache

You can increase performance by caching a large lookup source or small lookup tables. When you cache the lookup source, the Integration Service queries the lookup cache instead of querying the lookup source for each input row.

Based on your business requirements, you can create different types of lookup caches. You can create a static or dynamic cache. You can create a persistent or non-persistent cache. You can share a cache among multiple Lookup transformations.

If the Lookup transformation is in a dynamic mapping, you can have a persistent or a nonpersistent cache. When you persist a cache and you change the lookup source with a parameter, the mapping fails. The mapping also fails if you change the control file for a flat file lookup source.

Note: You cannot use a dynamic lookup cache or a persisted lookup cache when the Lookup transformation contains a dynamic port or a parameterized lookup source.

Query Properties

Configure query properties to view or change the lookup query on a relational lookup table. You can apply a filter on the lookup or customize the lookup query.

The following table describes query properties for Lookup transformations that perform relational lookups:

Property	Description
Simple	Select to view the lookup query and apply a filter on the lookup.
Advanced	Select to view the query, customize the query, or run the query in the database that contains the relational lookup table.
Filter	Enter a filter to reduce the number of rows the Integration Service queries. You must select the Simple option to view this option.
Use Custom Query	Select to override the lookup query. You must select the Advanced option to view this option.
Push Custom Query to Database	Select to run the query in the database that contains the relational lookup table. You must select the Advanced option to view this option.
SQL Query	Displays the SQL query used to perform the lookup. You can customize the SQL query. You must select the Advanced option to view this option.

Lookup Transformations in Dynamic Mappings

You can use a Lookup transformation in a dynamic mapping. You can configure dynamic ports to receive and return different ports based on the source data. You can parameterize the lookup source and the lookup condition to perform a lookup based on the different ports.

A dynamic mapping is a mapping in which the sources, targets, and transformation logic can change at run time. You can set parameters and rules to change the structure of the data. When you use a Lookup transformation in a dynamic mapping, the input ports of the Lookup transformation might change based on the source data. The structure of the lookup source and the ports in the lookup condition might change.

Note: When the Lookup transformation contains a dynamic port or a parameterized lookup source, you cannot persist the lookup cache. You also cannot configure a dynamic cache.

You can perform the following tasks for a Lookup transformation to use the transformation in a dynamic mapping:

Define dynamic ports

Define dynamic ports and generated ports to accommodate changes to the input columns.

Parameterize the lookup source

Assign a parameter for the data object that defines the lookup source. You can parameterize the lookup source in a nonreusable Lookup transformation.

Define port selectors

Define a port selector that specifies the ports to use in the lookup condition. You can parameterize the port selector ports in a nonreusable Lookup transformation.

Parameterize the lookup condition

Create an expression parameter and define a default value that contains a complete expression.

For more information about dynamic mappings, see the *Informatica Developer Mapping Guide*.

Define Dynamic Ports

You can define dynamic ports in the Lookup transformation.

You can reference a dynamic port in input column of a lookup condition. If the dynamic port contains multiple generated ports, you can use a port selector for the lookup column element of the lookup condition. The dynamic input port must contain the same number of ports as the port selector in the lookup condition.

If the dynamic port contains one value, you can use a single port in the lookup column element of the lookup condition.

You can reference generated ports in the lookup condition. However, if a source in the dynamic mapping changes, the generated port might not exist. The mapping fails.

Change the Lookup Source

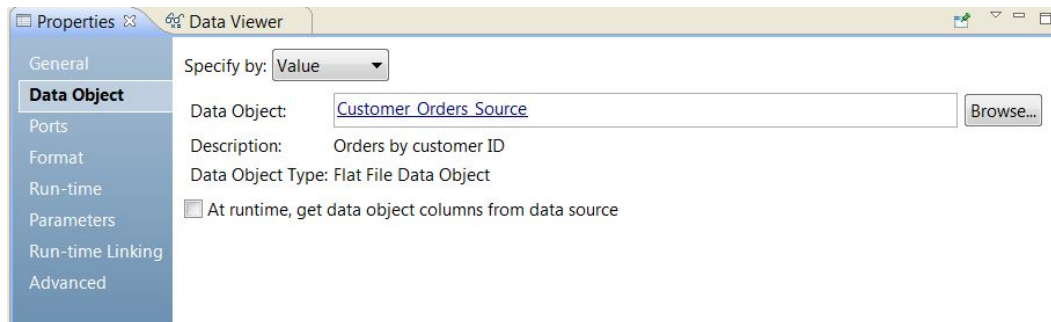
You can change the data object that is the lookup source for a reusable Lookup transformation. You can configure a parameter that determines the data object to use as the lookup source at run time.

When you create a transformation from a physical data object, information about the data object appears on the **Data Object** tab of the transformation properties. You can click the data object name to view the physical data object definition from the Model repository.

You can change the data object for the transformation by browsing for a different physical data object in the Model repository. When you change the data object, the transformation uses the run time properties and the advanced properties of the data object that you select.

You can update the structure of the data object at run time based on changes in the data source. The data source is the physical file or the database table that the data object represents. When you enable the Data Integration Service to get data columns from the data source, the Data Integration Service examines the structure of the data source. The Data Integration Service updates the data object ports in the transformation instance based on the data source. The Data Integration Service does not change the physical data object definition in the Model repository.

The following image shows the **Data Object** tab:



The **Data Object** tab has the following fields:

Specify by

Choose **Value** to enter a specific data object name. Choose **Parameter** to parameterize the data object.

Data object

The name of the data object in the Model repository. You can click the **Data Object** link to open the data object definition from the repository. You can also Browse for a different data object in the Model repository.

Description

The description of the data object in the repository. Read-only.

Data object type

Describes the type of data object, such as a flat file data object, a relational table object, or a customized data object.

At run time, get data object columns from the data source

The Data Integration Service fetches metadata and data definition changes from the data file or the table that the data object refers to and updates the structure of the data object for the transformation instance at run time.

To preview how the Data Integration Service fetches metadata and data definition changes at run time, view the mapping with resolved parameters.

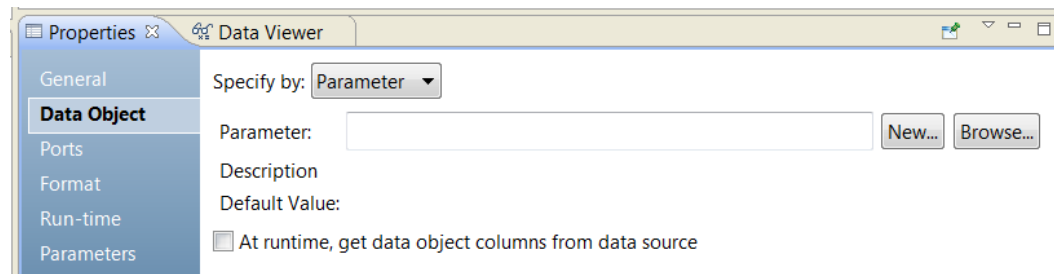
Parameterize the Lookup Source

You can configure a parameter for the Lookup source in a nonreusable Lookup transformation.

To parameterize a data object, choose **Specify by Parameter** in the **Data Object** tab. The properties in the **Data Object** tab change.

To parameterize the data object, create a resource type parameter or browse for a resource parameter that you already created. The parameter default value is the name of physical data object in the Model repository. When you create a default parameter value, you select a physical data object name from a list of data objects in the repository.

The following image shows the **Data Object** tab when you specify the data object by a parameter:



The **Data Object** tab has the following options by parameter:

Parameter

The name of a resource parameter that you configured as the data object. Read-only.

Description

The description of the parameter. Read-only.

New

Create a resource parameter. Browse for and select a data object in the Model repository for the parameter default value.

Browse

Browse for a resource parameter and select the parameter.

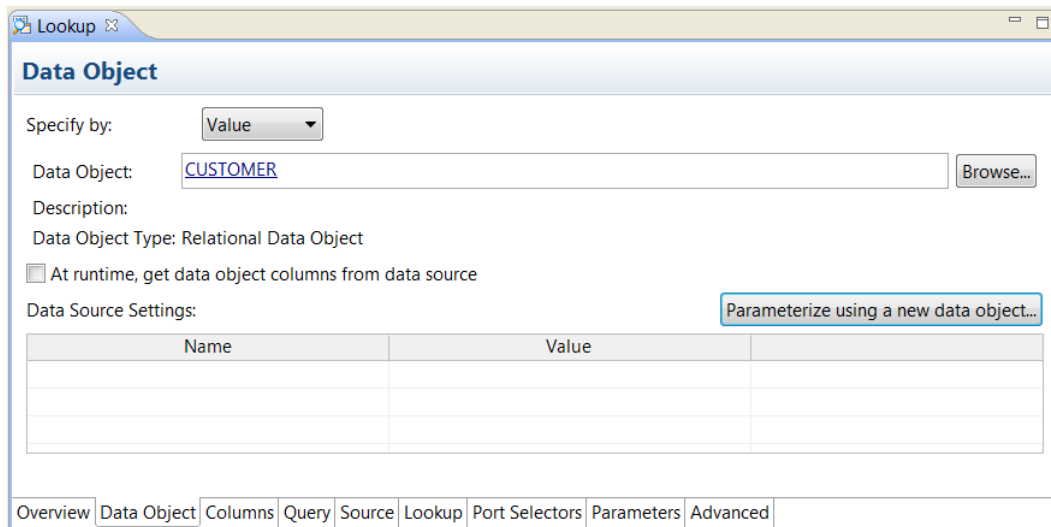
Default value

The default value of the resource parameter that you configured for the data object. The default value is a physical data object name and the path to the object in the Model repository. Read-only.

Port Name Conflicts with the Lookup Port

When you parameterize a lookup source, a Lookup transformation input port might have a name conflict with a port in the lookup source.

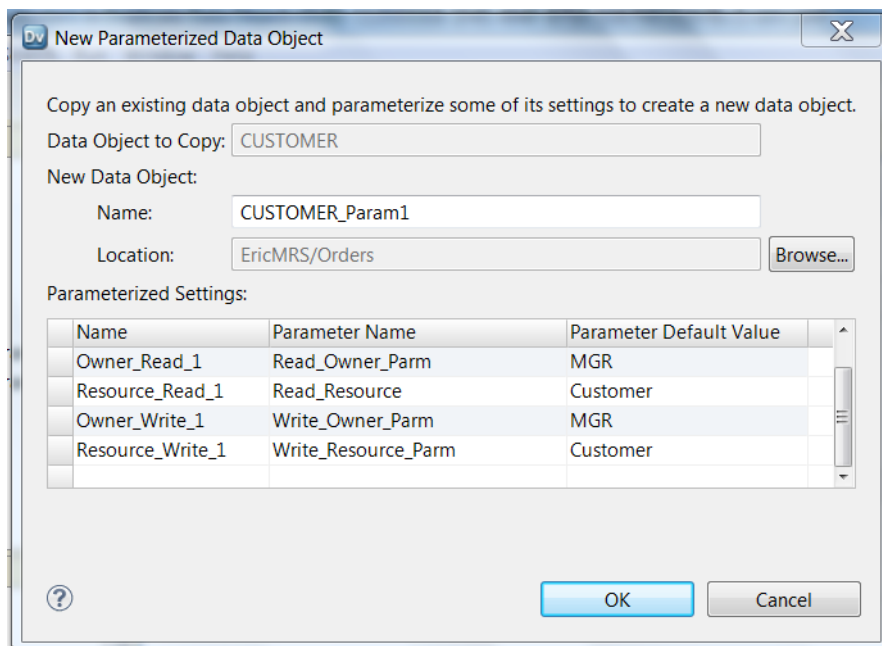
When a Lookup transformation input port has a name conflict with the lookup port in the lookup source, the Developer tool does not rename either of the ports. The Developer tool displays a validation error. You must change the input port name in the Lookup transformation or remove the port from the transformation.



When you parameterize a data object by creating a duplicate data object, the Developer tool creates set of parameters for the data object. The Developer tool creates different parameters based on whether the data object is a flat file or a relational data object.

When you create the duplicate data object, configure the default parameter values on the **New Parameterized Data Object** dialog box.

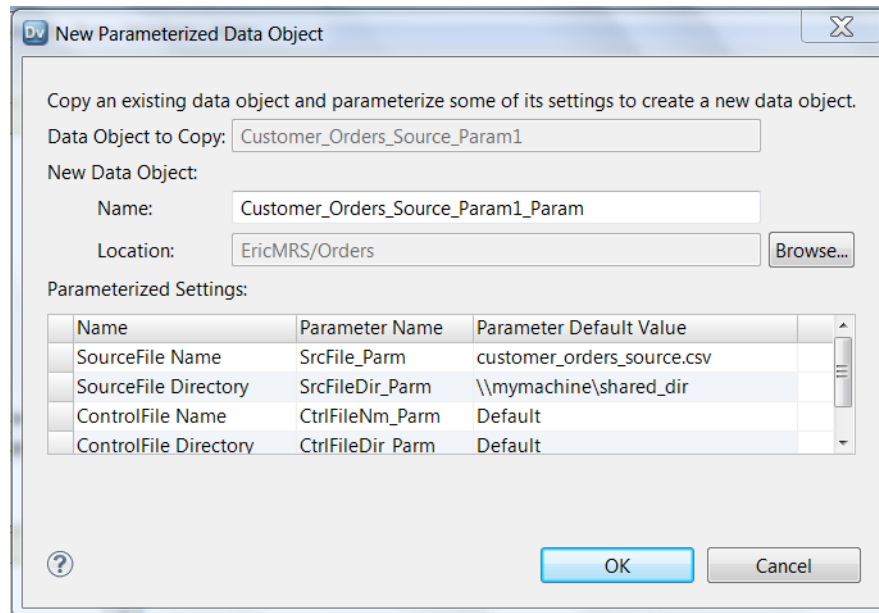
The following image shows the **New Parameterized Data Object** for a relational data object:



You can change the name of the data object. Enter the parameter default values for the owner and the resource parameters.

If the original data object is parameterized, the Developer tool copies the parameters from the original data object to the duplicate data object. If an original property is not parameterized, the Developer tool creates a parameter for it in the duplicate data object. The Developer tool uses the original property value as the default parameter value in the duplicate data object. When the Developer tool cannot determine an original property value, the Developer tool creates a parameter with a default value based on the parameter type.

The following image shows the **New Parameterized Data Object** dialog box for a flat file data object:



Configure default parameter values for the source file and the source file directory. If the flat file has a control file, configure the control file name and directory.

After you configure the default values, the Developer tool creates the duplicate data object. The duplicate data object name appears on the **Data Object** tab of the Lookup transformation. The duplicate data object appears in the **Object Navigator**.

To change the parameter values for the data object after you create it, open the physical data object in the **Object Navigator**. Click the **Parameters** tab.

Port Selectors

You can create a lookup condition when the Lookup transformation contains generated ports. You can reference a dynamic port or a port selector in a lookup condition. You can also use an expression parameter to parameterize the complete lookup expression.

When the dynamic port contains multiple generated ports, you can define a port selector to filter the generated ports in the lookup condition. The lookup source might change in a dynamic mapping. You can configure a port selector to filter which ports to use for the lookup column. The lookup source port selector must contain the same number of ports as the input columns port selector.

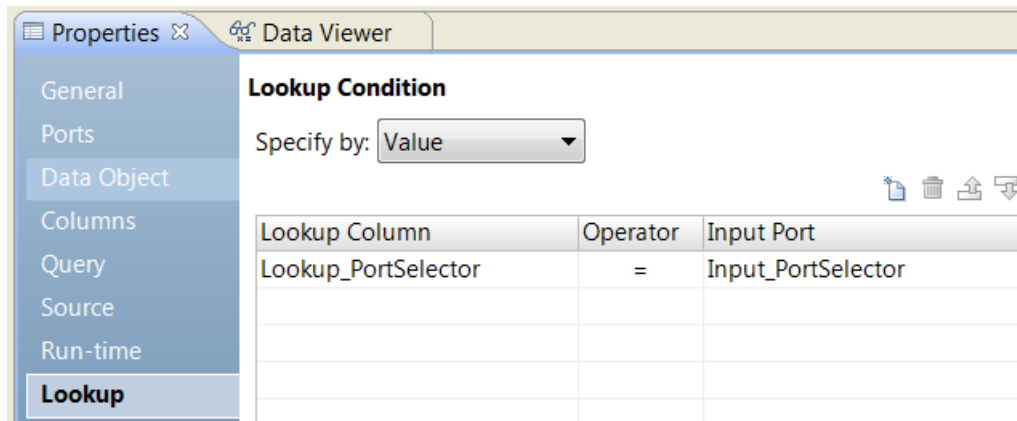
For example, `Lookup_PortSelector` contains the following ports:

```
C_CustKey
C_OrderKey
```

`Input_PortSelector` contains the following input ports:

```
CustomerID_IN
OrderID_IN
```

The following image shows a lookup condition that contains port selectors:



The lookup condition expands to the following expression:

```
C_CustKey = CustomerID_IN AND C_OrderKey = OrderID_IN
```

When the lookup condition contains multiple ports, you can configure one operator. For example, you can change the operator to greater than (>). The lookup condition expands to the following expression:

```
C_CustKey > CustomerID_IN AND C_OrderKey > OrderID_IN
```

You can create a lookup condition that contains a dynamic port:

```
Lookup_PortSelector = Dynamic_Input_Port
```

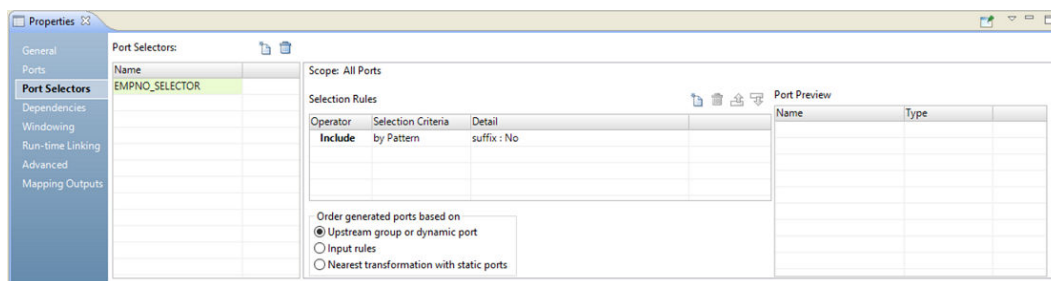
The dynamic port must contain the same number of ports as the port selector.

Port Selector Configuration

When you configure a port selector, you define selection rules to determine which generated ports to include. The selection rules are similar to the input rules that you can configure for dynamic ports.

A port selector can include static or generated ports. Configure a port selector on the **Port Selector** tab.

The following image shows the **Port Selector** tab:



Configure the following properties for a port selector:

Name

Identifies the port selector. You can create multiple port selectors in a transformation and reference them in expressions.

Scope

Identifies a group of ports that the port selector applies to. You must choose the scope when you create a port selector for a Joiner or a Lookup transformation. These transformations have multiple input groups. The Joiner transformation has a Master or a Detail scope. The Lookup transformation has an

Import or a Lookup scope. The Expression transformation has one input group. The scope is always All Ports.

Selection Rules

Determines the ports to include in the port selector. When you create the selection rules, the **Port Preview** panel shows the ports that qualify from the current input ports. These ports might change. Configure the selection rules to accommodate ports from different sources.

Selection Rules

The selection rules associated with a port selector determine the ports to include in the port selector.

When you create the selection rules, the **Port Preview** panel shows the ports that qualify from the current input ports. These ports might change. Configure the selection rules to accommodate ports from different sources.

Create selection rules based on the following criteria:

Operator

Includes or excludes the ports that selection rules return. Default is include. You must include ports before you can exclude ports.

Selection Criteria

The type of selection rule you want to create. You can create a rule based on the column name, port type, pattern, or complex data type definition. To include ports based on the column name, search for specific names or search for a pattern of characters in the name.

Detail

The values to apply to the selection criteria. If the selection criteria is by column name, configure the string or name to search for. If the selection criteria is by port type, select the port types to include.

The following table describes the selection criteria and how to specify the details for the criteria:

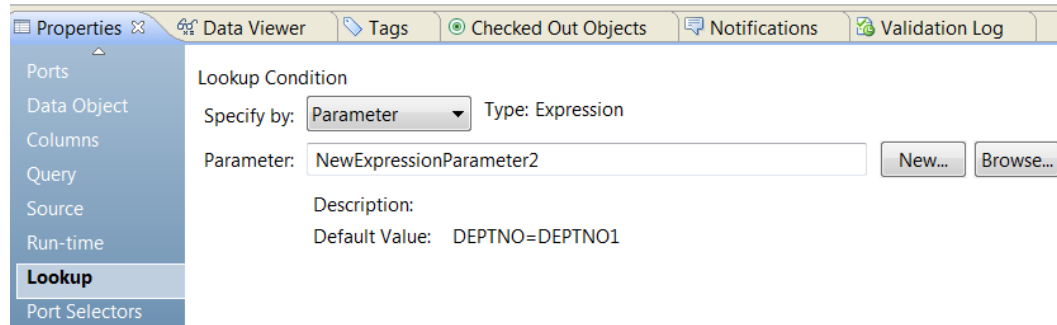
Selection Criteria	Description	Detail
All	Includes all ports.	No details required.
Name	Filters ports based on the port name.	Select the port names from a list of values or use a parameter of type Port or Port List.
Type	Filters ports based on the data type of each port.	Select data types from a list.
Pattern	Filters ports by a string of characters in the name or by a regular expression.	Choose prefix, suffix, or regular expression as the pattern type for the port name. Then, enter a value for the pattern or use a parameter of type String.
Complex Data Type Definition	Filters ports by a complex data type definition.	Choose prefix, suffix, or regular expression as the pattern type for the complex data type definition. Then, enter a value for the pattern or use a parameter of type String.

Parameterize the Lookup Condition

You can configure an expression parameter that defines the lookup condition. An expression parameter contains a complete expression that you create in an expression editor. You can define a mapping parameter to override the expression parameter at run time.

If you specify a lookup condition using a parameter, you can browse for an expression parameter or you can create a parameter.

The following image shows where to configure an expression parameter for the Lookup condition:



To create a parameter, click **New**. Define a name for the parameter and edit the default value. The expression parameter default value is the full expression to define the lookup condition. You can use generated ports, dynamic ports, and port selectors in the expression.

Note: When you create the expression, the lookup column is always the first value and the input column is the second value.

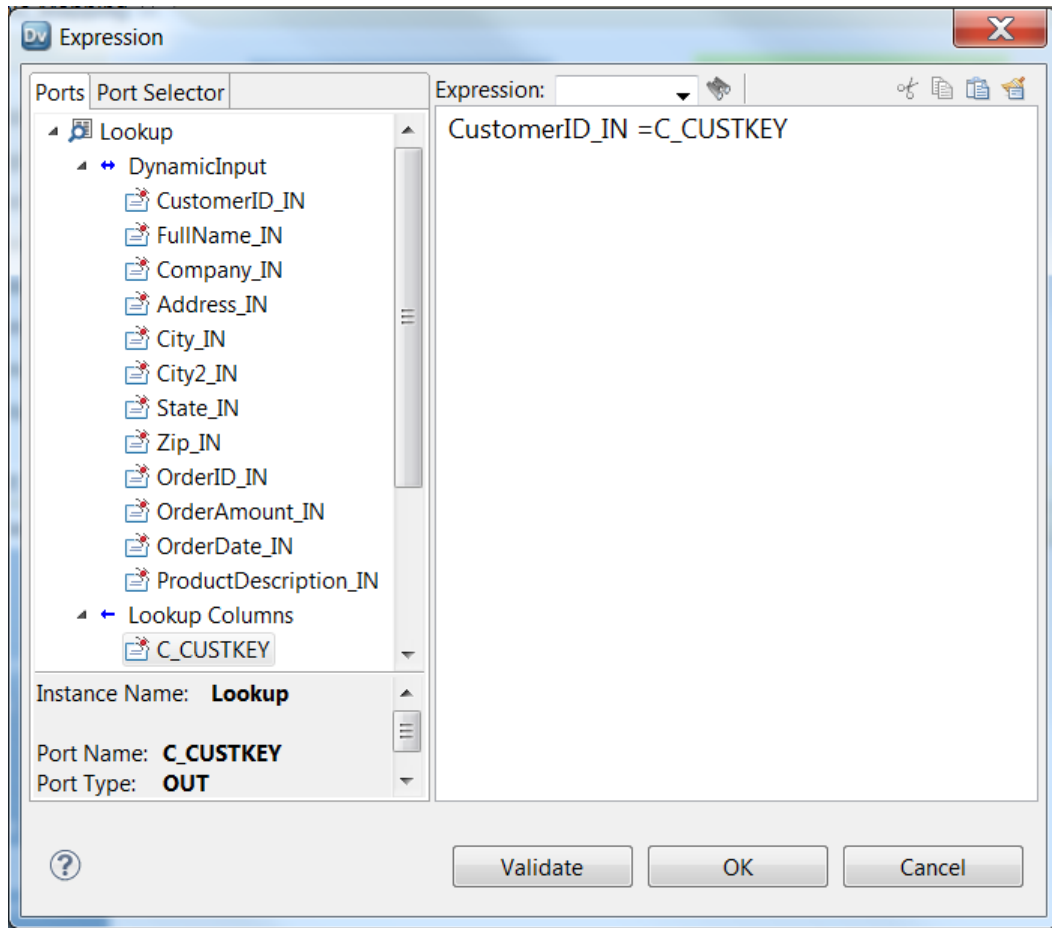
For example, you create the following lookup condition in an expression parameter:

```
CustomerID_IN = C_CUSTKEY
```

CustomerID_IN is the lookup column.

C_CUSTKEY is the input column.

The following image shows the lookup expression in the expression editor:



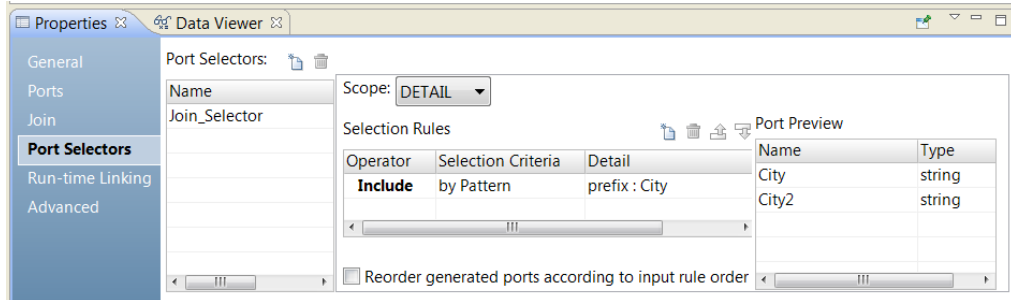
Creating a Port Selector

Create a port selector to determine which ports to use in a dynamic expression, a lookup condition, or a joiner condition.

1. Click the **Port Selectors** tab.
2. In the **Port Selectors** area, click **New**.
The Developer tool creates a port selector with a default selection rule that includes all ports.
3. In the **Port Selectors** area, change the port selector name to a unique name.
4. If you are working on the Joiner transformation or the Lookup transformation, choose the scope.
The available ports change based on the group of ports that you choose.
5. In the **Selection Rules** area, select an **Operator**.
 - Include. Create a rule that includes ports for the port selector. You must include ports before you can exclude ports.
 - Exclude. Create a rule that excludes specific ports from the port selector.
6. Choose the **Selection Criteria**.
 - By Name. Select specific ports by name. You can select the port names from a list of ports in the scope.

- By Type. Select ports by type. You can select one or more data types.
- By Pattern. Select ports by a pattern of characters in the port name. You can search with specific characters or you can create a regular expression.

The following image shows the Port Selector tab:



7. Click the **Detail** column.

The **Input Rule Detail** dialog box appears.

8. Select the values to filter ports by.

- By Name. Choose to create a port list by value or by a parameter. Click **Choose** to select the ports in the list.
- By Type. Select one or more data types from a list. The **Port Preview** area shows ports of the types that you select.
- By Pattern. Choose to search the prefix or suffix of the port name for a specific pattern of characters. Or, choose to create a regular expression to search with. Configure a parameter or configure the pattern to search with.

The **Port Preview** area shows the ports in the port selector as you configure the rules.

9. To reorder the ports in the port selector, select **Reorder generated ports according to the input rule order**.

Run-time Properties

Set the run-time properties to enable and configure lookup caching. You must add the Lookup transformation to a mapping before you can configure run-time lookup properties.

The following table describes run-time properties for Lookup transformations that perform flat file, reference table, or relational lookups:

Property	Description
Lookup Caching Enabled	<p>Indicates whether the Integration Service caches lookup values.</p> <p>When you enable lookup caching, the Integration Service queries the lookup source once, caches the values, and looks up values in the cache. Caching the lookup values can increase performance on large lookup tables.</p> <p>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.</p> <p>The Integration Service always caches flat file lookups.</p>
Lookup Data Cache Size	<p>Amount of memory that the Data Integration Service allocates to the data cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.</p>
Lookup Index Cache Size	<p>Amount of memory that the Data Integration Service allocates to the index cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.</p>
Cache File Name Prefix	<p>Prefix for the cache file. You can specify the cache file name prefix for a persistent lookup cache.</p>
Pre-build Lookup Cache	<p>Allows the Integration Service to build the lookup cache before the Lookup transformation receives the data. The Integration Service can build multiple lookup cache files at the same time to increase performance.</p> <p>Configure one of the following options:</p> <ul style="list-style-type: none">- Auto. The Integration Service determines the value.- Always Allowed. Allows the Integration Service to build the lookup cache before the Lookup transformation receives the data. The Integration Service can build multiple lookup cache files at the same time to increase performance.- Always disallowed. The Integration Service cannot build the lookup cache before the Lookup transformation receives the first row.
Lookup Cache Directory Name	<p>Directory used to build the lookup cache files when you configure the Lookup transformation to cache the lookup source.</p> <p>Default is the CacheDir system parameter. You can configure another system parameter or user-defined parameter for this property.</p>
Re-cache from Lookup Source	<p>Rebuilds the lookup cache to synchronize the persistent cache with the lookup table. Recache the lookup from the database when the Lookup transformation has a persistent lookup cache and the lookup table changes occasionally.</p>

RELATED TOPICS:

- [“Cache Size” on page 71](#)

Advanced Properties

Configure the persistent lookup cache and connection to a relational database in the advanced properties. The properties that appear are based on the type of lookup source.

The following table describes advanced properties for each type of lookup source:

Property	Lookup Source Type	Description
Lookup Cache Persistent	Flat file, reference table, relational lookup	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.
Case Sensitive String Comparison	Flat file	The Integration Service uses case-sensitive string comparisons when performing lookups on string columns.
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. For relational lookups, null ordering is based on the database default value.
Sorted Input	Flat file	Determines whether the data in source is already sorted on Lookup Transformation key and data ports. When set to TRUE, Informatica skips sorting of data while building a cache for Lookup results. Default is false.
Tracing Level	Flat file, logical data object, reference table, relational lookup	Sets the amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Lookup policy on multiple match	Flat file	Determines the result when the lookup query finds multiple matches. Choose one of the following methods: <ul style="list-style-type: none">- Return First Row- Return Last Row- Report Error- Return Any Row- Return All Rows Default is Return Any Row.
Ignore null values that match	Flat file	Ignore null values that match when performing lookups.
Dynamic lookup cache	Flat file	Update lookup cache with unfound or new lookup values.

Property	Lookup Source Type	Description
Update Else Insert	Reference table, relational lookup	Applies to dynamic lookup cache only. The Integration Service updates the row in the cache if the row type entering the Lookup transformation is update, 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.
Insert Else Update	Reference table, relational lookup	Applies to dynamic lookup cache only. The Integration Service inserts the row in the cache if the row type entering the Lookup transformation is insert and 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.
Output Old Value on Update	Reference table, relational lookup	The Integration Service outputs the value that existed in the cache before it updated the row. Otherwise, the Integration Service outputs the updated value that it writes in the cache.
Update Dynamic Cache Condition	Reference table, relational lookup	Applies to dynamic lookup cache only. An expression that indicates whether to update the dynamic cache. The Integration Service updates the cache when the condition is true and the data exists in the cache. Default is true.
Connection	Reference table, relational lookup	Connection to the relational database that contains the relational lookup source. You can use a parameter for the connection. For reference table lookups, this field is read-only.
Sorted Input	Flat file	Indicates that input data is presorted by groups.
Datetime Format	Flat file	Define a datetime format and field width. Milliseconds, microseconds, or nanoseconds formats have a field width of 29. If you do not select a datetime format for a port, you can enter any datetime format. Default is YYYY-MM-DD HH24:MI:SS. The Datetime format does not change the size of the port. This field is read-only.
Thousand Separator	Flat file	Value is None. This field is read-only.
Decimal Separator	Flat file	Value is a period. This field is read-only.

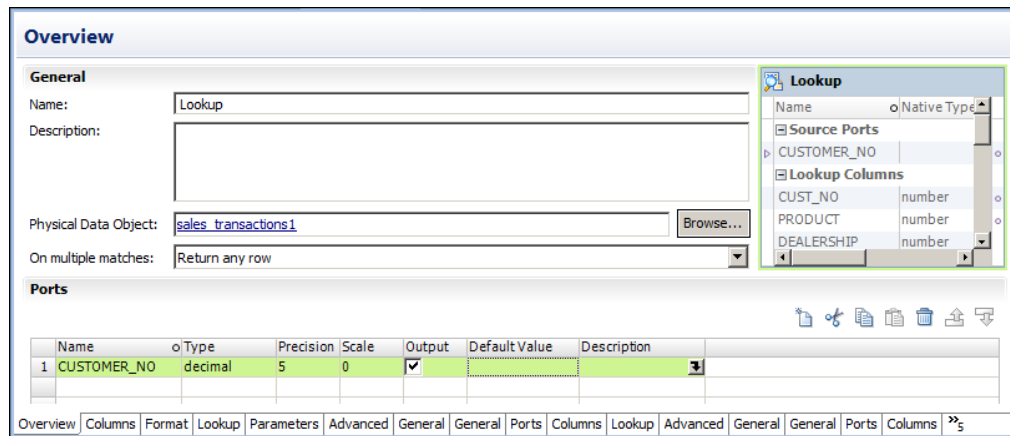
Creating a Reusable Lookup Transformation

Create a Lookup transformation to look up data in a flat file, logical data object, reference table, or relational data object.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
3. Browse to the Lookup wizard.
4. Select **Flat File Data Object Lookup**, **Logical Data Object Lookup**, **Reference Table Lookup**, or **Relational Data Object Lookup**.

5. Click **Next**.
The **New Lookup Transformation** dialog box appears.
6. Select a physical data object or reference table in the Developer tool.
7. Enter a name for the transformation.
8. In the **On multiple matches** dropdown, choose the strategy to use when the Lookup transformation finds multiple rows that match the lookup condition.
9. Click **Finish**.
The Lookup transformation appears in the editor.
10. In the **Ports** section of the **Overview** view, add output ports to the transformation.

The following figure shows the CUSTOMER_NO output port in a Lookup transformation:



11. On the **Run-time** tab of the **Properties** view, select **Lookup Caching Enabled** to enable lookup caching.
Note: You must add the Lookup transformation to a mapping before you can configure run-time lookup properties.
12. On the **Lookup** tab of the **Properties** view, add one or more lookup conditions.
13. On the **Advanced** tab of the **Properties** view, configure the tracing level, dynamic lookup cache properties, and run-time connection.
14. Save the transformation.

Creating a Non-Reusable Lookup Transformation

Create a non-reusable Lookup transformation in a mapping or maplet.

1. In a mapping or maplet, drag a Lookup transformation from the Transformation palette to the editor.
The **New** dialog box appears.
2. Select **Flat File Data Object Lookup**, **Logical Data Object Lookup**, **Reference Table Lookup**, or **Relational Data Object Lookup**.
3. Click **Next**.
The **New Lookup Transformation** dialog box appears.
4. Select a physical data object or reference table in the Developer tool.

5. Enter a name for the transformation.
6. In the **On multiple matches** dropdown, choose the strategy to use when the Lookup transformation finds multiple rows that match the lookup condition.
7. Click **Finish**.

The Lookup transformation appears in the editor.

8. Select the Lookup transformation in the editor.

The toolbar appears above the transformation.

9. On the **Ports** tab of the **Properties** view, add output ports to the transformation.

The following figure shows the CUSTOMER_NO output port in a Lookup transformation:

Properties							
Data Viewer Tags Validation Log Navigator							
General							
Ports							
	Name	Type	Precision	Scale	Output	Default Value	Description
1	CUSTOMER_NO	decimal	5	0	<input checked="" type="checkbox"/>		

10. On the **Run-time** tab of the **Properties** view, select **Lookup Caching Enabled** to enable lookup caching.
Note: You must add the Lookup transformation to a mapping before you can configure run-time lookup properties.
11. On the **Lookup** tab of the **Properties** view, add one or more lookup conditions.
12. On the **Advanced** tab of the **Properties** view, configure the tracing level, dynamic lookup cache properties, and run-time connection.
13. Save the transformation.

Creating an Unconnected Lookup Transformation

Create an unconnected Lookup transformation when you want to perform a lookup from an expression. You can create a reusable or a non-reusable unconnected Lookup transformation on a flat file, reference table, or relational data object.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
3. Browse to the Lookup wizard.
4. Select **Flat File Data Object Lookup**, **Reference Table Lookup**, or **Relational Data Object Lookup**.
5. Click **Next**.
The **New Lookup** dialog box appears.
6. Select a physical data object or reference table in the Developer tool.
7. Enter a name for the transformation.

8. In the **On multiple matches** dropdown, choose the strategy to use when the Lookup transformation finds multiple rows that match the lookup condition.
Note: Do not choose **Return All** for an unconnected lookup.
9. Click **Finish**.
The Lookup transformation appears in the editor.
10. In the **Ports** section in the **Overview** view, add ports to the transformation.
Create an input port for each argument in the :LKP expression. Create an input port for each lookup condition that you create. You can use an input port in multiple conditions.
11. In the **Ports** section in the **Overview** view, configure one port as the return port.
12. In the **Lookup** view, add one or more lookup conditions to compare the transformation input values with values in the lookup source or cache.
When the condition is true, the lookup returns a value in the return port. If the lookup condition is false, the lookup returns NULL.
13. Create a :LKP expression for a port in a transformation that allows expressions, such as an Aggregator transformation, Expression transformation, or Update Strategy transformation.
14. When you create a mapping, add the unconnected Lookup transformation to the mapping in the editor, but do not connect the ports to the other transformations in the mapping.

Unconnected Lookup Example

A retail store in California adds a state sales tax to each price for items that it sells to customers within the state. The amount of tax is based on what county that the customer resides in. To retrieve the sales tax, you create a Lookup transformation that receives a county name and then returns a sales tax amount for the county. If the county does not charge a sales tax, the Lookup transformation returns NULL. Call the lookup from an Expression transformation.

Complete the following steps to configure an unconnected lookup of sales tax by county:

1. Import a flat file physical data object that contains the sales tax amounts by county.
2. Create the unconnected Lookup transformation.
3. Add input ports to the Lookup transformation.
4. Define the return port.
5. Create the lookup condition.
6. Call the lookup from an Expression transformation.

Step 1. Import the sales tax lookup source to the Model repository.

The sales tax file must be in the Model repository before you create the Lookup transformation. For this scenario, the sales tax file contains two fields, Sales_County and County_SalesTax. The county is a string that contains a county name. County_SalesTax is a decimal field that contains a tax rate for the county. The sales tax file is the lookup source.

Step 2. Create the Unconnected Lookup Transformation

Create a reusable flat file Lookup transformation with the sales tax flat file data object. For this scenario, the transformation name is Sales_Tax_Lookup. Select **Return First Row** on multiple matches.

Step 3. Define the Lookup Transformation Ports

Define the Lookup transformation ports on the **Ports** tab of the **Properties** view.

Port Type	Name	Type	Length	Scale
Input	In_County	String	25	
Output	SalesTax	Decimal	3	3

Step 4. Configure the Lookup Transformation Return Port

The return port is the field in the flat file that the lookup retrieves. On the **Columns** tab, the County_SalesTax column is the return port.

When the lookup is true, the Integration Service finds the county in the flat file source. The Integration Service returns a sales tax value in the return port. If the Integration Service does not find the county, the lookup result is false and the Integration Service returns NULL in the return port.

Step 5. Define the Lookup Condition

On the **Lookup** view, define the lookup condition to compare the input value with values in the lookup source.

To add the lookup condition, click the **Lookup Column**.

The lookup condition has the following syntax:

```
SALES_COUNTY = IN_COUNTY
```

Step 6. Create an Expression Transformation

Create an Expression transformation that receives sales records from a flat file. The Expression transformation receives a customer number, sales amount, and the county of the sale. It returns the customer number, sales amount, and a sales tax.

The Expression transformation has the following ports:

Port Type	Name	Type	Length	Precision	Default Value
Input	County	String	25	10	
Pass-through	Customer	String	10		
Pass-through	SalesAmt	Decimal	10	2	
Output	SalesTax	Decimal	10	2	0

The SalesTax port contains a :LKP expression. The expression calls the Sales_Tax_Lookup transformation and passes the county name as the parameter. The Sales_Tax_Lookup transformation returns the sales tax rate to the expression. The Expression transformation multiplies the tax rate by the sales amount.

Enter the following expression for the SalesTax port:

```
(:LKP.Sales_Tax_Lookup(County) * SalesAmt)
```

The SalesTax port contains the expression result. If the lookup fails, the Lookup transformation returns NULL and the SalesTax port contains null values.

You can add an expression to check for null values in the SalesTax port. If SalesTax is NULL you can configure the SalesTax port to return zero. Add the following text to the lookup expression to check for null values and return zero:

```
IIF(ISNULL(:LKP.Sales_Tax_Lookup(County) * SalesAmt),0, SalesTax)
```

Lookup Transformation in a Non-native Environment

The Lookup transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Lookup Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Multiple Matches

The Lookup transformation finds values based on the condition you configure in the transformation. Choose how to handle multiple matches in the lookup source.

Return First Row; Return Last Row

The transformation returns the first matching result or the last matching result.

The Data Integration Service orders results to identify the first and last row. The following rules determine result order:

- Ordering depends on the lookup ports present in the lookup condition and lookup output ports.
- Numerical value sorting is by ascending order.
- String value sorting is lexicographical.
- Date value sorting is by earliest date first.
- When mappings contain a non-equijoin(<=,>=,<,>,!<=) comparison and some rows contain NULL values, results might be different depending on the run-time engine:
 - On the Spark engine, NULL results are treated as TRUE. For more information about how Spark processes null values, see the Apache documentation: <https://spark.apache.org/docs/3.0.0-preview/sql-ref-null-semantics.html#comp-operators>
 - On the Blaze or native engines, NULL results are treated as FALSE.

Return Any Row

The transformation returns any of the rows that match the lookup condition. The transformation creates an index based on the key ports instead of all Lookup transformation ports. When you choose this option, performance can improve because the process of indexing rows is simpler.

Return All Rows

The Lookup transformation returns all rows that match the lookup condition.

Report Error

When the Lookup transformation uses a static cache or no cache, the Data Integration Service marks the row as an error. The Lookup transformation writes the row to the session log by default, and increases the error count by one.

When the Lookup transformation has a dynamic cache, the Data Integration Service fails the session when it encounters multiple matches. The session fails while the Data Integration Service is caching the lookup table or looking up the duplicate key values.

Also, if you configure the Lookup transformation to output old values on updates, the Lookup transformation returns an error when it encounters multiple matches. The transformation creates an index based on the key ports instead of all Lookup transformation ports.

Rules and Guidelines

Mapping validation fails in the following situations:

- Case sensitivity is disabled.
- The lookup condition contains binary data type.
- The cache is configured to be shared, named, persistent, dynamic, or uncached. The cache must be a static cache.
- The lookup condition uses a field with a complex data type.

If you add a data object that uses Sqoop as a Lookup transformation in a mapping, the Data Integration Service does not run the mapping through Sqoop. It runs the mapping through JDBC.

Lookup Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Complex Data Types

You can use a port with a complex data type as the input port or lookup port in the lookup condition. To use a port with a complex data type in the lookup condition, specify the condition by parameter.

Multiple Matches

The Lookup transformation finds values based on the condition you configure in the transformation. Choose how to handle multiple matches in the lookup source.

Return First Row; Return Last Row

The transformation returns the first matching result or the last matching result.

The Data Integration Service orders results to identify the first and last row. The following rules determine result order:

- Ordering depends on the lookup ports present in the lookup condition and lookup output ports.
- Numerical value sorting is by ascending order.
- String value sorting is lexicographical.
- Date value sorting is by earliest date first.
- When mappings contain a non-equi-join (\leq , \geq , $<$, $>$, \neq) comparison and some rows contain NULL values, results might be different depending on the run-time engine:
 - On the Spark engine, NULL results are treated as TRUE. For more information about how Spark processes null values, see the Apache documentation: <https://spark.apache.org/docs/3.0.0-preview/sql-ref-null-semantics.html#comp-operators>
 - On the Blaze or native engines, NULL results are treated as FALSE.

Return Any Row

The transformation returns any of the rows that match the lookup condition. The transformation creates an index based on the key ports instead of all Lookup transformation ports. When you choose this option, performance can improve because the process of indexing rows is simpler.

Return All Rows

The Lookup transformation returns all rows that match the lookup condition.

Report Error

When the Lookup transformation uses a static cache or no cache, the Data Integration Service marks the row as an error. The Lookup transformation writes the row to the session log by default, and increases the error count by one.

When the Lookup transformation has a dynamic cache, the Data Integration Service fails the session when it encounters multiple matches. The session fails while the Data Integration Service is caching the lookup table or looking up the duplicate key values.

Also, if you configure the Lookup transformation to output old values on updates, the Lookup transformation returns an error when it encounters multiple matches. The transformation creates an index based on the key ports instead of all Lookup transformation ports.

Rules and Guidelines

Mapping validation fails in the following situations:

- Case sensitivity is disabled.
- The lookup condition contains binary data type.
- The cache is configured to be shared, named, persistent, dynamic, or uncached. The cache must be a static cache.

The mapping fails in the following situation:

- The transformation is unconnected and used with a Joiner or Java transformation.

Note: If an HBase lookup does not result in a match, it generates a row with null values for all columns. You can add a Filter transformation after the Lookup transformation to filter out null rows.

Lookup Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Multiple Matches

When you choose to return the first, last, or any value on multiple matches, the Lookup transformation returns any value.

If you configure the transformation to report an error on multiple matches, the Databricks Spark engine drops the duplicate rows and does not include the rows in the logs.

Note: If an HBase lookup does not result in a match, it generates a row with null values for all columns. You can add a Filter transformation after the Lookup transformation to filter out null rows.

Mapping Validation

Mapping validation fails in the following situations:

- Case sensitivity is disabled.
- The lookup condition contains binary data type.
- The cache is configured to be shared, named, persistent, dynamic, or uncached. The cache must be a static cache.
- The lookup condition uses a field with a complex data type.
- The lookup source is not Microsoft Azure SQL Data Warehouse.

Note: If an HBase lookup does not result in a match, it generates a row with null values for all columns. You can add a Filter transformation after the Lookup transformation to filter out null rows.

Lookup Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- The lookup is a data object.
- An Aggregator transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.
- A Rank transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.
- A pipeline contains more than one passive Lookup transformation configured with an inequality condition.

The mapping fails in the following situations:

- The transformation is unconnected.

General Guidelines

Consider the following general guidelines:

- Using a float data type to look up data can return unexpected results.
- Use a Lookup transformation to look up data in a flat file, HDFS, Hive, relational, JDBC V2, and HBase data.
- To avoid cross join of DataFrames, configure the Lookup transformation to ignore null values that match.

HBase Lookups

To use a Lookup transformation on uncached HBase tables, perform the following steps:

1. Create an HBase data object. When you add an HBase table as the resource for a HBase data object, include the ROW ID column.
2. Create a HBase read data operation and import it into the streaming mapping.
3. When you import the data operation to the mapping, select the **Lookup** option.
4. On the Lookup tab, configure the following options:
 - Lookup column. Specify an equality condition on ROW ID
 - Operator. Specify =
5. Verify that format for any date value in the HBase tables is of a valid Java date format. Specify this format in the **Date Time Format** property of the **Advanced Properties** tab of the data object read operation.

Note: If an HBase lookup does not result in a match, it generates a row with null values for all columns. You can add a Filter transformation after the Lookup transformation to filter out null rows.

Mapping validation fails in the following situations:

- The condition does not contain a ROW ID.
- The transformation contains an inequality condition.
- The transformation contains multiple conditions.
- An input column is of a date type.

JDBC V2 Lookups

You can add a JDBC V2 data object read operation as a lookup in a streaming mapping. You can run streaming mappings with JDBC V2 lookup in Azure Databricks service in Microsoft Azure cloud services.

CHAPTER 26

Lookup Caches

This chapter includes the following topics:

- [Lookup Caches Overview, 417](#)
- [Lookup Cache Types, 418](#)
- [Uncached Lookup, 419](#)
- [Static Lookup Cache, 419](#)
- [Persistent Lookup Cache, 419](#)
- [Dynamic Lookup Cache, 420](#)
- [Shared Lookup Cache, 421](#)
- [Cache Comparison, 422](#)
- [Cache Partitioning for Lookups, 422](#)

Lookup Caches Overview

You can configure a Lookup transformation to cache a relational or flat file lookup source. Enable lookup caching on a large lookup table or file to increase lookup performance.

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 caches as the source rows enter the Lookup transformation. It allocates memory for the cache based on the amount you configure in the transformation. 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.

If the data does not fit in the memory cache, the Integration Service stores the overflow values in cache files. The Integration Service creates the cache files in the cache directory. By default, the Integration Service creates cache files in the directory specified in the CacheDir system parameter. When the mapping 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 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.

When you do not configure the Lookup transformation for caching, the Integration Service queries the lookup source for each input row. Whether or not you cache the lookup source, the result of the Lookup query and processing is the same. However, you can increase lookup performance on a large lookup source if you enable lookup caching.

Lookup Cache Types

You can configure different types of lookup caches. For example, you can configure a shared cache if you want to share the cache among multiple Lookup transformations in the same mapping.

You can configure the following types of lookup caches:

Static Cache

A static cache does not change while the Integration Service processes the lookup. The Integration Service rebuilds a static cache each time it processes the lookup. By default, the Integration Service creates a static cache when you enable caching for a Lookup transformation. The Integration Service builds the cache when it processes the first lookup request. It looks up values in the cache for each row that comes into the Lookup transformation. When the lookup condition is true, the Integration Service returns a value from the lookup cache.

Use a static cache for the following reasons:

- The lookup source does not change while the mapping runs.
- The lookup is an unconnected lookup. You must use a static cache for an unconnected lookup.
- To increase performance. Because the Integration Service does not update the cache while it processes the Lookup transformation, the Integration Service processes a Lookup transformation with a static cache faster than a Lookup transformation with a dynamic cache.
- You want the Integration Service to return the default value for connected transformations or NULL for unconnected transformations when the lookup condition is false.

Persistent Cache

A persistent cache does not change each time the Integration Service processes the lookup. The Integration Service saves the lookup cache files and reuses them the next time it processes a Lookup transformation configured to use the cache. Use a persistent cache when the lookup source does not change.

You can configure the Lookup transformation to rebuild a persistent lookup cache if necessary.

Dynamic Cache

A dynamic lookup cache changes while the Integration Service processes the lookup. The Integration Service builds a dynamic lookup cache when it processes the first lookup request. When it processes each row, the Integration Service dynamically inserts or updates data in the lookup cache and passes the data to the target. The dynamic cache is synchronized with the target.

Use a dynamic cache when you want to update a target based on new and changed records. You might also use a dynamic cache when the mapping requires a lookup on target data, but connection to the target is slow.

Shared Cache

A shared cache can be used by multiple Lookup transformations in the same mapping. Use a shared cache to increase mapping performance. Instead of generating a separate lookup cache for each Lookup transformation, the Integration Service generates one cache.

Uncached Lookup

An uncached lookup is when the Integration Service does not cache the lookup source. By default, the Integration Service does not use a lookup cache for a Lookup 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 lookup cache.

When the lookup condition is true, the Integration Service returns the values from the lookup source. When the Integration Service processes a connected Lookup transformation, it returns the values represented by the lookup/output ports. When it processes an unconnected Lookup transformation, 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. When the Integration Service processes a connected Lookup transformation, it returns the default value of the output port when the condition is not met. When it processes an unconnected Lookup transformation, the Integration Service returns NULL when the condition is not met.

Static Lookup Cache

A static lookup cache is a cache that the Integration Service does not update when it processes the Lookup transformation. 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.

When the lookup condition is true, the Integration Service returns the values from the static lookup cache. When the Integration Service processes a connected Lookup transformation, it returns the values represented by the lookup/output ports. When the Integration Service processes an unconnected Lookup transformation, it returns the value represented by the return port.

When the condition is not true, the Integration Service returns either NULL or default values. When the Integration Service processes a connected Lookup transformation, the Integration Service returns the default value of the output port when the condition is not met. When the Integration Service processes an unconnected Lookup transformation, it returns NULL when the condition is not met.

Persistent Lookup Cache

A persistent lookup cache is a cache that the Integration Service reuses for multiple runs of the same mapping. Use a persistent lookup cache when the lookup source does not change between mapping runs.

By default, the Integration Service uses a non-persistent cache when you enable lookup caching in a Lookup transformation. The Integration Service deletes the cache files when the mapping completes. The next time you run the mapping, the Integration Service builds the memory cache from the lookup source.

If you configure the Lookup transformation to use a persistent lookup cache, the Integration Service saves and reuses cache files for multiple runs of the mapping. A persistent cache eliminates the time required to read the lookup table and rebuild the lookup cache.

The first time the Integration Service runs a mapping with a persistent lookup cache, the Integration Service saves the cache files to disk. The next time the Integration Service runs the mapping, it builds the memory cache from the cache files.

You can configure the Integration Service to rebuild a persistent lookup cache if the original lookup source changes. When you rebuild a cache, the Integration Service creates new cache files and writes a message to the Integration Service log.

Rebuilding a Persistent Lookup Cache

You can configure the Integration Service to rebuild the persistent lookup cache. In some cases, the Integration Service will rebuild the persistent lookup cache even if you do not configure it to.

Consider the following rules and guidelines when you rebuild a persistent lookup cache:

- Rebuild a persistent lookup cache if the lookup source changed since the last time the Integration Service built the cache.
- You can rebuild the cache when the mapping contains one or more Lookup transformations that share a cache.
- If the lookup table does not change between mapping runs, configure the Lookup transformation to use a persistent lookup cache. The Integration Service saves and reuses cache files, eliminating the time required to read the lookup table.
- If you configure subsequent Lookup transformations to rebuild the lookup cache, the Integration Service shares the cache instead of rebuilding the cache when it processes the subsequent Lookup transformation.
- If a mapping contains two persistent lookups and you configure the second Lookup transformation to rebuild the cache, then the Integration Service rebuilds the persistent lookup cache for both.

The Integration Service rebuilds the persistent lookup cache in the following scenarios:

- The Integration Service cannot find the cache files.
- The Integration Service cannot reuse the cache. In this instance, it either rebuilds the lookup cache or fails the mapping.
- You enable or disable high precision for the Integration Service.
- You edit the Lookup transformation or the mapping.
Note: The Integration Service does not rebuild the cache if you edit the transformation description.
- You change the number of partitions.
- You change the database connection or the file location used to access the lookup source.
- You change the sort order in Unicode mode.
- You change the Integration Service code page.

Dynamic Lookup Cache

A dynamic cache is a cache that the Integration Service updates when it processes each row. Use a dynamic lookup cache to keep the cache synchronized with the target.

You can use a dynamic cache with a relational lookup and flat file lookup. 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 Lookup transformation. The Integration Service updates the lookup cache when it processes each row.

Based on the results of the lookup query, the row type, and the Lookup transformation properties, the Integration Service either inserts or updates rows in the cache, or makes no change to the cache.

Shared Lookup Cache

A shared lookup cache is a static lookup cache that is shared by multiple Lookup transformations in a mapping. Use a shared lookup cache to decrease the amount of time required to build the cache.

By default, the Integration Service shares the cache for Lookup transformations in a mapping that have compatible caching structures. For example, if a mapping contains 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.

The Integration Service builds the cache when it processes the first Lookup transformation. It uses the same cache to process subsequent Lookup transformations that share the cache. When the Integration Service shares a lookup cache, it writes a message in the Integration Service log.

The Integration Service allocates data cache memory and index cache memory for the first Lookup transformation. It does not allocate additional memory for subsequent Lookup transformations that share the lookup cache.

If the transformation or the cache structure do not allow sharing, the Integration Service creates a new cache.

Rules and Guidelines for Sharing a Lookup Cache

Consider the following rules and guidelines when you share a lookup cache:

- You can share one or more static caches with a dynamic lookup. If a dynamic lookup shares the cache with a static lookup in the same mapping, the static lookup reuses the cache created by the dynamic lookup.
- You cannot share a cache between dynamic lookups.
- If you configure multiple Lookup transformations to rebuild a persistent lookup cache, the Integration Service builds the cache for the first Lookup transformation, and then shares the persistent lookup cache for the subsequent Lookup transformations.
- If you do not configure the first Lookup transformation to rebuild a persistent lookup cache, but you configure a subsequent Lookup transformation to rebuild the cache, the transformations cannot share the cache. The Integration Service builds the cache when it processes each Lookup transformation.
- The lookup/output ports for subsequent Lookup transformations must match or be a subset of the ports in the Lookup transformation that the Integration Service uses to build the cache. The order of the ports do not need to match.
- Lookup transformations that share the cache must have the following characteristics:
 - The Lookup transformations must use the same ports in the lookup condition.
 - The Lookup transformations must use the same SQL override if one is used.
 - Lookup caching must be enabled in all Lookup transformations.
 - The Lookup transformations must use the same type of lookup source.

- All relational Lookup transformations must use the same database connection.
- The Lookup transformations must use the same lookup table name.
- The structure of the cache for all Lookup transformations must be compatible.

Cache Comparison

The Integration Service performs differently based on the type of lookup cache that you configure.

The following table compares Lookup transformations with an uncached lookup, a static cache, and a dynamic cache:

Uncached	Static Cache	Dynamic Cache
The Integration Service does not insert or update the cache.	The Integration Service does not insert or update the cache.	The Integration Service can insert or update rows in the cache as it passes rows to the target.
You can use a relational lookup.	You can use a relational or flat file lookup.	You can use a relational or flat file lookup.
<p>When the condition is true, the Integration Service returns a value from the lookup table or cache.</p> <p>When the condition is not true, the Integration Service returns the default value for connected transformations and NULL for unconnected transformations.</p>	<p>When the condition is true, the Integration Service returns a value from the lookup table or cache.</p> <p>When the condition is not true, the Integration Service returns the default value for connected transformations and NULL for unconnected transformations.</p>	<p>When the condition is true, the Integration Service either updates rows in the cache or leaves the cache unchanged, based on the row type. This indicates that the row is in the cache and target table. You can pass updated rows to a target.</p> <p>When the condition is not true, the Integration Service either inserts rows into the cache or leaves the cache unchanged, based 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.</p>

Cache Partitioning for Lookups

Cache partitioning creates a separate cache for each partition that processes an Aggregator, Joiner, Rank, or Lookup transformation. Cache partitioning increases mapping performance because each partition queries a separate cache in parallel.

When the Integration Service creates partitions for a mapping, it might use cache partitioning for partitioned Lookup transformations.

The Integration Service uses cache partitioning for connected Lookup transformations when the following conditions are true:

- The lookup condition contains only equality operators.
- When the connected Lookup transformation looks up data in a relational table, and the database is configured for case-sensitive comparison.

The Integration Service does not use cache partitioning for unconnected Lookup transformations.

When the Integration Service does not use cache partitioning for a Lookup transformation, all partitions of the Lookup transformation share the same cache. Each partition queries the same cache serially.

CHAPTER 27

Dynamic Lookup Cache

This chapter includes the following topics:

- [Dynamic Lookup Cache Overview, 424](#)
- [Uses for a Dynamic Lookup Cache, 425](#)
- [Dynamic Lookup Cache Properties, 425](#)
- [Dynamic Lookup Cache and Output Values, 427](#)
- [Lookup Transformation Values, 427](#)
- [SQL Override and Dynamic Lookup Cache, 430](#)
- [Mapping Configuration for a Dynamic Lookup Cache, 430](#)
- [Conditional Dynamic Lookup Cache Updates, 433](#)
- [Dynamic Cache Update with Expression Results, 434](#)
- [Dynamic Lookup Cache Example, 435](#)
- [Rules and Guidelines for Dynamic Lookup Caches, 436](#)

Dynamic Lookup Cache Overview

Use a dynamic lookup cache to keep the cache synchronized with the target. You can use a dynamic cache with a relational lookup or flat file lookup.

The Integration Service builds the dynamic lookup 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 updates the lookup cache when it processes each row.

Based on the results of the lookup query, the row type, and the Lookup transformation properties, the Integration Service performs one of the following actions on the dynamic lookup cache when it reads a row from the source:

Inserts the row into the cache

The Integration Service inserts the row when the row is not in the cache and you configured the Lookup transformation 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 Integration Service updates the row when the row exists in the cache and you configured the Lookup transformation to update rows in the cache. The Integration Service updates the row in the cache based on the input ports. The Integration Service flags the row as an update row.

Makes no change to the cache

The Integration Service makes no change when the row exists in the cache and you configured the Lookup transformation 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.

Based on the value of the NewLookupRow, you can also configure a Router or Filter transformation with the dynamic Lookup transformation to route insert or update rows to the target table. You can route unchanged rows to another target table or flat file, or you can drop them.

Uses for a Dynamic Lookup Cache

You can configure a Lookup transformation with a dynamic lookup cache to update the cache based on changes in the lookup source.

You might use a dynamic lookup cache for the following reasons:

Update a master customer table with new and updated customer information.

For example, you can use a Lookup transformation to perform a lookup on the customer table to determine if a customer exists in the target. The cache represents the customer table. The Lookup transformation inserts and update rows in the cache as it passes rows to the target.

Use an exported flat file as a lookup source instead of a relational table.

If the connection to the database is slow, you can export the relational table contents to a flat file and use the file as a lookup source. For example, you might need to use this method if an ODBC connection to a database is slow. You can configure the database table as a relational target in the mapping and pass the lookup cache changes back to the database table.

Dynamic Lookup Cache Properties

Configure the dynamic lookup properties to enable a dynamic lookup cache and configure how the cache is updated. For example, you can configure the values that are inserted and updated in the dynamic cache.

Configure the following properties when you enable a dynamic lookup cache:

On Multiple Matches

Set to Report Error.

Dynamic Lookup Cache

Enables a dynamic lookup cache.

This option is available after you enable lookup caching.

Update Else Insert

Applies to rows entering the Lookup transformation where the row type is update. When enabled, the Integration Service updates existing rows in the cache, and inserts a row if it is new. When disabled, the Integration Service does not insert new rows.

This option is available after you enable dynamic caching.

Insert Else Update

Applies to rows entering the Lookup transformation where the row type is insert. When enabled, the Integration Service inserts rows in the cache and updates existing rows. When disabled, the Integration Service does not update existing rows.

This option is available after you enable dynamic caching.

Output Old Value on Update

The Lookup transformation can output the existing or new values from the cache. When enabled, the Integration Service outputs the existing values from the lookup/output ports before it updates the value in the cache. When the Integration Service updates a row in the cache, it outputs the value in the lookup cache before it updates the row based on the input data. When the Integration Service inserts a row in the cache, it outputs null values.

Disable the property for the Integration Service to pass the same values from the lookup/output and input/output ports. This property is enabled by default.

This option is available after you enable dynamic caching.

Update Dynamic Cache Condition

When enabled, the Integration Service uses the condition expression to determine whether to update the dynamic cache. The Integration Service updates the cache when the condition is true and the data exists in the cache.

Create the expression using lookup ports or input ports. The expression can contain input values or values in the lookup cache. Default is true.

This option is available after you enable dynamic caching.

NewLookupRow

The Developer tool adds this port to a Lookup transformation that is configured with a dynamic cache.

The NewLookupRow property can contain one of the following values:

- 0 = No update to the cache.
- 1 = Insert row into cache.
- 2 = Update row in cache.

To keep the lookup cache and the target table synchronized, pass rows to the target when the NewLookupRow value is equal to 1 or 2.

Associated Port

The Integration Service uses the value of the associated port when it updates data in the cache. The Integration Service associates input ports and lookup source ports specified in the lookup condition. You must configure associated ports for the remaining lookup source ports in the dynamic lookup. If you do not configure an associated port for all lookup source ports in a dynamic lookup, the mapping validation fails.

You can associate a lookup source port with the following objects:

Object	Description
Input port	Updates the cache based on the value of an input port.
Associated expression	Select to enter an expression. The Integration Service updates the cache based on the result of the expression.
Sequence ID	Generates a primary key for rows inserted in the lookup cache. You can associate a sequence ID with bigint and int columns only.

Ignore Null Inputs for Updates

The Developer tool 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 with a null input value.

Ignore in Comparison

The Developer tool 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 ports by default. Select this property if you want the Integration Service to ignore the port when it compares values before updating a row. Use this property to increase the performance of the comparison.

Dynamic Lookup Cache and Output Values

If you enable a dynamic lookup cache, the output port values vary based on how you configure the dynamic lookup cache. The output value of the lookup/output port depends on whether you choose to output old or new values when the Integration Service updates a row.

You can configure the **Output Old Value On Update** property to specify one of the following types of output values for a lookup/output port:

- 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 output port value.

Lookup Transformation Values

The Lookup transformation contains values for input ports, the lookup, and output ports. If you enable a dynamic lookup cache, the output port values vary based on how you configure the dynamic lookup cache.

The Lookup transformation contains the following types of values:

Input value

Value that the Integration Service passes into the Lookup transformation.

Lookup value

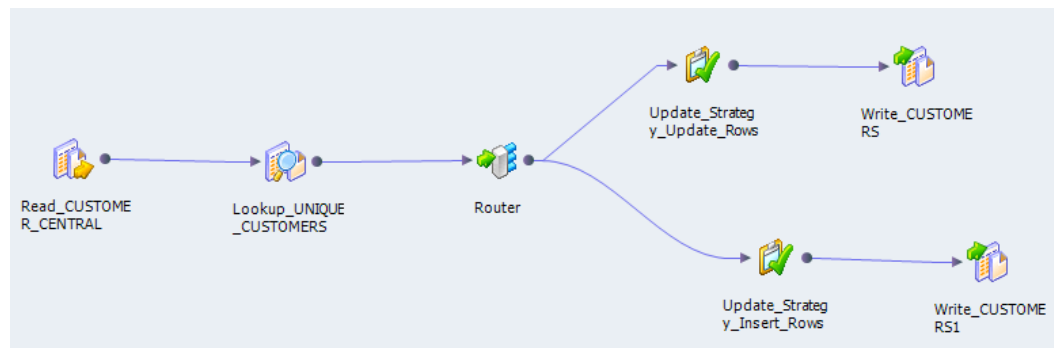
Value that the Integration Service inserts into the cache.

Output value

Value that the Integration Service passes from the output port of the Lookup transformation. The output value of the lookup/output port depends on whether you choose to output old or new values when the Integration Service updates a row.

Lookup Transformation Values Example

For example, you create a mapping with the following objects:



For the Lookup transformation, you enable dynamic lookup caching and 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.

Initial Cache Values

When you run the mapping, the Integration Service builds the lookup cache from the target table.

The following table shows the initial values of the lookup cache:

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 Integration Service passes the source rows into the Lookup transformation.

The following table shows the source rows:

SQ_CUST_ID	SQ_CUST_NAME	SQ_ADDRESS
80001	Marion Atkins	100 Main St.
80002	Laura Gomez	510 Broadway Ave.

SQ_CUST_ID	SQ_CUST_NAME	SQ_ADDRESS
99001	Jon Freeman	555 6th Ave.

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.

The following table shows the rows and values returned from the lookup:

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. The Integration Service eventually pass the rows to a Router transformation that creates a branch for insert rows and another branch for update rows. Each branch contains an Update Strategy transformation. The Update Strategy transformations flag the rows for insert or update 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.

If 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.

If 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.

When the Integration Service updates rows in the lookup cache it uses the primary key (PK_PRIMARYKEY) values for rows in the cache and the target table.

The Integration Service uses the sequence ID to generate a primary key for the customer that it does not find in the cache. The Integration Service inserts the primary key value into the lookup cache and it returns the value to the lookup/output port.

The Integration Service outputs the values from the input/output ports that match the input values.

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.

SQL Override and Dynamic Lookup Cache

You can add a WHERE clause in the lookup query to filter the records used to build the cache and to perform a lookup on the database table for an uncached lookup. However the Integration Service does not use the WHERE clause when inserting rows into a dynamic cache.

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 that you do not want to insert into the cache or target table. If you do not include the Filter transformation, you might get inconsistent results between the cache and the target table.

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 mapping, the Integration Service builds the lookup cache from the target table based on the lookup SQL override. All rows in the cache match the condition in the WHERE clause, EMP_STATUS = 4.

For example, the Integration Service reads a source row that meets the lookup condition you specify, but the value of EMP_STATUS is 2. Although the target might have the row where EMP_STATUS is 2, the Integration Service does not find the row in the cache because of the SQL override. The Integration Service inserts the row into the cache and passes the row to the target table. When the Integration Service inserts this row in the target table, you might get inconsistent results when the row already exists. In addition, not all rows in the cache match the condition in the WHERE clause in the SQL override.

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 in the Filter transformation and the WHERE clause in the SQL override:

```
EMP_STATUS = 4
```

Mapping Configuration for a Dynamic Lookup Cache

If you use a Lookup with a dynamic cache, you must configure the mapping to update the dynamic lookup cache and write the changed rows to the target.

Complete the following steps to configure a mapping with a dynamic lookup cache:

Flag the input rows of the Lookup transformation for insert or update.

By default, the row type of all input rows is insert. Add an Update Strategy transformation before the Lookup transformation to specify different row types for the input rows.

Specify how the Integration Service handles the input rows for the dynamic cache.

Select the **Insert Else Update** or **Update Else Insert** options to process rows flagged for insert or update.

Create separate mapping pipelines for rows to be inserted into the target and updated in the target.

Add a Filter or Router transformation after the Lookup transformation to route insert and update rows into separate mapping branches. Use the value of the `NewLookupRow` to determine the appropriate branch for each row.

Configure the row type for output rows of the Lookup transformation.

Add an Update Strategy transformation to flag rows for insert or update.

Insert Else Update

Use the **Insert Else Update** property to update existing rows in the dynamic lookup cache when the row type is insert.

This property only applies to rows entering the Lookup transformation where the row type is 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.

The following table describes how the Integration Service changes the lookup cache when the row type of the rows entering the Lookup transformation is insert:

Insert Else Update Option	Row Found in Cache	Data Cache is Different	Lookup Cache Result	NewLookupRow Value
Cleared - insert only	Yes	-	No change	0
Cleared - insert only	No	-	Insert	1
Selected	Yes	Yes	Update	2 ¹
Selected	Yes	No	No change	0
Selected	No	-	Insert	1

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.

Update Else Insert

Use the **Update Else Insert** property to insert new rows in the dynamic lookup cache when the row type is update.

You can select the **Update Else Insert** property in the Lookup transformation. This property only applies to rows entering the Lookup transformation where the row type is 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.

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.

The following table describes how the Integration Service changes the lookup cache when the row type of the rows entering the Lookup transformation is update:

Update Else Insert Option	Row Found in Cache	Data Cache is Different	Lookup Cache Result	NewLookupRow Value
Cleared (update only)	Yes	Yes	Update	2
Cleared (update only)	Yes	No	No change	0
Cleared (update only)	No	-	No change	0
Selected	Yes	Yes	Update	2
Selected	Yes	No	No change	0
Selected	No	-	Insert	1

Dynamic Lookup Cache and Target Synchronization

Configure downstream transformations to ensure that the dynamic lookup cache and the target are synchronized.

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 might reject the data.

Consider 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, output the rows to a different target.
- Use Update Strategy transformations after the Lookup transformation to flag rows for insert or update into the target.
- Verify that the Lookup transformation outputs 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 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.
- Select Insert and Update as Update when you define the update strategy target table options. This ensures that the Integration Service updates rows marked for update and inserts rows marked for insert.

Conditional Dynamic Lookup Cache Updates

You can update a dynamic lookup cache based on the results of a boolean expression. The Integration Service updates the cache when the expression is true.

For example, you might have a product number, quantity on hand, and a timestamp column in a target table. You need to update the quantity on hand with the latest source values. You can update the quantity on hand when the source data has a timestamp greater than the timestamp in the dynamic cache. Create an expression in the Lookup transformation similar to the following expression:

```
lookup_timestamp < input_timestamp
```

The expression can include the lookup and the input ports. You can access built-in, mapping, and parameter variables. You can include user-defined functions and refer to unconnected transformations.

The expression returns true, false, or NULL. If the result of the expression is NULL, the expression is false. The Integration Service does not update the cache. You can add a check for NULL values in the expression if you need to change the expression result to true. The default expression value is true.

Conditional Dynamic Lookup Cache Processing

You can create a condition that determines if the Integration Service updates the dynamic lookup cache. If the condition is false or NULL, the Integration Service does not update the dynamic lookup cache.

If the condition is false or NULL, regardless of the Lookup transformation properties, the `NewLookupRow` value is 0 and the Integration Services does not update the dynamic lookup cache with any row inserts or updates.

If the `NewLookupRow` value is 2, that is, the row is marked for update, and you enable `Update Else Insert`, the Integration Service performs one of the following:

- If the row exists in the cache, the Integration Service updates the existing row in the cache.
- If the row does not exist in the cache, the Integration Service inserts the new row in the cache.

If the `NewLookupRow` value is 1, that is, the row is marked for insert, and you enable `Insert Else Update`, the Integration Service performs one of the following:

- If the row does not exist in the cache, the Integration Service inserts the new row in the cache.
- If the row exists in the cache, the Integration Service updates the existing row in the cache.

Configuring a Conditional Dynamic Lookup Cache

You can configure an expression that determines whether the Integration Service updates a dynamic lookup cache.

1. Create the Lookup transformation.
2. On the **Run-time** tab of the **Properties** view, select **Lookup Caching Enabled**.
3. On the **Advanced** tab of the **Properties** view, select **Dynamic Lookup Cache**.
4. To enter the condition, click the down arrow for the **Update Dynamic Cache Condition** property.
The Expression Editor appears.
5. Define an expression condition.
You can select input ports, lookup ports, and functions for the expression.
6. Click **Validate** to verify that the expression is valid.
7. Click **OK**.

8. If applicable, configure the other advanced properties that apply to the dynamic lookup cache.

Dynamic Cache Update with Expression Results

The Lookup transformation can update the dynamic lookup cache values with the results of an expression.

For example, a product table target has a numeric column that contains an order count. Each time the Lookup transformation receives an order for the product, it updates the dynamic cache `order_count` with the results of the following expression:

```
order_count = order_count + 1
```

The Lookup transformation returns the `order_count`.

You can configure how the Integration Service handles the case where the expression evaluates to null.

Null Expression Values

The expression returns NULL if one of the values in the expression is null. However, you can configure an expression to return a non-null value.

If the expression refers to a lookup port, but the source data is new, the lookup port contains a default value. The default might be NULL. You can configure an `IsNull` expression to check for null values.

For example, the following expression checks if `lookup_column` is NULL:

```
iif (isnull(lookup_column), input_port, user_expression)
```

If the column is null, then return the `input_port` value. Otherwise return the value of the expression.

Expression Processing

The Integration Service can insert and update rows in the dynamic lookup cache based on the expression results. The expression results might vary based on whether the lookup port value is NULL and is included in the expression.

When you enable `Insert Else Update`, the Integration Service inserts a row with the expression result if the data is not in the cache. The lookup port value is NULL when data does not exist in the cache. If the expression refers to the lookup port value, the Integration Service substitutes the default port value in the expression. When you enable `Insert Else Update` and the data exists in the cache, the Integration Service updates the cache with the expression result.

When you enable `Update Else Insert`, the Integration Service updates the cache with the expression result if the data exists in the cache. When the data does not exist in the cache, the Integration Service inserts a row that contains the expression result. If the expression refers to a lookup port value, the Integration Service substitutes the default port value in the expression.

Configuring an Expression for Dynamic Cache Updates

You can configure an expression for a dynamic cache lookup update.

You must enable the Lookup transformation to perform dynamic lookups before you can create a conditional expression.

1. Create the Lookup transformation.

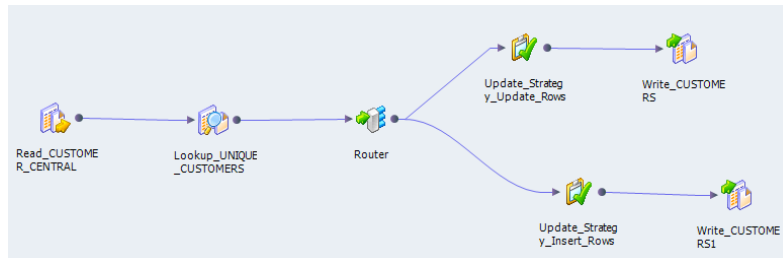
2. On the **Run-time** tab of the **Properties** view, select **Lookup Caching Enabled**.
3. On the **Advanced** tab of the **Properties** view, select **Dynamic Lookup Cache**.
4. If applicable, configure the other advanced properties that apply to the dynamic lookup cache.
5. To create an expression, select the **Columns** tab on the **Properties** view.
6. Click the drop-down arrow in the **Associated Port** column for the lookup port that you want to update.
7. Select **Associated Expression** from the drop-down list, and click **Enter**.
The Expression Editor appears.
8. Define the expression.
You can select input ports, lookup ports, and functions for the expression. The expression return value must match the datatype of the lookup port.
9. Click **Validate** to verify that the expression is valid.
10. Click **OK**.

Dynamic Lookup Cache Example

You can use a dynamic lookup cache to insert and update rows in the target. When you use a dynamic lookup cache, you can insert and update the same rows in the cache that you insert and update in the target.

For example, you need to update a table that contains customer data. The source data contains rows of customer data to insert or update in the target. Create a dynamic cache that represents the target. Configure a Lookup transformation to lookup customers in the dynamic cache.

The following figure shows a mapping that has a Lookup transformation with a dynamic lookup cache:



The Router transformation splits into two branches. The Router transformation passes insert rows into one branch and update rows into the other branch. Each branch contains an Update Strategy transformation that writes rows to the target. Both branches contain the same target.

When the mapping starts, the Integration Service builds the lookup cache from customer target table. When the Integration Service reads a row that is not in the lookup cache, it inserts the row in the cache.

The Lookup transformation returns each row to a Router transformation. Based on whether the row is marked for insert or update, the Router transformation directs the row to one of the Update Strategy transformations. The Router transformation determines whether rows are marked for insert or update based on the NewLookupRow property. The Update Strategy transformations mark each row as insert or update before passing it to the target.

The customer target table changes as the mapping runs. The Integration Service inserts new rows and updates existing rows in the lookup cache. The Integration Service keeps the lookup cache and customer target table synchronized.

To generate keys for the target, use Sequence-ID in the associated port. The Integration Service uses the sequence ID as the primary key for each new row inserted into the target table.

You increase session performance when you use a dynamic lookup cache because you build the cache from the database once.

Rules and Guidelines for Dynamic Lookup Caches

Consider the following guidelines when you use a dynamic lookup cache:

- You must set **On Multiple Matches** property to Report Error when you use a dynamic lookup cache. To reset the property, change the dynamic lookup to a static lookup, change the property, and then change the static lookup to a dynamic lookup.
- You cannot share the cache between a dynamic Lookup transformation and static Lookup transformation in the same target load order group.
- You can enable a dynamic lookup cache for a relational or flat file lookup.
- The Lookup transformation must be a connected transformation.
- You can 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 **Re-cache from Lookup Source**.
- You can only create an equality lookup condition. You cannot look up a range of data in a dynamic cache.
- You must associate each lookup port that is not in the lookup condition with an input port, sequence ID, or associated expression.
- 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 the rows to a different target.
- Verify that the Integration Service outputs the same values to the target that it 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 the 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, 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 that the Integration Service inserts rows in the dynamic cache and target table that match the WHERE clause.
- 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.

CHAPTER 28

Macro Transformation

This chapter includes the following topics:

- [Macro Transformation Overview, 437](#)
- [Macro Instruction, 437](#)
- [Macro Transformation Parameters, 438](#)
- [Macro Transformations in Dynamic Mappings, 438](#)
- [Macro Transformation Example, 438](#)
- [Macro Transformation in a Non-native Environment, 440](#)

Macro Transformation Overview

The Macro transformation is a passive transformation that enables dynamic functionality for any transformation logic.

Use the Macro transformation in a dynamic mapping that can accommodate changes to sources and targets at run time. The Macro transformation propagates the changes to a mapplet referenced by the Macro transformation, enabling the mapplet to act as a dynamic mapplet. The Macro transformation is always associated with a mapplet, called the macro instruction. The macro instruction mapplet can contain transformations that do not support dynamic functionality.

The Macro transformation cannot be configured as a reusable transformation.

Macro Instruction

The macro instruction is the mapplet referenced by the Macro transformation. Every Macro transformation must have a macro instruction.

The Developer tool prompts you to select a mapplet as the macro instruction when you create a Macro transformation. You cannot select a different mapplet as the macro instruction after you create the transformation.

Consider the following rules and guidelines for the macro instruction mapplet:

- The mapplet must be a midstream mapplet.
- The mapplet must have exactly one Input transformation.

- The mapplet cannot use active transformations.

Macro Transformation Parameters

When you add a Macro transformation to a mapping, you can use parameters in the macro instruction mapplet as mapping parameters as you would when you add a mapplet to a mapping. A mapping parameter represents a constant value that you can change between mapping runs.

When you add a mapplet with mapplet parameters as a macro instruction, you can set the instance values for the parameters. The instance value of a mapplet parameter is the parameter value for a specific mapping. Set the instance value from the **Parameters** tab in the **Properties** view for the Macro transformation.

For more information, see "Mapplet Parameters in Mappings" in the *Informatica® Developer Mapping Guide*.

Macro Transformations in Dynamic Mappings

You can add a Macro transformation to a dynamic mapping. The transformation contains dynamic ports that you can link to dynamic sources and targets.

When you create a Macro transformation and assign a macro instruction mapplet to the transformation, dynamic input and output ports are created based on the input and output groups of the macro instruction.

By default the output ports have the same type, precision, scale, and description as the input ports. You can configure the output ports by editing the base port associated with each output port in the **Ports** properties.

The number of generated ports created by a dynamic port in the Macro transformation must be either one or equal to the number of ports generated by the other dynamic ports in the transformation, excluding the passthrough ports. The passthrough ports in the Macro transformation can have any number of generated ports.

For example, a Macro transformation has three dynamic ports `transformed_input1`, `transformed_input2`, and `passthrough_input`. If `transformed_input1` generates three ports, then `transformed_input2` must generate either one or three ports. `passthrough_input` can generate any number of ports.

Macro Transformation Example

You are a data steward for a health care insurer, and you need to encrypt personal information about your insurance policy holders. The schema of the data source might vary, so you need to use a dynamic mapping.

You need a dynamic mapping that reads data from a source, encrypts all string data using a Data Masking transformation, and writes the modified data to a target. Since the Data Masking transformation does not support dynamic functionality, you need to use a Macro transformation to enable dynamic functionality for a mapplet that contains the Data Masking transformation. The Macro transformation allows mapplet transformation logic to act as dynamic mapping logic by accepting input with any number of string ports and passing each string input port through a copy of the data masking mapplet.

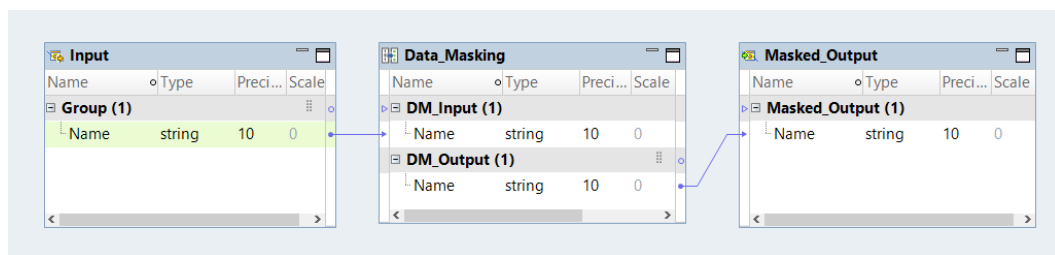
To use the Macro transformation, you perform the following steps:

1. Configure the mapplet.
2. Configure the dynamic mapping with the Macro transformation.
3. Run the mapping.

Configure the mapplet

Before you configure the Macro transformation, you create a mapplet that performs the data encryption. The mapplet includes an Input transformation, a Data Masking transformation, and an Output transformation. The Input transformation has one port of type string that connects to the input group of the Data Masking transformation. In the Data Masking transformation, you set a masking type for the port and then connect the output port to the Output transformation.

The following image shows the mapplet:



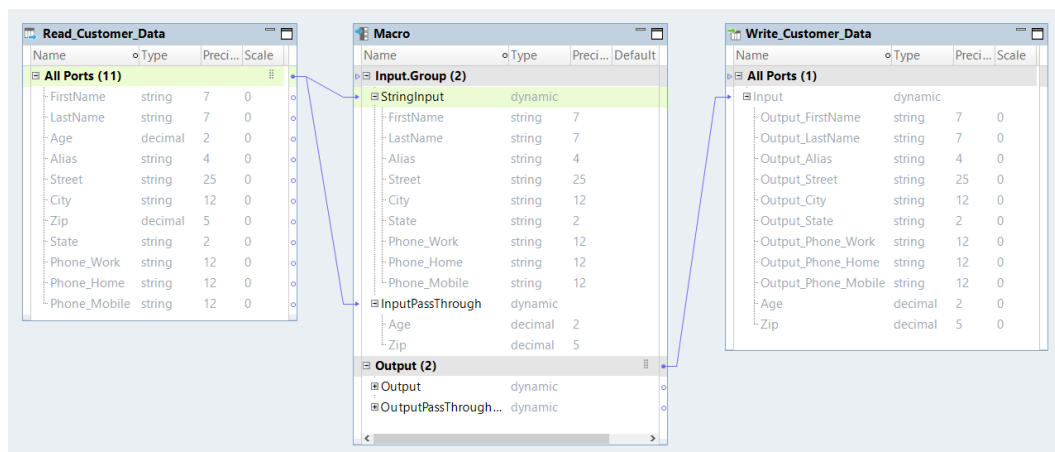
Configure the mapping

After you configure the mapplet, you configure the dynamic mapping. You create a mapping that reads from a parameterized source and configure the mapping to get the column names at run time. Then you add a Macro transformation to the dynamic mapping and select the data masking mapplet as the macro instruction.

You connect the Read transformation port group to the input port *StringInput* and to the input passthrough port *InputPassThrough*. You configure the input rule for *StringInput* to include only string type ports.

Finally, you add a Write transformation to the mapping and configure it to define ports by the mapping flow. You connect the Macro transformation output group to the Write transformation dynamic input port.

The following image shows the dynamic mapping:



Run the mapping

When you run the mapping, the mapping determines the ports from the data source. The Macro transformation creates a copy of the mapplet for each string input port and passes each string port through a

copy of the maplet. The Data Masking transformation encrypts the string data, and passes the output back to the mapping through the Macro transformation. The mapping writes the encrypted string data and the unchanged decimal data to the target.

Macro Transformation in a Non-native Environment

The Macro transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions for batch and streaming.
- Databricks spark engine. Supported without restrictions.

CHAPTER 29

Match Transformation

This chapter includes the following topics:

- [Match Transformation Overview, 441](#)
- [Match Analysis, 442](#)
- [Match Score Calculations, 445](#)
- [Master Data Analysis, 449](#)
- [Identity Match Analysis and Persistent Index Data, 450](#)
- [Match Mapping Performance, 451](#)
- [Match Performance in Identity Analysis, 453](#)
- [Match Transformation Views, 455](#)
- [Match Transformation Ports, 457](#)
- [Match Mapplets, 462](#)
- [Configuring a Match Analysis Operation, 463](#)
- [Match Transformation in a Non-native Environment, 464](#)

Match Transformation Overview

The Match transformation is an active transformation that analyzes the levels of similarity between records. Use the Match transformation to find records that contain duplicate information in a data set or between two data sets.

The Match transformation analyzes the values on an input port and generates a set of numeric scores that represent the degrees of similarity between the values. You can select multiple ports to determine the overall levels of similarity between the input records. You specify a minimum score as a threshold value to identify the records that are likely to contain duplicate information.

You can use the Match transformation in the following data projects:

- Customer Relationship Management. For example, a store designs a mail campaign and must check the customer database for duplicate customer records.
- Mergers and acquisitions. For example, a bank buys another bank in the same region, and the two banks have customers in common.
- Regulatory compliance initiatives. For example, a business operates under government or industry regulations that insist all data systems are free of duplicate records.

- Financial risk management. For example, a bank may want to search for relationships between account holders.
- Master data management. For example, a retail chain has a master database of customer records, and each retail store in the chain submits records to the master database on a regular basis.
- Any project that must identify duplicate records in a data set.

Match Analysis

You can define different types of duplicate analysis in the Match transformation. The duplicate analysis operations that you define depend on the number of data sources in the mapping and the type of information that the sources contain.

Consider the following factors when you configure a Match transformation:

- You can select a single column from a data set or you can select multiple columns.
- You can analyze columns from a single data source or you can analyze two data sources.
- You can configure the Match transformation to analyze the raw data in the input port fields, or you can configure the transformation to analyze the identity information in the data.
- You can configure the Match transformation to write different types of output. The type of output that you select determines the number of records that the transformation writes and the order of the records.
- To increase performance, sort the input records into groups before you perform the match analysis.

Column Analysis

When you configure a Match transformation, you select one or more columns for analysis.

The Match transformation analyzes columns in pairs. When you select a single column for analysis, the transformation creates a temporary copy of the column and compares the source column with the temporary column. When you select two columns for analysis, the transformation compares the values across the two columns that you select. The transformation compares each value in one column with all of the values in the other column. The transformation returns a match score for each pair of values that it analyzes.

You select the columns to analyze when you configure a strategy in the Match transformation. The strategy specifies the columns to analyze and the algorithm to apply to the columns. The algorithm calculates the levels of similarity between each pair of values. The different algorithms in the transformation use different criteria to measure the levels of similarity between the values. You can define multiple strategies in a transformation, and you can assign different columns to each strategy.

Column Analysis Example

You want to compare the values in a column of surname data. You create a mapping that includes a data source and a Match transformation. You connect the *Surname* port to the Match transformation. The transformation creates a temporary copy of the data on the *Surname* port when the mapping runs.

The following image shows a fragment of the surname data:

	A	B
1	Surname	Surname_1
2	Annan	Annan
3	Baker	Baker
4	Barker	Barker
5	Edwards	Edwards
6	Parker	Parker
7	Smith	Smith
8	Smith	Smith
9	Zhang	Zhang

The mapping generates a set of match scores that indicate that the following values might be duplicates:

- Baker, Barker
- Barker, Parker
- Smith, Smith

When you review the data, you decide that *Baker*, *Barker*, and *Parker* are not duplicate values. You decide that *Smith* and *Smith* are duplicate values.

Single-Source Analysis and Dual-Source Analysis

You can configure the Match transformation to analyze data from one or two data sources. You select the ports from each data source when you define a strategy in the transformation.

When you configure the transformation to perform single-source analysis, you select one or more ports from a single data set. When you configure the transformation to perform dual-source analysis, you select one or more ports from each data set. You select the ports in pairs. For each pair of ports that you select, the transformation compares each value in one port with every value in the other port. If you perform single-source analysis on data from a single column, the transformation creates a temporary copy of the port that you select.

Note: When you perform identity match analysis, you can compare a data source to a persistent index of identity data that you created in an earlier mapping. Use the **Match Type** options to specify identity analysis with a persistent index.

Field Match Analysis and Identity Match Analysis

You can configure a Match transformation to perform field match analysis or identity match analysis.

In field match analysis, the Match transformation analyzes the source data that enters the transformation. You can perform field match analysis on any type of data. In identity match analysis, the Match transformation generates an index of alternative data values from the input data and analyzes the index data. Configure the Match transformation for identity match analysis when the input ports contain identity data. An identity is a group of data values that identifies a person or an organization.

A data set can represent a single identity in different ways. For example, the following data values represent the name John Smith:

- John Smith
- Smith, John

- jsmith@email.com
- SMITHJMR

The Match transformation reads the identity data in a record and calculates the possible alternative versions of the identity. The transformation creates an index that includes the current versions and alternative versions of the identities. The Match transformation analyzes the index values and not the values in the input records.

Identity Population Files

Identity match operations read reference data files called populations. The population files define the potential variations in the identity data. The files do not install with Informatica applications. You buy and download the population data files from Informatica.

Install the files to a location that the Content Management Service can access. Use Informatica Administrator to set the location on the Content Management Service.

Groups in Match Analysis

A match analysis mapping can take a long time to run because of the number of data comparisons that the transformation must perform. The number of comparisons relates to the number of data values on the ports that you select.

The following table shows the number of calculations that a mapping performs for different numbers of data values on a single port:

Number of data values	Number of comparisons
10,000	50 million
100,000	5,000 million
1 million	500,000 million

To reduce the time that the mapping takes to run, assign the input data records to groups. A group is a set of records that contain identical values on a port that you specify. When you perform match analysis on grouped data, the Match transformation analyzes the records within each group. The transformation does not compare the records in one group with the records in another group. The groups reduce the overall number of comparisons that the transformation must perform without any loss of accuracy in the mapping analysis.

Consider the following rules and guidelines when you organize data into groups:

- The port on which you group the data is the group key port. A group key port must contain a range of duplicate values, such as a city name or a state name in an address data set. If the mapping data does not contain a usable group key port, use the Key Generator to create the port from the current mapping data. Connect the group key output port from the Key Generator transformation to the Match transformation. You can also use the Key Generator transformation to add sequence identifiers to the mapping data.
- Field match operations must specify a group key port. If you configure the Match transformation for identity analysis, do not select a group key port. The identity analysis generates group keys for the identity index data.
- Do not specify a group key port that you plan to use in the match analysis.
- When you create groups, you must verify that the groups are a valid size. If the groups are too small, the match analysis might not find all the duplicate data in the data set. If the groups are too large, the match

analysis might return false duplicates. Select group keys that create an average group size of 10,000 records.

- Groups do not reorder the position of the records in the mapping data set.

Match Pairs and Clusters

The Match transformation can read and write different numbers of input rows and output rows, and it can change the sequence of the output rows. You determine the output format for the results of the match analysis.

The transformation can write rows in the following formats:

Matched pairs

The transformation writes a row for every pair of records that match with a score that meets the match threshold. The transformation writes each pair of records to a single row.

Because a record might match more than one other record, a record might appear on more than one output row.

Best match

The transformation writes a row for each record in a data set and adds the most similar record from another data set to the same row.

Clusters

The transformation assigns the output records to clusters based on the levels of similarity between the records. A cluster is a set of records in which each record matches at least one other record with a score that meets the match threshold. The transformation writes each record to a single row.

Each record in a cluster must match at least one other record in the cluster. Therefore, a cluster can contain pairs of records that do not match each other. A cluster can contain a single record if the record does not match any other record.

Note: The Clusters option in field analysis corresponds to the Clusters - Match All option in identity analysis. The Clusters - Best Match option in identity analysis combines cluster calculations and matched pair calculations.

Configure the output options on the **Match Output** view of the transformation.

RELATED TOPICS:

- [“Cluster Output Options” on page 446](#)

Match Score Calculations

The match score is a numerical value that represents the degree of similarity between two column values. An algorithm calculates a match score as a decimal value in the range 0 through 1. An algorithm assigns a score of 1 when two column values are identical.

When you select multiple column pairs for analysis, the transformation calculates an average score based on the scores in the selected columns. By default, the transformation assigns equal weight to scores from each pair of columns. The transformation does not infer the relative importance of the column data in the data set.

You can edit the weight values that the transformation uses to calculate the match score. Edit the weight values when you want to assign higher or lower priority to columns in the data set.

You can also set the scores that the transformation applies when it finds a null value in a column. By default, the transformation treats null values as data errors and assigns a low match score to any pair of values that contains a null.

Note: The algorithm you select determines the match score between two values. The algorithm generates a single score for the two values. The match scores do not depend on the type of match output or the type of scoring method you select.

Weighted Scores

When you select multiple columns for match analysis, the transformation calculates an average score for each record based on the scores in the columns. The average score includes any weight values that you apply to the comparison algorithms for each column.

By default, all algorithms use a weight value of 0.5. You can increase this value if the selected columns are more likely to contain duplicated information. You can decrease the weight value if duplicate values in the selected columns are less likely to indicate genuine duplicate information between records. The Match transformation uses the average score as the single match score for each pair of records.

Null Match Scores

A match algorithm applies a predefined match score to a pair of values when one or both values are null. You can edit the match score that a field match algorithm applies to null values.

Null Match Scores and Field Match Algorithms

When you configure a field match algorithm, verify the match score values that the algorithm applies to null data. A field match algorithm applies a default score of 0.5 when it compares two values and one or both values are null. A score of 0.5 indicates a low level of similarity between data values.

Consider the following rules and guidelines when you verify the null match scores:

- When the algorithm analyzes columns that contain primary keys or other critical data, do not edit the default scores. In this case, a null value represents a data error, and the default scores are appropriate for the data.
- When the algorithm analyzes columns that can optionally contain data, update the null match score values to the same value as the match threshold. You cancel the effect of the null values on the match analysis when you set the null match scores to the match threshold value.

Null Match Scores and Identity Match Algorithms

An identity match algorithm applies a match score of 0 when it compares two values and one or both values are null. Identity match analysis assigns a record with a null match score to a unique record cluster and records a cluster size value of 1. You cannot edit the score that an identity match algorithm applies to null data.

Cluster Output Options

Select a cluster output option when you want to organize similar or identical records in the output data.

When you select a cluster output option, the transformation adds a cluster ID value to each output record. You can sort the records by the cluster ID values. The transformation output includes a row for every record. If a record does not match another record with a score that meets the match threshold, the transformation assigns a unique cluster ID to the record. Use the **Match Output** view to select or update the cluster output options.

You can select the following cluster output options:

Clusters

Select the option to assign cluster ID values to the output records.

Clusters - Best Match

Select the option to add the record pair with the highest match score to a cluster. Because a record might represent the best match with more than one other record, more than one record pair can share a cluster ID value.

Clusters - Match All

The **Clusters - Match All** option works in the same way as the **Clusters** option.

The transformation uses **Clusters - Match All** and **Clusters - Best Match** as option names in identity match analysis.

Note: If a Data Integration Service runs multiple Match transformations concurrently, the Data Integration Service generates unique cluster ID values for the output from each transformation. Therefore, the cluster ID values for the records that each transformation generates can be non-consecutive.

The Clusters Option and the Clusters - Match All Option

Select the Clusters option in field match analysis. Select the Clusters - Match All option in identity match analysis.

The Match transformation uses the following rules to create the clusters:

- When two records have a match score that meets the match threshold, the Match transformation adds the records to a cluster.
- When a record in the data set matches any record in the cluster, the transformation adds the record to the cluster.
- If a record in one cluster matches a record in another cluster, the process merges the clusters.
- The transformation performs a continual sweep of the match results until all the records belong to a cluster.
- If a record does not match any other record in the data set, the transformation assigns a unique cluster ID value to the record.

The Clusters - Best Match Option

Select the Clusters - Best Match option in identity match analysis.

The transformation uses the following rules to create the clusters:

- The transformation identifies the record that has the highest match score with the current record. If the match score meets the threshold, the transformation adds the pair of records to a cluster.
- If one of the matching records is in a cluster, the transformation adds the other record to the current cluster.
- The transformation performs a continual sweep of the match score results until all the records belong to a cluster.
- A cluster can contain a single record if the record does not match any other record in the data.

Note: You can use the **Match** property on the **Match Output** view to specify how the transformation compares a single data source to a persistent data store. The **Match** property determines whether the transformation looks for duplicates within the source data or the persistent data store.

RELATED TOPICS:

- [“Match Pairs and Clusters” on page 445](#)

Driver Scores and Link Scores in Cluster Analysis

When you select a cluster output option in the Match transformation, you can add link score and driver score data to the output.

The link score is the score between two records that identifies the records as members of the same cluster. The links between records determine the composition of the cluster. Any record can link to any other record in the same cluster.

The driver score is the score between the record with the highest sequence ID value in a cluster and another record in the same cluster. Driver scores provide a means to assess all records in a cluster against a single record. When you add driver scores to the match output, the mapping runs more slowly, as the Match transformation cannot calculate the driver scores until all the clusters are complete.

Note: Match analysis generates a single set of scores for each strategy that you define. The driver score and the link score indicate the match scores for different pairs of records in each cluster. The driver scores and link scores can depend on the order in which the records enter the transformation. The driver score might be lower than the match threshold.

Cluster Analysis Example

You configure a field match strategy to analyze a column of surname data. You set a match threshold of 0.825 in the strategy. You select a clustered output format, and you run the Data Viewer on the transformation.

The following table shows the data that the Data Viewer displays:

Surname	Sequence ID	Cluster ID	Cluster Size	Driver ID	Driver Score	Link ID	Link Score
SMITH	1	1	2	1 - 6	1	1 - 1	1
SMYTH	2	2	2	1 - 3	0.83333	1 - 2	1
SMYTHE	3	2	2	1 - 3	1	1 - 2	0.83333
SMITT	4	3	1	1 - 4	1	1 - 4	1
SMITS	5	4	1	1 - 5	1	1 - 5	1
SMITH	6	1	2	1 - 6	1	1 - 1	1

The Data Viewer contains the following information about the surname data:

- SMITT and SMITS do not match any record with a score that meets the match threshold. The Match transformation determines that the records are unique in the data set.
SMITT and SMITS each has a cluster size of 1, as each record is the sole record in its cluster. To find unique records in cluster output, search for clusters that contain a single record.
- SMITH and SMITH have a link score of 1. The Match transformation determines that the records are identical. The transformation adds the records to a single cluster.
- SMYTH and SMYTHE have a link score of 0.83333. The score exceeds the match threshold. Therefore, the transformation adds the records to a single cluster.

Master Data Analysis

When you analyze two data sources in the Match transformation, you must identify a source as the master data set. The transformation compares data values from each record in the data set that you specify with the corresponding values in every record in the second data set.

In many organizations, a master data set constitutes a permanent, high-quality data store. Before you add records to a master data set, use the Match transformation to verify that the records do not add duplicate information to the master data.

Master Data Example

A bank maintains a master data set of customer account records. The bank updates the master data set every day with records that identify new customer accounts. The bank uses a duplicate analysis mapping to verify that the new records do not duplicate the customer information in the master data set. The master data set and the new account tables have a common structure, and the tables use the same type of database. Therefore, the bank can reuse the duplicate analysis mapping each time it needs to update the master data set.

Directionality in Master Data Set Analysis

The Match transformation compares records from one data set to another in a single direction. The transformation compares each record in the master data set to all of the records in the second data set. The transformation does not compare each record in the second data set to all of the records in the master data set. Therefore, the selection of the master data set can affect the results of the match analysis.

The following table shows two data sets that you can compare in identity match analysis:

Data Set 1	Data Set 2
Alex Bell	Alexander Bell
Alexander Graham Bell	Thomas Edison
Alva Edison	Nicola Tesla
Marie Curie	Irene Joliot Curie
Dorothy Crowfoot	Dorothy Hodgkin

If you select Data Set 1 as the master data set and you select the **Best Match** output option, the output includes the following records:

- Alex Bell, Alexander Bell
- Alexander Graham Bell, Alexander Bell

If you select Data Set 2 as the master data set and you select the **Best Match** output option, the output includes the following records:

- Alexander Bell, Alex Bell

When Data Set 2 is the master data set, the transformation cannot match Alexander Bell and Alexander Graham Bell, because Alexander Bell already matches Alex Bell in the output data.

Mapping Reuse

If you add data to a master data set on a regular basis, configure a duplicate analysis mapping that you can reuse. You can reuse the mapping if the port configurations on the data source that you compare to the master data set do not change.

When you run the mapping, verify that the transformation specifies the master data and the newer data. You can run the mapping without any other configuration update.

Identity Match Analysis and Persistent Index Data

When you run a mapping to analyze identity information, the Match transformation generates an index that stores the alternative versions of the identities in the data set. By default, the Match transformation writes the index data to temporary files. You can configure the transformation to save the index data to database tables.

The index tables that you generate represent a data store that you can reuse in subsequent mappings. You can compare the index tables to a data source, and you can optionally update the index tables with index data from the data source. Because the transformation does not regenerate the index tables, the subsequent mappings run faster. Additionally, the index tables can represent a trusted data store of identity data.

RELATED TOPICS:

- [“Match Performance in Identity Analysis” on page 453](#)

Rules and Guidelines for Persistent Index Data

Consider the following rules and guidelines when you configure a Match transformation to analyze a master data set of identity information:

- To generate a reusable index for a master data set, configure the transformation to write the index data to database tables. The database tables constitute a persistent store of the index data.
- To compare the identities in another data set to the index data store, configure the data set as the mapping data source. Configure the Match transformation to read the data source and the index data store. Select the index tables from the default schema of the database connection that you specify.
- The Match transformation adds the sequence identifier value from the input record to the index data rows that correspond to the record. The *SequenceID* input port contains the sequence identifiers. The transformation uses the sequence identifiers to track the index data through the different steps in the match analysis. Do not disconnect the sequence ID port.
- When you connect a Match transformation to an index store, the transformation reuses the population, key level, key type, and key field property values from the transformation that created the store. The transformation also reuses the port configuration from the transformation that created the store.
If the transformation properties do not match, the identity analysis cannot compare the mapping source data and the index data correctly.
- The Match transformation uses the data on the input port that you select as the key field to generate the identity index. The transformation can also write data from other ports to the index. If you disconnect the non-key field data ports from the transformation, you erase any data in the corresponding index columns when you run the mapping. To preserve the input port data in the index tables, do not disconnect the input data ports.

- You can disable the match analysis in the Match transformation when you generate the index table data for a data set. For example, you might disable the match analysis when you create an index store for a data set. When you disable match analysis, the mapping runs faster.

When you disable match analysis, the Match transformation can generate and display persistence status codes and persistence status descriptions. The transformation does not generate or display match scores or other data associated with the results of the match analysis. For example, if you configure the transformation to assign records to clusters and you disable match analysis, the transformation does not generate or display cluster ID values.

- You determine whether the Match transformation updates the index store with data from the mapping source. The Match transformation uses sequence identifiers to determine whether the rows in the index store and in the mapping data represent the same records.

Match Mapping Performance

You can preview the data factors that determine the performance of the Match transformation before you run the mapping that contains the transformation. You can verify that the system has the resources to run the mapping. You can also verify that you configured the transformation correctly to measure the levels of similarity in the input data.

Use the **Match Performance Analysis** option to verify that the system has the required resources. Use the **Match Cluster Analysis** option to verify that the mapping can accurately measure the levels of similarity in the input data.

Run match performance analysis and match cluster analysis on any Match transformation that reads a single data source. Run match performance analysis on any Match transformation that performs dual-source field match analysis. Do not run match performance analysis or match cluster analysis on an identity match strategy that connects to index tables.

Drill-down on Match Performance Analysis

You can drill down on the match analysis data to view the record pairs that meet or exceed the match threshold. Double-click a record in the **Details** view, and use the Data Viewer to view the records that match the record that you select. The Data Viewer displays the data for each pair of records on a single row. The row contains the row identifier of each record in the pair.

Drill-down on Match Cluster Analysis

You can drill down on the cluster analysis data to view the records in each cluster. Double-click a cluster in the **Details** view and read the data in the Data Viewer. The Data Viewer displays one cluster at a time. The cluster data includes the score options that you selected, such as the driver score, link score, driver identifier, or link identifier.

Match Transformation Logging

When you run a mapping that uses a Match transformation, the Developer tool log tracks the number of comparison calculations that the mapping performs. To view the log data, select the **Show Log** option in the Data Viewer.

The mapping updates the log every 100,000 calculations.

Viewing Match Cluster Analysis Data

You can view statistical data on the clusters that the transformation can create. The cluster statistics summarize the level of record duplication in the data set based on the current mapping configuration.

To view the data, right-click the Match transformation in the mapping canvas and select **Match Cluster Analysis**.

Before you run the analysis, validate the mapping that contains the transformation.

Match cluster analysis displays data for the following properties:

Property	Description
Source	The number of input data rows.
Last run	The date and time of the analysis.
Total number of discovered clusters	The number of clusters that the match analysis generates when the mapping runs.
Minimum cluster size	The number of records in the cluster or clusters that contain the fewest records. If the minimum cluster size is 1, the data set contains at least one unique record.
Maximum cluster size	The number of records in the cluster or clusters that contain the most records. If this value greatly exceeds the average cluster size, the largest cluster might contain false duplicates.
Number of unique records	The number of records in the data set that do not match another record with a score that meets the match threshold.
Number of duplicate records	The number of records in the data set that match another record with a score that meets the match threshold.
Total comparisons	The number of comparison operations that the mapping performs.
Average cluster size	The average number of records in a cluster.

Viewing Match Performance Analysis Data

You can view statistical data on the record groups that the mapping reads as input data.

To view the data, right-click the Match transformation in the mapping canvas and select **Match Performance Analysis**.

Before you run the analysis, validate the mapping that contains the transformation.

Match performance analysis displays data for the following properties:

Property	Description
Source	The number of input data rows.
Last run	The date and time of the analysis.
Total number of discovered groups	The number of groups defined for the data set, based on the selected group key value.

Property	Description
Throughput (records per minute)	Variable value that estimates the speed of the match analysis. You set this value. Use the value to estimate the time required to run the match analysis.
Estimated time to match records	The time taken to analyze all records in the data set, based on the Match transformation configuration.
Total number of pairs generated	The number of comparisons that the transformation must perform, based on the number of input data rows and the number of groups.
Minimum group size	Variable value that indicates the minimum number of records a group can contain. You set this value. Use the value to verify that the mapping will create groups of a usable size. Note: The minimum group size value does not determine the size of the groups created when the mapping runs.
Number of groups below minimum threshold	The number of groups that contain fewer records than the minimum group size value. If many groups are below the minimum group size, you might need to edit the transformation and select a different group key.
Maximum group size	Variable value that indicates the maximum number of records a group can contain. You set this value for the performance analysis. Use the value to verify that the mapping will create groups of a usable size. Note: The value does not determine the size of the groups created when the mapping runs.
Number of groups above maximum threshold	The number of groups that contain more records than the maximum group size value. If many groups are above the maximum group size, you might need to edit the transformation and select a different group key.

Match Performance in Identity Analysis

To improve the mapping performance when you perform identity analysis on two data sets, configure the Match transformation to read identity index data from database tables. Run a mapping to create the index tables for the master data set. Run the mapping again to compare the index data to another data source.

Use the options on the **Match Type** view to identify the database tables that store the index data. Use the same options to select the index tables when you configure the transformation to compare the index data to data from another source.

To write the index data to database tables, perform the following tasks:

1. Create a mapping that reads a data source of identity information.
2. Configure a Match transformation in the mapping to write the index data to a database.
3. Run the mapping to generate the index data. The index data represents a data store that you can reuse.

To read the index data from database tables, perform the following tasks:

1. Create a mapping that reads another identity data source.
2. Configure a Match transformation in the mapping to read the index data from the database that you specified earlier.

When the mapping data source and the index data share a common structure, you can reuse the mapping that generated the index data.

3. Run the mapping to compare the data source to the index data.

The mapping generates index data for the data source. The mapping does not need to generate index data for the larger data set. Therefore, the mapping runs faster than a dual-source mapping that generates index data for both data sets.

RELATED TOPICS:

- [“Identity Match Analysis and Persistent Index Data” on page 450](#)

Creating a Data Store for Identity Index Data

Configure a mapping that reads a data source that contains identity information. Use a Match transformation to write the index data to a database.

1. Create a mapping, and add the data source to the mapping canvas.
2. Add a Match transformation to the mapping canvas.
3. On the data source, select the ports that contain the identity information.
 - Connect the identity information ports to the Match transformation.
 - Connect the port that contains the sequence identifier values to the Match transformation.
4. On the Match transformation, select the **Match Type** view.
5. Set the match type to **Identity Match with Persistent Record ID**.
6. Configure the following options to create the index data store.
 - Set the persistence method to **Update the database with new IDs**.
 - Verify the matching process value. The default value is **Enable**. If you disable the matching process, the mapping creates the identity index tables but does not perform match analysis on the data.
 - On the DB Connection menu, select a database for the index tables.
 - On the Persistent Store menu, select **Create New**. Enter a name for the index in the **Create Storage Tables** dialog box.
7. Perform any other configuration step that the Match transformation requires. For example, configure a transformation strategy.
8. Add a target data object to the mapping.
9. Connect the Match transformation output ports to the target data object.
10. Run the mapping.

The mapping writes the index data for the data source to the database tables that you specify.

Using the Index Data Store in Single-Source Analysis

Configure a mapping that reads a data source of identity information. Use a Match transformation to compare the data source to the index data store for the master data set.

Before you configure the mapping, verify that the data source contains a column of sequence identifier values.

To save time, you can copy or reuse the mapping that created the index data store.

1. Open the mapping that generated the index data store.
Alternatively, open a copy of the mapping.
2. Verify the data source in the mapping.

If required, replace the data source with a source that contains the current data.

Note: If you delete the data source, you also delete the port connections on the Match transformation.

3. Identify the data source ports that contain the identity information.
 - Connect the identity information ports to the Match transformation.
 - Connect the port that contains the sequence identifier values to the Match transformation. The input ports and the input port order must correspond to the input ports on the transformation that created the index tables.
4. On the Match transformation, select the **Match Type** view.
5. Set the match type to **Identity Match with Persistent Record ID**.
6. Verify the population, key level, key type, and key field values.
7. Configure the options to identify the index data store:
 - Set the persistence method. For example, select **Do not update the database** to preserve the current data in the index tables.
 - Set the matching process to **Enable**.
 - On the DB Connection menu, select the database that contains the index tables.
 - On the Persistent Store menu, browse to the tables that contain the index data.
8. Configure the following properties on the Match Output view:
 - **Match**. Identifies the records to analyze when the transformation reads index data from database tables.
 - **Output**. Filters the records that the transformation writes as output.
9. Perform any other configuration step that the Match transformation requires.
For example, configure a transformation strategy.
10. Verify the data target in the mapping.
If required, replace the data target with another target object.
11. Connect the Match transformation output ports to the target data object.
12. Run the mapping.

The mapping compares the data source records with the index data store. The transformation writes the index data for the data source to the data store.

Match Transformation Views

The Match transformation organizes the options that you can configure on a series of views. In a reusable transformation, the views appear as tabs on the Developer tool canvas. To open a view, click a tab. In a nonreusable transformation, the views appear as a list on the canvas. To open a view, click an item on the list.

When you configure a match analysis operation, you can configure the following views:

General

Use the General view to update the name and the description of a nonreusable Match transformation. The description is optional.

Ports

Use the Ports view to verify the input ports and output ports on a nonreusable Match transformation.

Overview

Use the Overview to update the name and the description of a reusable transformation. The description is optional. You also use the Overview to create the input ports and output ports on a reusable transformation.

Match Type

Use the Match Type view to select the type of duplicate analysis that the transformation performs. You can select field match analysis or identity match analysis. You can specify a single data source or two data sources.

Strategies

Use the Strategies view to define one or more strategies to analyze the input data. You select two data columns in each strategy, and you assign a match analysis algorithm to the columns.

Match Output

Use the Match Output view to specify the structure and format of the output data.

Parameters

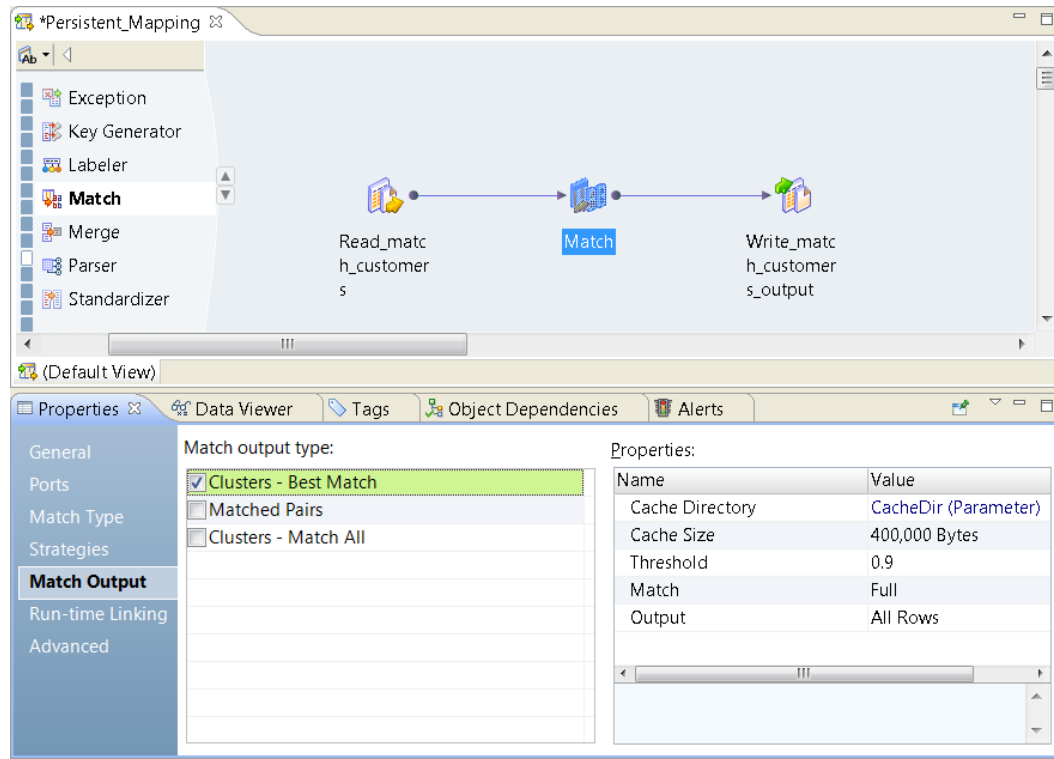
Use the Parameters view to define any parameter that the Data Integration Service can apply to the transformation when you run a mapping that contains the transformation.

Advanced

Use the Advanced view to specify the following properties:

- The level of detail in any log message that the mapping writes for the transformation.
- The number of processes that identity match operations use when you run a mapping that contains the transformation.
- Whether the transformation passes data from identical records directly to the output ports. You can filter identical records when you configure the transformation to write clustered output.

The following image shows the views on a Match transformation that you configure for identity analysis with a persistent index store:



Match Transformation Ports

The Match transformation includes a set of predefined input ports and output ports that contain metadata for the match analysis operations that you define. The transformation selects or clears the ports when you configure the match type and match output options.

When you configure the transformation, review the metadata ports. When you add the transformation to a mapping, verify that you connect the metadata ports to the correct ports on the upstream and downstream mapping objects.

Match Transformation Input Ports

The predefined input ports contain the metadata that the transformation requires for match analysis.

After you create a Match transformation, you can configure the following input ports:

SequenceId

Unique identifier for each record in the mapping source data set. Every record in an input data set must include a unique sequence identifier. If a data set contains duplicate sequence identifiers, the Match transformation cannot identify duplicate records correctly. Use the Key Generator transformation to create unique identifiers if none exist in the data.

When you create an index data store for identity data, the Match transformation adds the sequence identifier for each record to the data store. When you configure the transformation to compare a data

source with the index data store, the transformation might find a common sequence identifier in both data sets. The transformation can analyze the sequence identifiers if they are unique in the respective data sets.

GroupKey

Key value that identifies the group to which the record belongs.

Note: To improve mapping performance, configure the GroupKey input port and the output port that connects to it with the same precision value.

Match Transformation Output Ports

The predefined output ports contain metadata about the analysis that the transformation performs.

After you create a Match transformation, you can configure the following output ports:

GroupKey

Key value that identifies the group to which the record belongs.

Downstream transformations such as the Association transformation can read the group key value.

ClusterId

The identifier of the cluster to which the record belongs. Used in cluster output.

ClusterSize

The number of records in the cluster to which a record belongs. When a cluster contains a unique record, the cluster size is 1. Used in cluster output.

RowId and RowId1

A unique row identifier for the record. The Match transformation uses the row identifier to identify the row during the match analysis operations. The identifier might not match the row number in the input data.

DriverId

The row identifier of the driver record in a cluster. Used in cluster output. The driver record is the record in the cluster with the highest value on the SequenceID input port.

DriverScore

The transformation assigns a driver score in matched pair output and clustered output. In a matched pair, the driver score is the match score between the pair of records. In a cluster, the driver score is the match score between the current record and the driver record in the cluster.

LinkId

The row identifier of the record that matched with the current record and linked it to the cluster. Used in cluster output.

LinkScore

The match score between two records that results in the creation of a cluster or the addition of a record to a cluster. The LinkID port identifies the record with which the current record shares the link score. Used in cluster output.

PersistenceStatus

An eight-character code that represents the results of the match analysis on an input record. Used in single-source identity analysis when the transformation compares the data source to an index data store.

The transformation populates the first three characters in the code. The transformation can return different characters at each position. The transformation returns 0 for positions four through eight.

When you configure the transformation to generate matched pair output, the transformation creates a PersistenceStatus port and a PersistenceStatus1 port.

PersistenceStatusDesc

A text description of the persistence status code values. Used in single-source identity analysis when the transformation compares the data source to an index data store.

When you configure the transformation to generate matched pair output, the transformation creates a PersistenceStatusDesc port and a PersistenceStatusDesc1 port.

Persistence Status Codes and Persistence Status Descriptions

The persistence status codes and the persistence status descriptions describe the relationship between the different types of index data that the Match transformation analyzes. The transformation generates the status codes and the status descriptions when you configure the transformation to read a persistent identity data store.

The transformation writes the persistence status code to the PersistenceStatus port. The code contains eight characters. The transformation populates the first three positions in the string with code values. The transformation returns 0 for positions four through eight.

The transformation writes the persistence status description to the PersistenceStatusDesc port. The description contains three comma-separated text strings that describe the values in the first three positions in the persistence status code.

The transformation uses the sequence identifier values from the source data records to compare the index data for the two data sets.

The following table describes the types of information that the transformation writes at each position in the status description and the status code:

Position	Description
1	Identifies the data set that contains the record.
2	Indicates the duplicate status of the record. The transformation looks for common sequence identifiers between the transformation input data and the index data store.
3	Describes any action that the transformation performs on the data.
4-8	The status code contains 0 at each position. The status description does not contain text for the positions.

Status Code Values and Status Description Values

The persistence status codes and the persistence status descriptions describe the relationship between the transformation input records and the records that the data store represents. The transformation uses sequence identifier values to identify the records and to determine the relationship between the records in the data sets.

The persistence status code and the persistence status descriptions have a common structure. The status codes and the status descriptions contain the same information at each position in the output data string.

Data Set Status

The first value in the status code and in the status description identifies the data set that contains the record.

The following table describes the status codes and the status descriptions that the transformation can return in the first position:

Status Code	Status Description
S	Store. The current record originates in the index data store.
I	Input. The current record originates in the transformation input data.

Duplicate Record Status

The second value in the status code and in the status description describes the relationship between the transformation index data and the persistent data store.

The following table describes the status codes and the status descriptions that the transformation can return in the second position:

Status Code	Status Description
A	Absent. The index data store does not contain data for the current record.
E	Exists. The current record is present in the index data store and in the transformation input data.
I	Invalid. The transformation cannot analyze the current record. For example, the transformation cannot generate index data for the record because the key field on the Match Type tab is not compatible with the record data.
N	New. The record is present in the data source.
0	[Dash] The record is present in the index data store.

Data Store Status

The third value in the status code and in the status description describes any action that the transformation performs on the index data tables.

The following table describes the status codes and the status descriptions that the transformation can return in the third position:

Status Code	Status Description
A	<p>Added.</p> <p>The transformation adds the index data for the current input record to the persistent data store.</p> <p>The transformation input data and the persistent index data have different sequence identifiers.</p>
I	<p>Ignored.</p> <p>The transformation does not add any index data for the current input record to the persistent data store.</p>
N	<p>The transformation returns one of the following descriptions:</p> <ul style="list-style-type: none"> - No change. The current record originates in the persistent data store, and the transformation takes no action. - Not added. The transformation does not update the persistent data store with any data for the current input record because of the match policy that you defined.
R	<p>Removed.</p> <p>The transformation removes the index data for the record from the index data store.</p>
U	<p>Updated.</p> <p>The transformation updates the rows in the persistent data store with index data from the transformation input record.</p> <p>The transformation input data and the persistent index data have common sequence identifiers.</p>

Persistence Status Description Example

The persistence status code INA00000 has the following persistence status description:

Input, New, Added

The status code and the status description contain the following information about the record:

- The record originates in the transformation input data.
- The persistent data store does not contain a copy of the record.
- The transformation adds the index data for the record to the persistent data store.

Output Ports and Match Output Selection

The match output options that you select determine the output ports on the transformation. For example, the transformation creates a ClusterId port and ClusterSize port when you select a clustered output type.

Select the type of transformation output that you need, and review the ports on the transformation.

If you update the match output type, verify the output port configuration on the transformation after you do so. If you use the transformation in a mapping, you might need to reconnect the output ports to the downstream objects in the mapping.

Match Mapplets

A match mapplet is a type of mapplet that you can create and embed in Match transformations.

You create match mapplets by saving the configuration of a Match transformation as a match mapplet. When you create a match mapplet, you convert Match transformation settings into Comparison and Weighted Average transformations.

After you create a match mapplet, you can add transformations to customize the match analysis. For example, you can add an Expression transformation to evaluate the link scores of two strategies and choose the highest score.

Unlike Match transformations, match mapplets are passive, which means that you can use them as rules within the Analyst tool. Use match mapplets in the Analyst tool to match records as part of data profiling processes.

The Match transformation can only read match mapplets that you create from within a Match transformation.

Creating a Match Mapplet

Create a match mapplet to define a match analysis operation that uses multiple transformations.

1. Open a Match transformation in the editor and select the **Strategies** view.
2. Select **Use Match Rule**.
3. In the **Name** field, select **Create new**.
The **New Mapplet** window opens.
4. In the **New Mapplet** window, enter a name for the mapplet and choose a location to save the mapplet.
5. Optionally, select **Reuse Strategies from the Match transformation** to copy the inputs, strategies, and weights from the current Match transformation to the match mapplet.
Note: Informatica recommends using this setting to quickly create match mapplets that replicate the match functionality currently defined in Match transformations.
6. Click **Finish**.
The match mapplet opens in the editor.
7. Optionally, create match operations by adding and configuring Comparison transformations and Weighted Average transformations in the match mapplet.
8. Click **File > Save** to save the mapplet.
9. Close the mapplet and select the editor that contains the Match transformation. Verify that the mapplet you created appears in the **Name** field.
10. Optionally, configure the match fields in the mapplet by clicking the **Match Fields** button.
The **Configure Match Rule** window opens.
11. Double-click the fields in the **Input Fields** and **Available Inputs** columns to assign input ports to match inputs.
12. Click **File > Save** to save the transformation.

Using a Match Mapplet

You can select and configure a previously defined match mapplet in the Match transformation.

1. Open a Match transformation in the editor and select the **Strategies** view.

2. Select **Use Match Rule**.
3. In the **Name** field, select **Use Existing**.
The **Configure Match Rule** window opens.
4. Click **Browse** to locate a match maplet in the repository.
Important: You can only select maplets created by the Match transformation.
The **Select Match Maplet** window opens.
5. Select a match maplet and click **OK**.
6. Double-click the fields in the **Input Fields** and **Available Inputs** columns to assign input ports to match inputs.
7. Click **OK**.
The **Configure Match Rule** window closes.
8. Click **File > Save** to save the Match transformation.

Configuring a Match Analysis Operation

To configure a match operation, connect source data to the Match transformation and edit the properties in the transformation views.

1. Create a Match transformation and connect source data to the transformation.
2. Select the **Match Type** view and choose a match type.
3. Configure the properties for the type of match operation that you select.
If you selected a dual source match type, configure the **Master Data Set** property.
4. Select the **Strategies** view and choose **Define match strategies**.
5. Click **New**.
The **New Match Strategy** wizard opens.
6. Choose a match strategy and click **Next**.
7. Optionally, edit the weight and null match settings. Click **Next**.
8. Double-click the cells in the Available column to select the input ports to analyze.
Click **Next** to configure another strategy, or click **Finish** to exit the wizard.
Note: To edit the strategy configuration, click the arrow in the cell for that strategy in the **Strategies** view.
9. Select the **Match Output** view.
Choose a match output type and configure the properties.

Note: You can also configure match strategies by selecting or creating a match maplet in the **Strategies** view. A match maplet is a type of maplet that you can embed in a Match transformation.

Match Transformation in a Non-native Environment

The Match transformation processing in the non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Match Transformation on the Blaze Engine

The Match transformation is supported with the following restrictions:

- A Match transformation generates cluster ID values differently in native and non-native environments. In a non-native environment, the transformation appends a group ID value to the cluster ID.
- Mapping validation fails when the Match transformation is configured to write identity index data to database tables.

Match Transformation on the Spark Engine

The Match transformation is supported with the following restrictions:

- A Match transformation generates cluster ID values differently in native and non-native environments. In a non-native environment, the transformation appends a group ID value to the cluster ID.
- Mapping validation fails when the Match transformation is configured to write identity index data to database tables.

Match Transformation on the Databricks Spark Engine

The Match transformation is supported with the following restrictions:

- A Match transformation generates cluster ID values differently in native and non-native environments. In a non-native environment, the transformation appends a group ID value to the cluster ID.
- Mapping validation fails when the Match transformation is configured to write identity index data to database tables.

CHAPTER 30

Match Transformations in Field Analysis

This chapter includes the following topics:

- [Field Match Analysis, 465](#)
- [Process Flow for Field Match Analysis, 465](#)
- [Field Match Type Options, 466](#)
- [Field Match Strategies, 466](#)
- [Field Match Output Options, 469](#)
- [Field Match Advanced Properties, 471](#)
- [Field Match Analysis Example, 471](#)

Field Match Analysis

Perform field match analysis to find similar or duplicate records in a data set or between two data sets.

When you configure the Match transformation for field match analysis, you set the options on the following views:

- Match Type
- Strategies
- Match Output

Optionally, set the options on the Parameters and Advanced views.

Process Flow for Field Match Analysis

The following process flow summarizes the steps that you take to configure a Match transformation for field match analysis. You can define a process that uses the Match transformation alone or that uses the Match transformation and other transformations.

Note: When you add a Match transformation to a mapping in field match analysis, add an upstream Key Generator transformation to the mapping.

To prepare the data for the Match transformation, perform the following steps:

1. Organize the source data records into groups.
Use a Key Generator transformation to assign a group key value to each record. The group assignments reduce the number of computations that the Match transformation must perform.
2. Verify that the data source records contain unique sequence identifier values. You can use a Key Generator transformation to create the values.

Perform the following steps in the Match transformation:

1. Specify field analysis as the match type, and specify the number of data sources.
If you configure the transformation to analyze two data sets, select a master data set.
Use the **Match Type** view to set the type and the number of data sources.
2. Define a match analysis strategy. Select an algorithm, and assign a pair of columns to the algorithm.
Use the **Strategies** view to define the strategy.
3. Specify the method that the transformation uses to generate the match analysis results.
4. Set the match threshold value. The match threshold is the minimum score that can identify two records as duplicates of one another.
Use the **Match Output** view to select the output method and the match threshold.
Note: You can set the match threshold in a Match transformation or a Weighted Average transformation. Use the Weighted Average transformation if you create a match maplet.

Field Match Type Options

The Match Type view contains a single option that applies to dual-source match analysis. The option identifies the master data set. The Match Type view does not contain an option for single-source match analysis.

When you analyze two data sets, you must select one as a master data set. If neither data set represents a master data set in the project or the organization, select the larger data set as the master data set.

Use the **Master Data Set** option to specify the master data set.

Field Match Strategies

The Strategies view lists the strategies that you define for the input data.

The strategies determine how the transformation measures the similarities and differences between the data source records.

Field Match Algorithms

The Match transformation includes algorithms that compare data values across two columns. Each algorithm calculates the degree of difference between data values in a different way.

Select an algorithm that can measure the types of data difference that you expect to find in the columns that you select.

Bigram

Use the Bigram algorithm to compare long text strings, such as postal addresses entered in a single field.

The Bigram algorithm calculates a match score for two data strings based on the occurrence of consecutive characters in both strings. The algorithm looks for pairs of consecutive characters that are common to both strings. It divides the number of pairs that match in both strings by the total number of character pairs.

Bigram Example

Consider the following strings:

- larder
- lerder

These strings yield the following Bigram groups:

```
l a, a r, r d, d e, e r  
l e, e r, r d, d e, e r
```

Note that the second occurrence of the string "e r" within the string "lerder" is not matched, as there is no corresponding second occurrence of "e r" in the string "larder".

To calculate the Bigram match score, the transformation divides the number of matching pairs (6) by the total number of pairs in both strings (10). In this example, the strings are 60% similar and the match score is 0.60.

Hamming Distance

Use the Hamming Distance algorithm when the position of the data characters is a critical factor, for example in numeric or code fields such as telephone numbers, ZIP Codes, or product codes.

The Hamming Distance algorithm calculates a match score for two data strings by computing the number of positions in which characters differ between the data strings. For strings of different length, each additional character in the longest string is counted as a difference between the strings.

Hamming Distance Example

Consider the following strings:

- Morlow
- Marlowes

The highlighted characters indicate the positions that the Hamming algorithm identifies as different.

To calculate the Hamming match score, the transformation divides the number of matching characters (5) by the length of the longest string (8). In this example, the strings are 62.5% similar and the match score is 0.625.

Edit Distance

Use the Edit Distance algorithm to compare words or short text strings, such as names.

The Edit Distance algorithm calculates the minimum "cost" of transforming one string to another by inserting, deleting, or replacing characters.

Edit Distance Example

Consider the following strings:

- Levenston
- Levenshtein

The highlighted characters indicate the operations required to transform one string into the other.

The Edit Distance algorithm divides the number of unchanged characters (8) by the length of the longest string (11). In this example, the strings are 72.7% similar and the match score is 0.727.

Jaro Distance

Use the Jaro Distance algorithm to compare two strings when the similarity of the initial characters in the strings is a priority.

The Jaro Distance match score reflects the degree of similarity between the first four characters of both strings and the number of identified character transpositions. The transformation weights the importance of the match between the first four characters by using the value that you enter in the Penalty property.

Jaro Distance Properties

When you configure a Jaro Distance algorithm, you can configure the following properties:

Penalty

Determines the match score penalty if the first four characters in two compared strings are not identical. The transformation subtracts the full penalty value for a first-character mismatch. The transformation subtracts fractions of the penalty based on the position of the other mismatched characters. The default penalty value is 0.20.

Case Sensitive

Determines whether the Jaro Distance algorithm considers character case when it compares characters.

Jaro Distance Example

Consider the following strings:

- 391859
- 813995

If you use the default Penalty value of 0.20 to analyze these strings, the Jaro Distance algorithm returns a match score of 0.513. This match score indicates that the strings are 51.3% similar.

Reverse Hamming Distance

Use the Reverse Hamming Distance algorithm to calculate the percentage of character positions that differ between two strings, reading from right to left.

The Hamming Distance algorithm calculates a match score for two data strings by computing the number of positions in which characters differ between the data strings. For strings of different length, the algorithm counts each additional character in the longest string as a difference between the strings.

Reverse Hamming Distance Example

Consider the following strings, which use right-to-left alignment to mimic the Reverse Hamming algorithm:

- 1-999-9999
- **011-01**-999-9991

The highlighted characters indicate the positions that the Reverse Hamming Distance algorithm identifies as different.

To calculate the Reverse Hamming match score, the transformation divides the number of matching characters (9) by the length of the longest string (15). In this example, the match score is 0.6, indicating that the strings are 60% similar.

Field Match Strategy Properties

Open the **Strategy** wizard on the **Strategies** view and configure the properties for each field match strategy.

When you configure a field match strategy, you can configure the following properties:

Name

Identifies the strategy by name.

Weight

Determines the relative priority assigned to the match score when the overall score for the record is calculated. Default is 0.5.

Single Field Null

Defines the match score that the algorithm applies to a pair of data values when one value is null. Default is 0.5.

Both Fields Null

Defines the match score that the algorithm applies to a pair of data values when both values are null. Default is 0.5.

Note: A match algorithm does not calculate a match score when one or both of the matched column values are null. The algorithm applies the scores defined in the null match properties. You cannot clear the null match properties.

Field Match Output Options

Configure the **Match Output** options to define the output format for field match analysis.

You configure the options in the **Match Output Type** area and the **Properties** area.

Match Output Types

The Match Output view includes options that specify the output data format. You can configure the transformation to write records in clusters or in matched pairs.

Select one of the following match output types:

Best Match

Writes each record in the master data set with the record that represents the best match in the second data set. The match operation selects the record in the second data set that has the highest match score for the master record. If two or more records return the highest score, the match operation selects the first record in the second data set. Best Match writes each pair of records to a single row.

You can select **Best Match** when you configure the transformation for dual-source analysis.

Clusters

Writes clusters that contain sets of records that link to each other with match scores that meet the match threshold. Each record must match at least one other record in the cluster with a score that meets the threshold.

You can select **Clusters** when you configure the transformation for single-source analysis and dual-source analysis.

Matched Pairs

Writes all pairs of records that match each other with a score that meets the match threshold. The transformation writes each pair to a single row and adds the match score for each pair to each row. If a record matches more than one other record, the transformation writes a row for each record pair.

You can select **Matched Pairs** when you configure the transformation for single-source and dual-source analysis.

Match Output Properties

The Match Output view includes properties that specify the cache memory behavior, the match score threshold, and the match scores that appear in the transformation output.

You can also use the match output properties to specify how the transformation adds match score values to the output records.

After you select a match output type, configure the following properties:

Cache Directory

Specifies the directory to which the Data Integration Service writes temporary data during field match analysis. The Data Integration Service writes temporary files to the directory when the volume of data that the match analysis generates is greater than the available system memory. The Data Integration Service deletes the temporary files after the mapping runs.

You can enter a directory path on the property, or you can use a parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory. The default value is the CacheDir system parameter.

Cache Size

Determines the amount of system memory that the Data Integration Service assigns to field match analysis. The default value is 400,000 bytes.

Before it sorts the data, the Data Integration Service allocates the amount of memory that you specify. If the match analysis generates a greater amount of data, the Data Integration Service writes the excess data to the cache directory. If the match analysis requires more memory than the system memory and the file storage can provide, the mapping fails.

Note: If you enter a value of 65536 or higher, the transformation reads the value in bytes. If you enter a lower value, the transformation reads the value in megabytes.

Threshold

Sets the minimum match score that identifies two records as potential duplicates of each other.

You can assign a parameter to the threshold value. Set a decimal value in the range 0 through 1.

Scoring Method

Determines the match score values that appear in the transformation output. Select a scoring method for cluster outputs.

The following table describes the scoring method options:

Scoring Method Option	Description
Both	Adds the link score and the driver score to each record in the cluster.
Link Score	Adds the link score to each record in the cluster. Default option.

Scoring Method Option	Description
Driver Score	Adds the driver score to each record in the cluster.
None	Does not add a match score to any record in the cluster.

Note: If you add the driver score to the records, you increase the mapping run time. The mapping waits until all clusters are complete before it adds the driver score values to the records.

Field Match Advanced Properties

The transformation includes advanced properties that determine the number of execution instances, how the transformation analyzes identical rows, and the tracing level for log data.

You can configure the following advanced properties:

Execution Instances

Determines the number of threads that the transformation uses at run time.

The Match transformation uses a single execution instance in field match analysis. You can edit the number of execution instances when you configure the transformation for identity match analysis.

Filter Exact Match

Determines whether the transformation applies the comparison algorithm in a match strategy to pairs of identical records in the input data.

When the transformation finds a pair of identical records, the algorithm does not need to analyze the levels of similarity between the records. The transformation can pass the records directly to the output stage without additional analysis. To configure the transformation to pass the identical records directly to the output stage, select Filter Exact Match. When the input data contains many identical rows, the comparison algorithm performs fewer calculations, and the mapping runs faster.

Select the option when the input data contains many identical rows. Do not select the option if the input data does not contain many identical rows, as the transformation might run more slowly.

Note: The transformation output contains the same record data when you select or clear the option. The transformation might assign different link scores and driver scores to the output records when you select and clear the option.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Field Match Analysis Example

You are a data steward at a retail bank. You receive a set of account records for the customers who opened bank accounts in the past seven days. You want to verify that the data set does not contain duplicate records. You design a mapping to search the records for duplicate data.

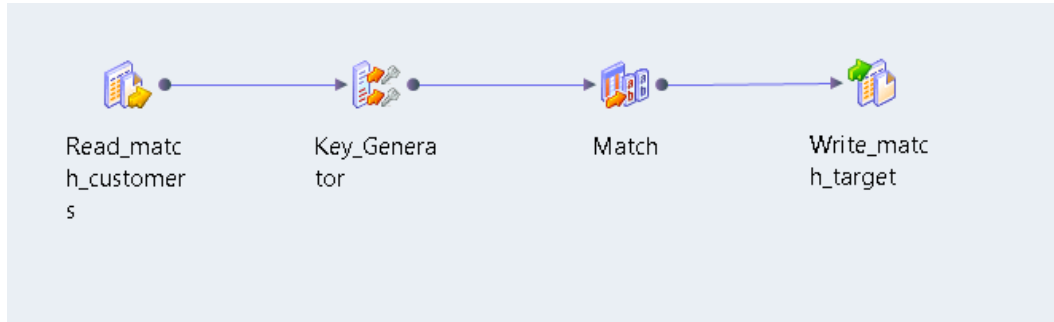
Create the Mapping

Create a mapping that looks for duplicate data in multiple fields.

The mapping performs the following tasks:

- It reads a data source.
- It adds a group key value and sequence identifier value to the source records.
- It analyzes the field data in the records.
- It writes the results to a data target.

The following image shows the mapping in the Developer tool:



The mapping that you create contains the following objects:

Object Name	Description
Read_Match_Customers	Data source. Contains the account holder details, including names, addresses, and account numbers.
Key_Generator	Key Generator transformation. Adds group key values and sequence identifier values to the source data.
Match	Match transformation. Analyzes the levels of duplication in the account data.
Write_match_target	Data target. Contains the results of the field analysis.

Input Data Sample

The data set contains the account number, name, address, and employer for each customer. You create a data source from the data set in the Model repository. You add the data source to the mapping.

The following data fragment shows a sample of the customer account data:

CustomerID	Lastname	City	State	ZIP
15954467	JONES	SCARSDALE	NY	10583
10110907	JONES	MINNEAPOLIS	MN	55437

CustomerID	Lastname	City	State	ZIP
19131127	JONES	INDIANAPOLIS	IN	46240
10112097	JONES	HOUSTON	TX	77036
19133807	JONES	PLANTATION	FL	33324
10112447	JONES	SCARSDALE	NY	10583
15952487	JONES	HOUSTON	TX	77002
10112027	JONES	OAKLAND	CA	94623

Key Generator Transformation Configuration

When you configure the Key Generator transformation, connect the data source ports that you want to analyze. Specify the port that contains the group key data. If the records do not include a unique identifier, use the sequence ID port to add unique identifiers to the record.

When you specify the group key port, consider the following guidelines:

- Select a port that contains values that repeat regularly in the port data. Preferably, select a port that creates groups that are similar in size.
- Select a port that is not relevant to the duplicate analysis.

In the current example, select the City port as the group key. If an account name appears more than once in a city, the accounts might contain duplicate data. If an account name appears more than once in different cities, the accounts are unlikely to be duplicates.

Tip: Run a column profile on the data source before you select the group key port. The profile results can indicate the number of times each value appears on a port.

Match Transformation Configuration

Add a nonreusable Match transformation to the mapping to perform the field analysis.

Complete the following tasks to configure the Match transformation:

1. Select the type of match analysis to perform.
2. Connect the input ports to the transformation.
3. Configure strategies to compare the record data.
4. Select the type of match output data that the transformation creates.
5. Connect the output ports to a data target.

Select the Type of Match Operation

Use the options on the **Match Type** view to select the match operation. To compare the account data, configure the transformation to perform single-source field analysis.

Connect the Input Ports

Connect the customer account data ports to the Match transformation.

The Match transformation uses preset input ports to determine the order in which it processes the records. The transformation uses a sequence identifier to track the records from the input ports to the matched pairs or clusters that it writes as output. The transformation uses a group key to sort the records that it processes.

Connect the following preset ports on the Match transformation to the preset output ports on the Key Generator transformation:

- SequenceID
- GroupKey

Configure the Strategies for Field Analysis

Use the options on the **Strategies** view to configure the strategies. The strategies determine the types of analysis that the transformation performs on the record data.

Create the following strategies:

- Create a strategy that uses the Edit Distance algorithm to analyze the customer identification numbers. Select the CustomerID_1 and CustomerID_2 ports.
- Create a strategy that uses the Jaro Distance algorithm to analyze the surname data. Select the Lastname_1 and Lastname_2 ports.
Note: The Jaro Distance algorithm applies an additional penalty to similar strings that start with different characters. Therefore, the Jaro Distance algorithm can apply a high match score to PATTON and PATTEN but apply a lower match score to BAYLOR and TAYLOR.
- Create a strategy that uses the Reverse Hamming Distance algorithm to analyze the ZIP code data. Select the Zip_1 and Zip_2 ports.

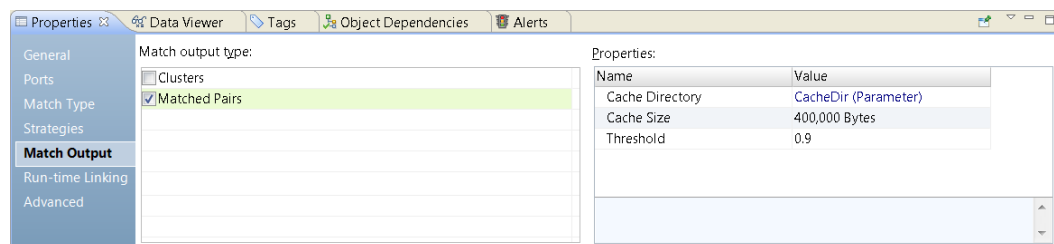
RELATED TOPICS:

- [“Field Match Algorithms” on page 466](#)

Select the Match Output Type

Use the options on the **Match Output** view to define the output format for the results of the match analysis.

The following image shows the Match Output view for single-source field analysis:



You configure the transformation to organize the output records in matched pairs. Because the transformation returns records in pairs, the results of the analysis do not include unique records.

Connect the Output Ports

Connect the Match transformation output ports to the data target in the mapping. Select the ports that contain the record data that you want to write to the data target.

The transformation includes preset ports that identify the records in each matched pair. The preset port names are **RowId** and **RowId1**. Each row ID value identifies a record in the output data.

The row ID values correspond to the ports that you select in a match strategy. When you configure a strategy, you select port names with the suffix **_1** or **_2**. The RowId value identifies the record that contains a port with the suffix **_1**. The RowId1 value identifies the record that contains a port the suffix **_2**.

You can use other output ports to review the relationships between the records. The link port values and driver port values indicate the extent of the similarity between the records in each cluster.

In the current example, you connect all the ports to the data target. To view the output data on the ports, run the Data Viewer.

Run the Data Viewer

Run the Data Viewer to review the results of the match analysis. By default, the Data Viewer shows all the output ports on the Match transformation. When you run the mapping, you update the data target with the data from the output ports.

The following image shows the output data in the Data Viewer:

The screenshot shows a software interface with a workflow diagram at the top and a Data Viewer window below. The workflow consists of four steps: Read_matc_h_customer_s_13, Key_Generator, Match, and Write_matc_h_target. The Data Viewer window is titled 'Field_Match' and shows the output of the Match transformation. The output table has 12 columns: Lastname_1, Firstname_1, Company_1, Address_1_1, City_1, State_1, Zip_1, CustomerID_1, Lastname_2, Firstname_2, Company_2, Address_1_2, City_2. The table contains 4 rows of data:

	Lastname_1	Firstname_1	Company_1	Address_1_1	City_1	State_1	Zip_1	CustomerID_1	Lastname_2	Firstname_2	Company_2	Address_1_2	City_2
1	Chan	David	SANDS BR...	515 FOLS...	SAN FRAN...	CA	94105	15954522	Chan	Roy	PG&E (PA...	77 BEALE...	SAN FRAN...
2	Andersen	Keith	CHARLES...	1100 N.E....	SAN ANT...	TX	78209	10110428	Andersen	Keith	CHARLES...	1100 N.E....	SAN ANT...
3	Abrary	David	AIRFINAN...	488 MADL...	NEW YORK	NY	10022	15954929	Abrary	Julianne	TRANSOC...	39 EAST 5...	NEW YORK
4	Chan	Agusta	J. STREICH...	86 TRINIT...	NEW YORK	NY	10006	19132518	Chan	Carmen	AMERICA...	86 TRINIT...	NEW YORK

The Data Viewer verifies that the customer account data includes one or more duplicate records.

Consider the following data in the Data Viewer:

- The transformation determines that the records for Augusta Chan and Carmen Chan might contain the same information because they contain the same surname and address data. When you review the records, you decide that the records are unique in the data set. However, you notice that the records share a common customer ID value. Because the customer ID column is a primary key in the data set, you contact the New York office. The New York office resolves the error.
- The transformation determines that the records for Keith Anderson might contain the same information. When you review the records, you verify that the two records represent the same account. However, you notice that the records have different customer ID values. Because a customer account must have a single ID value, you contact the San Antonio office. The San Antonio office resolves the error.

Conclusion

The results of the match analysis indicate that the customer account data set includes at least one pair of duplicate records. You contact the local branch for the accounts that you need to verify. You verify that the other records in the data set uniquely identify a customer account.

CHAPTER 31

Match Transformations in Identity Analysis

This chapter includes the following topics:

- [Identity Match Analysis, 477](#)
- [Process Flow for Identity Match Analysis, 478](#)
- [Identity Match Type Properties, 478](#)
- [Identity Match Strategies, 482](#)
- [Identity Match Output Options, 484](#)
- [Identity Match Advanced Properties, 487](#)
- [Persistent Index Case Study, 488](#)
- [Identity Match Analysis Example, 490](#)

Identity Match Analysis

Perform identity match analysis to find similar or duplicate identities in a data set or between two data sets.

Similar identities in two or more records might indicate record duplication. Or, the similar identities might indicate a connection between the records, such as a shared family identity or a shared employer identity.

When you configure the Match transformation for identity match analysis, you set the options on the following views:

- Match Type
- Strategies
- Match Output

Optionally, set the options on the Parameters and Advanced views.

When you configure a Match transformation for identity match analysis, connect input ports to all of the primary required fields in an identity strategy. Most identity strategies contain primary required fields. Some strategies also contain secondary required fields. Connect an input port to at least one secondary required field.

Process Flow for Identity Match Analysis

The following process flow summarizes the steps that you take to configure a Match transformation for identity match analysis. You can define a process that uses the Match transformation alone or that uses the Match transformation and other transformations.

Before you connect the Match transformation to upstream data objects, verify that the records contain unique sequence identifier values. You can use a Key Generator transformation to create the values. When you perform identity match analysis, you can optionally organize the input data into groups.

Perform the following steps in the Match transformation:

1. Specify identity analysis as the match type, and specify the number of data sources.
If you configure the transformation to analyze two data sets, select a master data set.
Use the **Match Type** view to set the type and the number of data sources.
2. Identify the location to store the index data. The transformation can write the index data to temporary files or save the index data to database tables.
Use the **Match Type** view to specify the index data store.
3. Define a match analysis strategy. Select a population and a comparison algorithm, and assign a pair of columns to the algorithm.
The population indicates the column pairs to select.
Use the **Strategies** view to define the strategy.
4. Specify the method that the transformation uses to generate the match analysis results.
5. Set the match threshold value. The match threshold is the minimum score that can identify two records as duplicates of one another.
Use the **Match Output** view to select the output method and the match threshold.
Note: You can set the match threshold in a Match transformation or a Weighted Average transformation. Use the Weighted Average transformation if you create a match mapplet.

Identity Match Type Properties

Use the Match Type view to specify the type of analysis that the Match transformation performs and to set the properties that define the analysis. You can specify single-source analysis or dual-source analysis. You can also specify a persistent data store for the identity index data.

The properties that you configure depend on the type of analysis that you select. Many of the options are common to all types of analysis.

Common Properties

The following properties are common to all types of identity analysis:

Population

Identifies the population file that the transformation uses. The population file contains the key-building algorithms that generate the index keys.

Key Level

Determines the number of keys that the identity algorithms generate. The default setting is Standard. The Limited setting results in a lower number of keys, higher accuracy, and longer processing time. The Extended setting results in a higher number of keys, lower accuracy, and shorter processing time.

Key Type

Describes the type of information that the key field contains. Identity analysis can generate keys for person names, organizations, and addresses. Select the key type that best describes the column that you specify on **Key Field** property.

Search Level

Indicates the balance of search depth and search speed that the transformation applies to the match analysis. The search depth correlates inversely to the number of matches returned. For example, the **Exhaustive** option returns fewer matches.

Key Field

Specifies the column that the Match transformation uses to generate the index key data. Verify that the column that you select contains the type of information that you specified on the **Key Type** property.

Index Directory

Identifies the directory to which the Data Integration Service writes index key data for the current transformation. By default, the property is blank. If you do not specify an index directory, the Data Integration Service uses the location that you set on the Content Management Service.

You can enter a path to the directory, or you can use a parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory.

Cache Directory

Identifies the directory to which the Data Integration Service writes temporary data during the index creation stage of identity match analysis. Update the property to specify a location for data from the current transformation. By default, the property is blank. If you do not specify a cache directory, the Data Integration Service uses the location that you set on the Content Management Service.

You can enter a path to the directory, or you can use a parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory.

Cache Size

Determines the amount of system memory that the Data Integration Service assigns to identity index creation. The default value is 400,000 bytes.

If the index creation operation generates a greater amount of data, the Data Integration Service writes the excess data to the cache directory. If the operation requires more memory than the system memory and the file storage can provide, the mapping fails.

Note: If you enter a value of 65536 or higher, the transformation reads the value in bytes. If you enter a lower value, the transformation reads the value in megabytes.

Dual-Source Properties

Set the following property in addition to the common properties when you configure the transformation for dual-source analysis:

Master Data Set

Identifies the data source that contains the master data. Specify a master data set in dual-source analysis.

Persistent Data Storage Properties

Set the following properties in addition to the common properties when you configure the transformation to use the persistent index data store:

Persistence Method

Specifies whether the transformation updates the current index tables with index data from the mapping data source. Select one of the following options:

- Update the database with new IDs.
The transformation adds all rows to the index data that do not duplicate a sequence identifier in the index data. The transformation does not update the current rows in the index.

By default, the transformation performs match analysis when you select the option. You can use the Matching Process option to enable or disable the match analysis.
- Do not update the database.
The transformation does not update the index tables with index data from the mapping data source.

The transformation performs match analysis when you select the option.
- Remove IDs from the database.
The transformation deletes rows from the index tables if the rows share sequence identifiers with the mapping source data.

The transformation does not perform match analysis when you select the option.
- Update the current IDs in the database.
The transformation replaces rows in the index tables with rows from the mapping source data if the rows share sequence identifiers. The transformation does not add rows to the index.

By default, the transformation performs match analysis when you select the option. You can use the Matching Process option to enable or disable the match analysis.

The default persistence method is **Update the database with new IDs**.

Matching Process

Determines whether the current transformation performs identity analysis.

The option that you select on the Persistence Method property determines the options on the Matching Process property.

DB Connection

Identifies the database that contains the index tables.

Persistent Store

Identifies the index tables within the database that you specify.

RELATED TOPICS:

- [“Persistent Index Case Study” on page 488](#)
- [“Persistence Method Parameters” on page 481](#)

Index Directory and Cache Directory Properties

The index directory and the cache directory store the temporary data that the Match transformation generates during identity analysis.

The Data Integration Service writes data to the index directory if you do not configure the transformation to write the index data to database tables. The Data Integration Service writes data to the cache directory if the

identity analysis requires a greater amount of memory than the cache size specifies. By default, the properties are blank on the Match Type view. If you do not specify an index directory or a cache directory, the Data Integration Service reads the directory paths from the Content Management Service.

When you specify an index directory or a cache directory on the Match Type view, enter a local path on the Data Integration Service machine. You can enter a fully qualified path or a relative path. If you enter a relative path, start the path with a period. The path is relative to the `tomcat/bin` directory on the Data Integration Service machine.

The following table shows a relative path on a cache directory or index directory property and identifies the fully qualified path that the property represents:

Relative Path	Fully Qualified Path
<code>./ch</code>	<code>[Informatica_installation_directory]/tomcat/bin/ch</code>

You can configure the index directory property and the cache directory property to identify the same directory. If you specify the same directory on each property, the Data Integration Service creates directories in the directory that you specify. The Data Integration Service creates an `index` directory for the index data and a `cache` directory for the cache data. If you specify different directories on each property, the Data Integration Service writes the data to the directories that you specify.

The Data Integration Service deletes the index files and the cache files from the directories after the mapping runs. The Data Integration Service does not delete the directories. The Data Integration Service can reuse the directories in other mappings if the Match transformation identifies the directories.

Content Management Service Properties

The Content Management Service specifies the following default locations for the index directory and the cache directory:

- `./identityIndex`. Default directory for the identity index data.
- `./identityCache`. Default directory for the identity cache data.

If you do not set the properties on the Match transformation, the Data Integration Service creates the default directories when you run an identity match mapping. The Data Integration Service creates the directories in the `tomcat/bin` directory.

Persistence Method Parameters

When you select a persistence data index in identity match analysis, you can use a parameter to identify the persistence method. Use a string to define the parameter value.

The following table lists the persistence methods that you can select on the **Match Type** tab and the parameter values that you can set for the methods:

Persistence Method	Parameter
Update the database with new IDs	<code>ignore</code>
Do not update the database	<code>addNone</code>
Remove IDs from the database	<code>remove</code>
Update the current IDs in the database	<code>update</code>

The parameter values are case-sensitive.

RELATED TOPICS:

- [“Identity Match Type Properties” on page 478](#)

Identity Match Strategies

The Strategies view lists the strategies that you define for the identity data. The strategies determine how the transformation measures the similarities and differences between the data in the identity index.

Identity Match Algorithms

The Match transformation includes predefined identity algorithms that compare the data values in the identity index. Select the algorithm that best represents the type of identity data in the data set.

The following table describes the algorithms and identifies the inputs that you select for each one:

Identity Algorithm	Description
Address	Identifies records that share an address. The algorithm requires the following primary input: - Address The algorithm does not require a secondary input.
Contact	Identifies records that share a contact at a single organization location. The algorithm requires the following primary inputs: - Person_Name - Organization_Name - Address_Part1 The algorithm does not require a secondary input.
Corp Entity	Identifies records that share corporate identification data. Optionally, select this algorithm to analyze address and telephone data. The algorithm requires the following primary input: - Organization_Name The algorithm does not require a secondary input.
Division	Identifies records that share an office location within an organization. The algorithm requires the following primary inputs: - Organization_Name - Address_Part1 The algorithm does not require a secondary input.
Family	Identifies individuals that belong to a family. Analyzes name, address, and telephone number data. The algorithm requires the following primary input: - Person_Name The algorithm requires one of the following secondary inputs: - Address_Part1 - Telephone_Number

Identity Algorithm	Description
Fields	<p>Identifies records that share data on ports that you select.</p> <p>The algorithm does not specify any required input. Select the port or ports that might contain duplicate identity data.</p>
Household	<p>Identifies individuals that belong to a household. Analyzes name data and address data.</p> <p>The algorithm requires the following primary inputs:</p> <ul style="list-style-type: none"> - Person_Name - Address_Part1
Individual	<p>Identifies duplicate individuals. Analyzes name, date of birth, and personal identification data such as Social Security numbers, account numbers, and vehicle identification numbers.</p> <p>The algorithm requires the following primary input:</p> <ul style="list-style-type: none"> - Person_Name <p>The algorithm requires one of the following secondary inputs:</p> <ul style="list-style-type: none"> - Date - ID
Organization	<p>Identifies records that share organization data.</p> <p>The algorithm requires the following primary input:</p> <ul style="list-style-type: none"> - Organization_Name <p>The algorithm does not require a secondary input.</p>
Person Name	<p>Identifies records that share information about individuals.</p> <p>The algorithm requires the following primary input:</p> <ul style="list-style-type: none"> - Person_Name <p>The algorithm does not require a secondary input.</p>
Resident	<p>Identifies duplicate individuals at an address. Optionally, configure this strategy to analyze personal identification data.</p> <p>The algorithm requires the following primary input:</p> <ul style="list-style-type: none"> - Person_Name - Address_Part1 <p>The algorithm does not require a secondary input.</p>
Wide Contact	<p>Identifies records that share a contact at an organization.</p> <p>The algorithm requires the following primary inputs:</p> <ul style="list-style-type: none"> - Person_Name - Organization_Name <p>The algorithm does not require a secondary input.</p>
Wide Household	<p>Identifies individuals that belong the same household.</p> <p>The algorithm requires the following primary input:</p> <ul style="list-style-type: none"> - Address_Part1 <p>The algorithm does not require a secondary input.</p>

Identity Match Strategy Properties

Configure the properties for each identity strategy.

When you configure an identity strategy, you can configure the following strategy properties:

Population

Determines the population to apply to identity analysis. Populations contain key-building algorithms for specific locales and languages.

Match level

Determines the balance of search quality and search speed. The search speed is inversely related to the number of matches returned. Searches that use the `Loose` setting return fewer matches, while searches that use the `Conservative` setting return more matches.

Identity Match Output Options

The Match Output view includes options that specify the output data format. You can configure the transformation to write records in clusters or in matched pairs. You can also configure the transformation to include or exclude different categories of identity when you perform identity analysis against a persistent index data store.

You configure the options in the **Match output type** area and the **Properties** area.

Match Output Types

The Match Output view includes options that specify the output data format. You can configure the transformation to write records in clusters or in matched pairs.

Select one of the following match output types:

Best Match

Writes each record in the master data set with the record that represents the best match in the second data set. The match operation selects the record in the second data set that has the highest match score for the master record. If two or more records return the highest score, the match operation selects the first record in the second data set. Best Match writes each pair of records to a single row.

You can select **Best Match** when you configure the transformation for dual-source analysis.

Clusters - Best Match

Writes clusters that represent the best match between one record and another record in the same data set or between two data sets. The match score between the two records must meet the match threshold. Best match clusters can contain more than two records if a record represents the best match with more than one other record.

You can select **Clusters - Best Match** in any type of identity analysis.

Note: The index data storage method that you select can affect the contents of the cluster output in **Clusters - Best Match** mode. A transformation that connects to index tables can create different clusters than a transformation that stores index data for the same records in temporary files. The index data storage method does not affect the match scores that the transformation generates for pairs of records.

Clusters - Match All

Writes clusters of records that match with a score that meets the match threshold. Each record must match at least one other record in the cluster.

You can select **Clusters - Match All** in any type of identity analysis.

Matched Pairs

Writes all pairs of records that match each other with a score that meets the match threshold. The transformation writes each pair to a single row and adds the match score for each pair to each row. If a record matches more than one other record, the transformation writes a row for each record pair.

You can select **Matched Pairs** in any type of identity analysis.

Match Output Properties

The Match Output view contains properties that specify the cache memory behavior and the match score threshold. You can also use the properties to determine how the transformation selects data store records for analysis and writes data store records as output.

After you select a match output type, configure the following properties:

Cache Directory

Specifies the directory to which the Data Integration Service writes temporary data during identity match analysis. The Data Integration Service writes temporary files to the directory when the volume of data that the match analysis generates is greater than the available system memory. The Data Integration Service deletes the temporary files after the mapping runs.

You can enter a directory path on the property, or you can use a system parameter to identify the directory. Specify a local path on the Data Integration Service machine. The Data Integration Service must be able to write to the directory. The default value is the CacheDir system parameter.

Cache Size

Determines the amount of system memory that the Data Integration Service assigns to identity match analysis. The default value is 400,000 bytes.

If the match analysis generates a greater amount of data, the Data Integration Service writes the excess data to the cache directory. If the match analysis requires more memory than the system memory and the file storage can provide, the mapping fails.

Note: If you enter a value of 65536 or higher, the transformation reads the value in bytes. If you enter a lower value, the transformation reads the value in megabytes.

Match

Identifies the records to analyze when the transformation reads index data from database tables. Use the options on the **Match Type** view to identify the index tables.

By default, the transformation analyzes all the records in the data source and the index database tables. Configure the Match property to specify a subset of the records for duplicate analysis.

Output

Filters the records that the transformation writes as output when you configure the transformation to read index database tables. Use the options on the **Match Type** view to identify the index tables.

By default, the Match transformation writes all records from the data source and the index database tables as output. Configure the **Output** property when you do not need to review all the records in the input data.

Threshold

Sets the minimum match score that identifies two records as potential duplicates of each other.

You can assign a parameter to the threshold value. Set a decimal value in the range 0 through 1.

Match Property Configuration

Use the **Match** property on the **Match Output** view to specify how the transformation selects input data for analysis. Configure the property when you configure the Match transformation to read a persistent store of index data. The Match property refines the options that you set on the **Match Type** view.

You can configure the Match property to perform the following types of analysis:

Compare the data source records with the index data records

To look for duplicate records between the data source and the index data tables, select **Exclusive**.

When you select the Exclusive option, the Match transformation compares the data source records with the index data store. The transformation does not analyze the records within the data source or within the data store.

Select **Exclusive** when you know that the index data store does not contain duplicate records, and you know that the data source does not contain duplicate records.

Compare the data source records with the index data records, and compare the data source records with each other

To look for duplicates in the data source and to look for duplicates between the data source and the index tables, select **Partial**.

The transformation compares the data source records with the index data store. The transformation also compares the records within the data source with each other.

Select **Partial** when you know that the index data store does not contain duplicate records, but you have not performed any duplicate analysis on the data source.

Compare all records in the data source and the index tables as a single data set

To look for duplicates between the data source and the index tables, and to look for duplicates within the data source and within the index tables, select **Full**. The default option is Full.

The transformation analyzes the data source and the data store as a single data set and compares all record data within the data set.

Select **Full** when you cannot verify that either data set is free of duplicate records.

Output Property Configuration

Use the **Output** property on the **Match Output** view to filter the records that the transformation writes as output. Configure the property when you specify index data tables and you select a clustered output format. Filter the records to limit the output to clusters that contain one or more records from the data source.

You can filter the output data in the following ways:

Write every cluster that includes a record from the data source or the index tables

Select **All Rows**. The transformation writes every cluster that contains at least one record from either the data source or the index data store. The default is All Rows.

Because a cluster can contain a single record, the output contains all records.

Write every cluster that includes a record from the data source

Select **New and Associated Rows**. The transformation writes every cluster that contains at least one record from the data source.

Because a cluster can contain a single record, the output contains all records in the data source. The clusters can also include records from the index tables.

Write every cluster from the data source

Select New Rows Only. The transformation writes the clusters that contains records from the data source. The output does not contain any record from the index tables.

Identity Match Advanced Properties

The transformation includes advanced properties that determine the number of execution instances, whether the transformation analyzes identical rows, and the tracing level for log data.

You can configure the following advanced properties:

Execution Instances

Specifies the number of threads that the Data Integration Service tries to create for the current transformation at run time. The Data Integration Service considers the Execution Instances value if you override the Maximum Parallelism run-time property on the mapping that contains the transformation. The default Execution Instances value is Auto.

The Data Integration Service considers multiple factors to determine the number of threads to assign to the transformation. The principal factors are the Execution Instances value and the values on the mapping and on the associated application services in the domain.

The Data Integration Service that runs the mapping reads the following values to determine the number of threads to use for the transformation:

- The *Maximum Parallelism* value on the Data Integration Service. Default is 1.
- Any *Maximum Parallelism* value that you set at the mapping level. Default is Auto.
- The *Execution Instances* value on the transformation. Default is Auto.

If you override the Maximum Parallelism value at the mapping level, the Data Integration Service attempts to use the lowest value across the properties to determine the number of threads.

If you use the default Maximum Parallelism value at the mapping level, the Data Integration Service ignores the Execution Instances value.

Consider the following rules and guidelines when you set the number of execution instances:

- Multiple users might run concurrent mappings on a Data Integration Service. To calculate the correct number of threads, divide the number of central processing units that the Data Integration Service can access by the number of concurrent mappings.
- When you use the default Execution Instances value and the default Maximum Parallelism values, the transformation operations are not partitionable.

Filter Exact Match

Determines whether the transformation applies the comparison algorithm in a match strategy to pairs of identical records in the input data.

When the transformation finds a pair of identical records, the algorithm does not need to analyze the levels of similarity between the records. The transformation can pass the records directly to the output stage without additional analysis. To configure the transformation to pass the identical records directly to the output stage, select Filter Exact Match. When the input data contains many identical rows, the comparison algorithm performs fewer calculations, and the mapping runs faster.

Select the option when the input data contains many identical rows. Do not select the option if the input data does not contain many identical rows, as the transformation might run more slowly.

Note: The transformation output contains the same record data when you select or clear the option. The transformation might assign different link scores and driver scores to the output records when you select and clear the option.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Persistent Index Case Study

You are a data steward at a retail bank that has multiple branches. You manage a master set of the customer account records from all of the branches. You use a set of index database tables to verify that the customer account database does not contain redundant or duplicate records.

To create and manage the index data store, you perform the following operations:

- You create the data store.
- You update the data store with the most recent data from the bank branches.

You might add account data to the data store, or you might update the current data in the data store.

- You remove obsolete records from the data store.

You understand that each operation might create duplicate records in the data store. You decide to develop a policy to analyze the branch data before you add the data to the master data store data. You use identity match analysis to analyze the branch data and to verify that the data does not create duplicate identities in the data store. You configure the persistent index options on the Match transformation to analyze the branch data and the data store.

Develop a Policy for Persistent Index Data Management

As a data steward, you define a business rule that states that the customer account data store cannot contain duplicate identities. You design an identity match mapping to analyze the branch data in a staging database before you add the data to the data store.

The operations to add the branch data to the data store can create duplicate identities in the following cases:

- The branch data contains duplicate identities.
- The branch data contains an identity that the index also contains.
- The branch data contains a newer version of an identity in the data store, and the newer version matches another identity in the index.

When you compare the staging database to the data store, select the persistent index options that reflect the duplicate record status of the branch data. Before you update the data store, you might decide to compare the branch data with the index data.

Note: You can enable and disable match analysis on some of the options. Enable match analysis to analyze the mapping data or to compare the index data store to the mapping data. Disable match analysis when you do not need to compare the data. You can also use the Match properties on the Match Output tab to include or exclude data from match analysis.

Compare a Mapping Data Source with the Index Data Store

To compare the mapping input data with the index data store and to make no change to the data store, select the following option:

- Do not update the database

The mapping compares the input data to the index data store. The mapping does not add, remove, or update any data in index data store.

You cannot disable identity match analysis when you select the option.

Because you do not update the index data, you cannot create duplicate rows in the store. Select the option from the Match properties on the Match Output tab that meets the current needs of the data project. For example, select the **Full** option. The **Full** option verifies that the mapping data does not contain duplicates and verifies that the mapping data does not add duplicates to the data store.

Note: Use the option to compare the mapping data and the data store before you update the data store. If the mapping output indicates that the mapping data does not add duplicates to the data store, run the mapping again. Select the option to update the database when you run the mapping again.

Create the Data Store and Add Rows to the Data Store

To create a data store or to add rows from the mapping data to a data store, select the following option:

- Update the database with new IDs

The mapping adds a row to the data store if the row does not share a sequence identifier with a row in the data store. The mapping does not overwrite any row in the index tables. When you specify empty database tables, the mapping writes all of the mapping index data to the tables.

You can enable or disable identity match analysis when you select the option. The option enables match analysis by default.

Because you do not update the index rows, select the **Exclusive** option or the **Partial** option from the Match properties on the Match Output tab. Use the **Exclusive** option if you verified the uniqueness of the mapping data rows in an earlier process.

Update the Rows in the Data Store

To update a current row in the data store with the mapping data, select the following option:

- Update the current IDs in the database

The mapping updates a current record in the data store if the record shares a sequence identifier with a record in the mapping data. The mapping does not add any row to the index tables.

You can enable or disable identity match analysis when you select the option. The option disables match analysis by default.

Because you do not add index rows to the index tables, select the **Full** option from the Match properties on the Match Output tab.

Note: When you update the rows in the data store, you expect to find duplicates between the mapping source data and the data store. Select the **Full** option to verify that the identity data that you add to the store does not match the current data in the store.

Remove Rows from the Data Store

To remove rows from the data store, select the following option:

- Remove IDs from the database

The mapping deletes a row from the data store if the row shares a sequence identifier with a record in the mapping data.

You can enable or disable identity match analysis when you select the option. The option disables match analysis by default.

Note: When you remove data from a data store, you change the relationships between the rows in the store. If the store contains duplicate identities, you might remove data for a driver record or a linked record in a cluster. Or, you might remove data for the best match in a matched pair. When you run the mapping again, the mapping might generate different clusters or duplicate pairs. If you remove rows from a data store that does not contain duplicate records, you cannot change the duplicate status of the records. When you run the mapping after you delete the rows, the mapping generates the same match scores for the identities that remain in the data set.

Identity Match Analysis Example

You are a human resources officer at a software organization with development centers in different cities. The organization stores personnel records for staff members in a database at the head office. The development centers hire staff at regular intervals, and the centers send you the personnel data for the staff members that they hire.

You add the personnel records to a spreadsheet file, and you use the file data to update the employee database. You are concerned that the current file might contain duplicate identities.

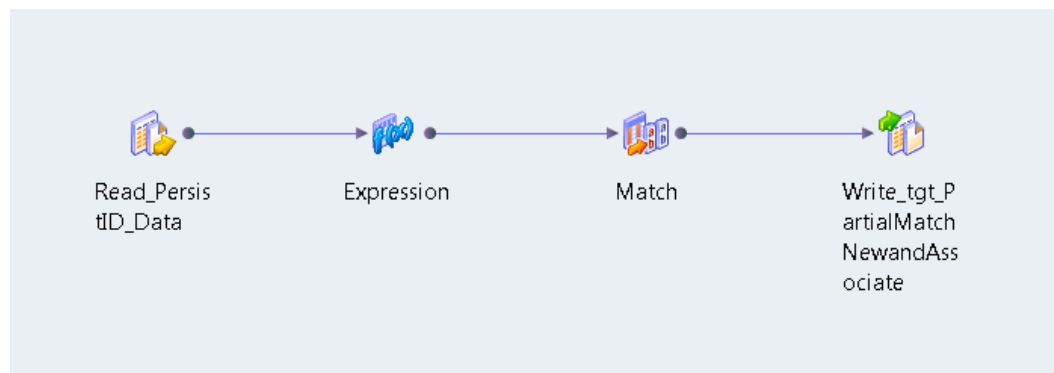
You design a mapping to perform identity analysis on the employee records. You configure a Match transformation to search for duplicate identities in the spreadsheet file. You must also verify that the file data does not duplicate any employee data in the master database. You configure the Match transformation to compare the file data with the master data that you store for the organization employees.

Because the database is a master data set, you store the index data for the staff records in a persistent data store.

Create the Mapping

Create a mapping that looks for duplicate identities. The mapping reads a data source, adds a sequence identifier to the source records, performs identity analysis, and writes the results to a data target.

The following image shows the mapping in the Developer tool:



The mapping that you create contains the following objects:

Object Name	Description
Read_PersistID_Data	Data source. Contains the employee names and details.
Expression	Expression transformation. Adds sequence identifier values to the source data.
Match	Match transformation. Analyzes the levels of duplication in the source data identities.
Write_tgt_PartialMatchNewandAssociate	Data target. Contains the results of the identity analysis.

Note: The mapping does not use a Key Generator transformation. In identity match analysis, the Key Generator transformation is optional.

Input Data Sample

The staff file contains the name of the employee, the city in which the employee works, and the designated role of the employee. You create a data source from the staff file in the Model repository. You add the data source to the mapping.

The following data fragment shows a sample of the employee data in the staff file:

Name	City	Designation
Chaithra	Bangalore	SE
Ramanan	Chennai	SSE
Ramesh	Chennai	SSE
Ramesh	Chennai	Lead
Sunil	Bangalore	Principal
Venu	Hyderabad	Principal
Harish	Bangalore	SE
Sachin	Bangalore	SSE

Expression Transformation Configuration

When you configure the Expression transformation, connect all the data source ports that you want to include in the mapping output. Connect the ports as pass-through ports. Create an expression to add a sequence identification value to the ports.

The following expression creates the variable *Init3* and adds the integer value 1267 to each sequence identifier:

Init3+1267

The following table describes the ports on the Expression transformation that reads the employee data source:

Name	Port Type	Port Group
SEQID	bigint	Output Only
Name	string	Passthrough
City	string	Passthrough
Designation	string	Passthrough
Init3	integer	Variable

Match Transformation Configuration

Add a nonreusable Match transformation to the mapping to perform the identity analysis.

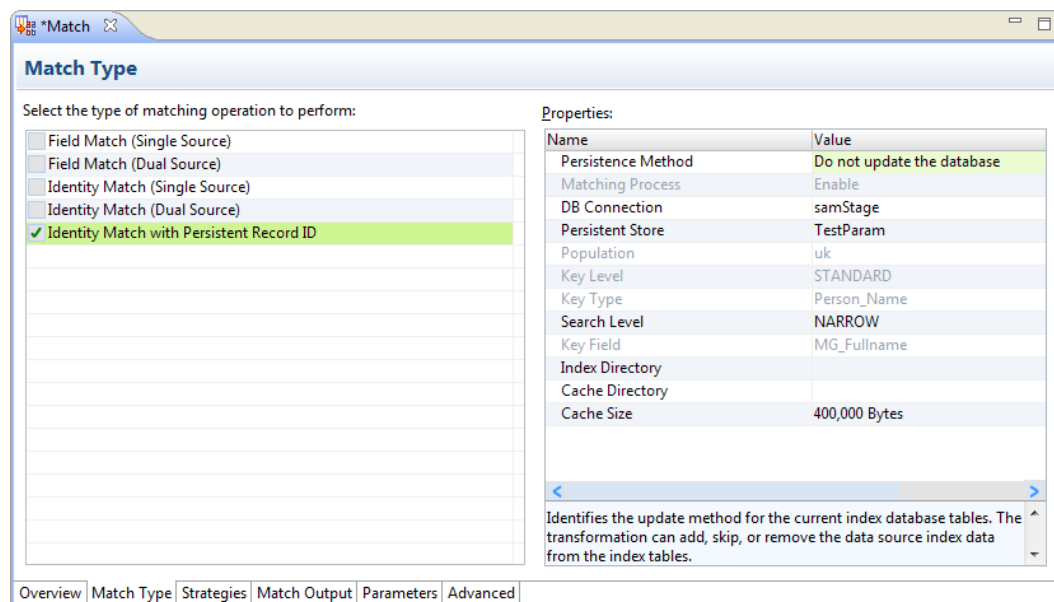
Perform the following tasks to configure the transformation:

1. Select the type of match analysis to perform.
2. Connect the input ports to the transformation.
3. Configure a strategy to compare the record data.
4. Select the type of match output data that the transformation creates.
5. Connect the output ports to a data target.

Select the Type of Match Operation

Use the options on the Match Type view to select the match operation.

The following image shows the Match Type view:



To compare the index data from the data source with the index data from the master data set, select **Identity Match with Persistent Record ID**. Update the persistence method so that the match analysis does not add data to the index tables. You can determine whether to update the index tables after you review the results of the mapping.

Use the **DB Connection** option to identify the database that contains the index data. Use the **Persistent Store** option to select the index tables.

Note: The Match transformation reads the identity population, key level, key type, and key field property values from the metadata in the index database tables. The values match the corresponding properties on the transformation that created the index data store.

Connect the Input Ports

Connect the input data ports to the transformation. Verify that the port names, port order, data types, and precisions must match the port configurations of the transformation that created the data store.

The Match transformation uses preset input ports to determine the order in which it processes the records. The transformation uses a sequence identifier to track the records from the input ports to the matched pairs or clusters that it writes as output. The transformation uses a group key to sort the records that it processes.

Connect the preset ports to the following ports on the Expression transformation:

- SequenceID. Connect to the SEQID port from the Expression transformation.
- GroupKey. Connect to the City port from the from the Expression transformation.

Configure a Strategy for Identity Analysis

Use the options on the Strategies view to configure a strategy. The strategy determines the type of analysis that the transformation performs on the record data.

Select the Person_Name algorithm for the record data. Select the Name input port for analysis. Because the transformation creates a copies of the port data, you select the Name_1 port and Name_2 port.

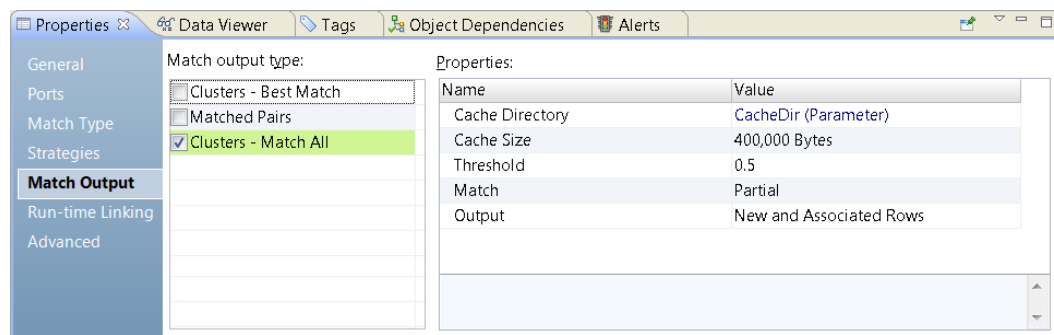
RELATED TOPICS:

- [“Identity Match Algorithms” on page 482](#)

Select the Match Output Type

Use the options on the Match Output view to define the output format for the results of the match analysis

The following image shows the Match Output view for single-source identity analysis:



You configure the transformation to organize the output records in clusters. Each cluster contains all the records that match at least one other record, based on the match properties that you specify. The match properties determine how the transformation compares the data source records to the index records.

The following table describes the match property options that you specify to analyze the staff record data:

Match Property	Option	Option Description
Match	Partial	The transformation compares the data source records with the index data store. The transformation also compares the records within the data source with each other.
Output	New and Associated Rows	The transformation writes every cluster that contains at least one record from the data source. The clusters can include records from the index data store. Because a cluster can contain a single record, the output contains all records in the data source.

RELATED TOPICS:

- [“Match Property Configuration” on page 486](#)
- [“Output Property Configuration” on page 486](#)

Connect the Output Ports

Connect the Match transformation output ports to the data target in the mapping. Select the ports that contain the record data that you want to write to the data target.

The transformation includes a series of preset ports for clustered data. Select the preset ports that indicate duplicate status of the records and identify the data source that stores each record.

The following ports contain data that you can use to find duplicate records and determine the source or the records:

- The **ClusterSize** port indicates the number of records in a cluster. If a record belongs to a cluster with a cluster size greater than 1, the transformation considers the record to be a duplicate of another record.
- The **ClusterID** port identifies the cluster that a record belongs to. Use the ClusterID data to find the records that are duplicates of the current record.
- The **PersistenceStatus** port uses a code value to describe the relationship between the index data from the mapping source and the index data in the data store.
- The **PersistenceStatusDesc** port returns a text description of the values on the PersistenceStatus port code.

You can use other ports to review the relationships between the cluster records. The link port values and driver port values indicate the extent of the similarity between the records in each cluster.

In the current example, you connect all the ports to the data target. To view the output data on the ports, run the Data Viewer.

RELATED TOPICS:

- [“Persistence Status Codes and Persistence Status Descriptions” on page 459](#)
- [“Status Code Values and Status Description Values” on page 459](#)

Run the Data Viewer

Run the Data Viewer to review the results of the match analysis. By default, the Data Viewer shows all the output ports on the Match transformation. When you run the mapping, you update the data target with the data from the output ports.

The following image shows the output data in the Data Viewer:

The screenshot shows a data flow diagram with four components: Read_IncreP, Expression, Match, and Write_tgtP. Below the diagram is the Data Viewer window, which displays a table of output data. The table has 16 rows and 10 columns. The columns are: Name, ClusterSize, PersistenceStatusDesc, Designation, GroupKey, LinkId, ClusterId, LinkScore, and PersistenceSt... The data is as follows:

Name	ClusterSize	PersistenceStatusDesc	Designation	GroupKey	LinkId	ClusterId	LinkScore	PersistenceSt...	
1	Chaith	2	Input, Exists, Ignored	SE	Bangalore	S - 1	000000001	1	IEI00000
2	Chaith	2	Store, -, No change	SE	Bangalore	1 - 1267	000000001	1	SON00000
3	Ramana	2	Input, Exists, Ignored	SSE	Chennai	S - 2	000000002	1	IEI00000
4	Ramana	2	Store, -, No change	SSE	Chennai	1 - 2534	000000002	1	SON00000
5	Ramesh	4	Input, Exists, Ignored	SSE	Chennai	S - 3	000000003	1	IEI00000
6	Ramesh	4	Input, Exists, Ignored	Lead	Chennai	S - 3	000000003	1	IEI00000
7	Ramesh	4	Store, -, No change	SSE	Chennai	1 - 3801	000000003	1	SON00000
8	Ramesh	4	Store, -, No change	Lead	Chennai	1 - 3801	000000003	1	SON00000
9	Sunil	2	Input, Exists, Ignored	Principal	Bangalore	S - 5	000000004	1	IEI00000
10	Sunil	2	Store, -, No change	Principal	Bangalore	1 - 6335	000000004	1	SON00000
11	Venu	2	Input, Exists, Ignored	Principal	Hydrabad	S - 6	000000005	1	IEI00000
12	Venu	2	Store, -, No change	Principal	Hydrabad	1 - 7602	000000005	1	SON00000
13	Harish	2	Input, Exists, Ignored	SE	Bangalore	S - 7	000000006	1	IEI00000
14	Harish	2	Store, -, No change	SE	Bangalore	1 - 8869	000000006	1	SON00000
15	Sachin	2	Input, Exists, Ignored	SSE	Bangalore	S - 8	000000007	1	IEI00000
16	Sachin	2	Store, -, No change	SSE	Bangalore	1 - 10136	000000007	1	SON00000

The Data Viewer verifies that the file contains records that duplicate the data in the master data set.

Consider the following data values in the Data Viewer:

- Each record in the data set belongs to a cluster that contains two or more records. Therefore, each record is a duplicate of at least one other record. The transformation assigns a cluster size of 1 to unique records. The data source does not contain any record that the master data set does not contain.
- The **PersistenceStatusDesc** data identifies the origin of the record and indicates if the Match transformation adds the record to the index tables. The column indicates that each input record exists in the master data set. The transformation does not add data to the master data index.

Conclusion

The results of the match analysis indicate that the staff record file does not contain any record that the master data set does not contain. The persistence status descriptions indicate that the mapping does not update the index tables with any data from the data source. You discard the staff record file.

When you receive another update from the regional offices, you can create another file and compare it to the master data set. You can reuse the mapping and the index tables. Because you store the index data for the master data set in the database tables, you do not need to regenerate the index data.

CHAPTER 32

Normalizer Transformation

This chapter includes the following topics:

- [Normalizer Transformation Overview, 497](#)
- [Multiple-Occurring Fields, 498](#)
- [Multiple-Occurring Records, 499](#)
- [Input Hierarchy Definition, 500](#)
- [Normalizer Transformation Output Groups and Ports, 508](#)
- [Key Generation for Output Groups, 512](#)
- [Normalizer Transformation Advanced Properties, 512](#)
- [Creating a Normalizer Transformation, 513](#)
- [Creating a Normalizer Transformation from an Upstream Source, 514](#)
- [Normalizer Mapping Example, 514](#)
- [Normalizer Transformation in a Non-native Environment, 518](#)

Normalizer Transformation Overview

The Normalizer transformation is an active transformation that transforms one source row into multiple target rows. When the Normalizer transformation receives a row that contains multiple-occurring data, it returns a row for each instance of the multiple-occurring data.

The Normalizer transformation parses multiple-occurring data into separate output rows. For example, a relational source row might contain four quarters of sales. The Normalizer transformation generates a separate output row for each sales occurrence.

When you define the Normalizer transformation, you configure an input row hierarchy that describes the source data structure. Optionally, you can define records in the transformation input hierarchy. A record is a container for a group of fields. Define a record when a group of fields occurs multiple times in the source data. The input hierarchy determines how you can configure the transformation output groups.

The Normalizer transformation transforms data from relational tables or from flat file sources.

Multiple-Occurring Fields

When a field repeats multiple-times in the source data, you can define the field as a multiple-occurring field in the input row hierarchy. The Normalizer transformation can return a separate row for each occurrence of a multiple-occurring field or group of fields in a source.

A source row might contain four quarters of sales by store:

Store	Sales (1)	Sales (2)	Sales (3)	Sales (4)
Store1	100	300	500	700
Store2	250	450	650	850

When you define the Normalizer input hierarchy, you can combine the four Sales fields into one multiple-occurring field. Define a field name such as Qtr_Sales and configure it to occur four times in the source.

When the output group contains the store data and the sales data, the Normalizer transformation returns a row for each Store and Qtr_Sales combination. The output row contains an index that identifies which instance of Qtr_Sales that is in the output row.

The transformation returns the following rows:

Store	Qtr_Sales	Qtr (GCID)
Store1	100	1
Store1	300	2
Store1	500	3
Store1	700	4
Store2	250	1
Store2	450	2
Store2	650	3
Store2	850	4

When an output group contains single-occurring columns and a multiple-occurring column, the Normalizer returns duplicate data for the single-occurring columns in each output row. For example, Store1 and Store2 repeat for each instance of Qtr_Sales.

A source row might contain more than one level of multiple-occurring data. You can configure the Normalizer transformation to return separate rows at each level based on how you define the input hierarchy.

Generated Column ID

The Normalizer transformation returns a generated column ID (GCID) output port for each instance of a multiple-occurring field.

The generated column ID port is an index for the instance of the multiple-occurring data. For example, if a field occurs four times in a source record, the Developer tool returns a value of 1, 2, 3, or 4 in the generated column ID port based on which instance of the multiple-occurring data occurs in the row.

Multiple-Occurring Records

You can define multiple-occurring records in the Normalizer transformation source data. Records are groups of fields. Define records in the Normalizer transformation when you need to define groups of source fields that are multiple-occurring.

Multiple-Occurring Records Example

The following Customer row contains customer information with home address information and business address information:

```
CustomerID
FirstName
LastName
Home_Street
Home_City
Home_State
Home_Country
Business_Street
Business_City
Business_State
Business_Country
```

When you configure the Normalizer transformation, you can define an input structure that contains the customer fields and a multiple-occurring address record. The address record occurs twice. When you configure the Normalizer transformation output groups, you can return the Address record to a different target than the CustomerID, FirstName, and LastName fields.

The following example shows an input structure with a multiple-occurring address record:

```
CustomerID
FirstName
LastName
Address (occurs twice)
  Street
  City
  State
  Country
```

Subrecords are records within records. When you define records and subrecords, you define an input hierarchy of fields in the source row. Each record is a node in a hierarchy that you can reference when you define the transformation output.

For example, the source row might contain multiple phone numbers for each address type:

```
CustomerID
FirstName
LastName
Home_Street
Home_City
Home_State
Home_Country
Telephone_No
Cell_Phone_No
Alternate_Phone_No
Business_Street
Business_City
Business_State
Business_Country
Business_Telephone_No
Business_Cell_Phone_No
Business-Alternate_Phone1
```

You define an input hierarchy where Address is the parent of Phone. When you define the Normalizer transformation output, you can return the addresses and the phone numbers to separate targets than the customer information.

Define an input hierarchy similar to the following example:

```
CustomerID
FirstName
LastName
Address (occurs twice)
  Street
  City
  State
  Country
Phone
  Telephone_No (occurs three times)
```

Input Hierarchy Definition

When you create a Normalizer transformation, you define an input hierarchy that describes records and fields in the source. Define the input hierarchy on the **Normalizer** view of the transformation.

The Developer tool creates the transformation input ports based on the fields that you define in the input hierarchy. Define the input group structure before you define the transformation output groups.

When you define an input hierarchy, you must define an input structure that corresponds to the structure of the source data. The source data might contain more than one group of multiple-occurring fields. To define the structure, you can configure a record that occurs at the same level as another record in the source. Or, you can define records that occur within other records.

Input Hierarchy Example

The following source row contains customer fields and an address record that occurs twice:

```
CustomerID
  FirstName
  LastName
  Address
    Street
    City
    State
    Country
  Address1
    Street1
    City1
    State1
    Country1
```

When you define the input structure in the **Normalizer** view, you can add the CustomerID, FirstName, and LastName as fields. Define an Address record and include the Street, City, State, and Country fields in the address. Change the Address Occurs value to 2.

The following image shows the input hierarchy in the **Normalizer** view:

Normalizer						
Name	Level	Occurs	Type	Precision	Scale	
CustomerID	1	1	string	10	0	
FirstName	1	1	string	10	0	
LastName	1	1	string	10	0	
Address	1	2				
Street	2	1	string	10	0	
City	2	1	string	10	0	
State	2	1	string	10	0	
Country	2	1	string	10	0	

The **Occurs** column in the **Normalizer** view identifies the number of instances of a field or record in a source row. Change the value in the **Occurs** column for multiple-occurring fields or records. In this example, the customer fields occur one time, and the Address record occurs twice.

The **Level** column in the **Normalizer** view indicates where a field or record appears in the input hierarchy. The customer fields are at level 1 in the hierarchy. The Address record is also level 1.

Normalizer Transformation Input Ports

The Developer tool creates the Normalizer transformation input ports when you define the input hierarchy in the **Normalizer** view. When you change fields in the input hierarchy, the Developer tool changes the input ports.

View the Normalizer transformation input ports in the **Overview** view. You can reorder the input ports in the **Overview** view. To change the input ports, update the input hierarchy in the **Normalizer** view.

When you define a field as multiple-occurring in the input hierarchy, the Developer tool creates one input port for each instance of the multiple-occurring field. When a record is multiple-occurring, the Developer tool creates an input port for each instance of the fields in the record.

Input Ports Example

The following image shows the input ports that the Developer tool creates for the customer data and the multiple-occurring address data:

Ports					
	Name	Type	Precision	Scale	Location
[-] Input					
1	CustomerID	string	10	0	CustomerID
2	FirstName	string	10	0	FirstName
3	LastName	string	10	0	LastName
4	Street	string	10	0	Address.Street
5	Street1	string	10	0	Address.Street
6	City	string	10	0	Address.City
7	City1	string	10	0	Address.City
8	State	string	10	0	Address.State
9	State1	string	10	0	Address.State
10	Country	string	10	0	Address.Country
11	Country1	string	10	0	Address.Country

Merge Fields

You can merge fields of similar data into a single multiple-occurring field in the **Normalizer** view. You might need to merge fields when you drag ports from another object to create a Normalizer transformation in a mapping.

A source row might contain multiple fields that contain different types of salary data, such as `Base_Salary`, `Bonus_Pay`, and `Sales_Commissions`. You can merge the fields to create one `Salary` field that occurs three times.

The following image shows an employee row with three types of salary selected in the Normalizer view:

Name	Level	Occurs	Type	Preci...	Scale
EmployeeID	1	1	string	10	0
Base_Salary	1	1	decimal	10	0
Bonus_Pay	1	1	decimal	10	0
Sales Commissions	1	1	decimal	10	0

Overview Normalizer Advanced

You can merge the three types of salary data into a `Salary` field that occurs three times.

The following image shows the `Salary` field:

Normalizer						
Name	Level	Occurs	Type	Preci...	Scale	
EmployeeID	1	1	string	10	0	
Salary	1	3	decimal	10	0	

Merging Fields

Merge fields of the same type into one multiple-occurring field on the Normalizer view.

1. Click the **Normalizer** view.
2. Select the fields that you want to merge.
3. Click the **Merge** button.
The **Merge Fields** dialog box appears.
4. Enter a name for the merge field, the type, the precision, the scale, and occurrence of the multiple-occurring field.
5. Click **OK**.

Flatten Fields

You can flatten fields of a complex data type in mappings that run on the Spark engine. You flatten fields in the **Normalizer** view to modify hierarchical data that passes through a complex port.

The output of the flatten action depends on the complex data type. When you flatten an array or struct data type, the Normalizer transformation creates a row for each element in the complex data type. When you flatten a map data type, the Normalizer transformation creates two columns for the map key and map value elements.

The flatten action on a nested data type extracts elements at the first-level. To flatten a nested data type at all levels, use the **Flatten Complex Port** hierarchical conversion wizard in the Developer tool. The **Flatten All** option extracts elements at each level and returns relational data of primitive data type. For more information about hierarchical conversion wizards, see the *Data Engineering Integration User Guide*.

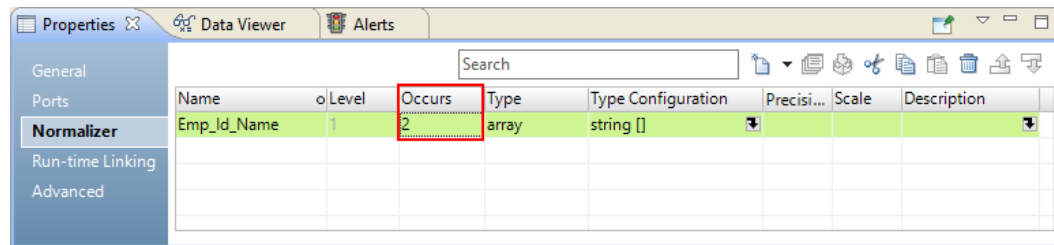
The flatten action changes the value of Occurs column in the Normalizer view to Auto and adds a flatten icon next to the flattened field. The value Auto indicates that the transformation flattens all the elements of the complex data type.

The following image shows a struct that is flattened to a string field with a flatten icon next to it and the Occurs value as Auto:

Name	Level	Occurs	Type	Type Configuration	Precisi...	Scale	Description
StructEmp		Auto	string	N/A	10	0	

You cannot flatten a multi-occurring field. For example, you cannot flatten an array field with Occurs value as 2.

The following image shows a multi-occurring field of an array data type that you cannot flatten:



Flatten Array

The Normalizer transformation flattens a one-dimensional array to a primitive data type and an n-dimensional array to an (n-1)-dimensional array. The number of rows that the transformation creates is the same as the size of the array.

For example, if you flatten an array port with 10 string elements, the output returns 10 string ports. If you flatten a 3-dimensional array, the output returns a 2-dimensional array.

A table contains a string port Name and an array port Phones. You want to flatten the array port. The table contains the following values:

Name	Phones
Adams	[205-128-6478, 722-515-2889, 650-213-4020]
Jane	[650-321-4506]

When you flatten the array port, the output is as follows:

Name	Phones	GCID_Phones
Adams	205-128-6478	1
Adams	722-515-2889	2
Adams	650-213-4020	3
Jane	650-321-4506	1

You can edit the Occurs value of a flattened field to extract a specific number of elements in the array. The value must be a positive integer greater than 1. The value determines the number of elements to extract. For example, you can change the value of Occurs to 2 to extract the first two elements of the array. The output is as follows:

Name	Phones	GCID_Phones
Adams	205-128-6478	1
Adams	722-515-2889	2
Jane	650-321-4506	1

Flatten Struct

The transformation flattens a struct into a field of the data type of the elements in the struct. To flatten a struct data type, all the struct elements must be of the same data type. The transformation creates a row for each element in the struct data type.

For example, you want to flatten the following struct field:

```
customer_address{
  city : string
  state : string
  zip : string
}
```

The table contains the following values:

Name	customer_address
Clara	{ New York NY 10032 }

When you flatten the struct port, the output is as follows:

Name	customer_address	GCID_customer_address
Clara	New York	1
Clara	NY	2
Clara	10032	3

When the struct elements are of different data types and at least the first two elements are of the same data type, you can flatten the struct data for consecutive elements of the same data type. To extract consecutive struct elements of the same data type, edit the Occurs value. The value must be a positive integer greater than 1. For example, a struct emp_address contains the following elements:

```
emp_address{
  city : string
  state : string
  zip : int
  country : string
}
```

You can define the value of Occurs to 2 to extract city and state struct elements. If you define the value as 3 or 4, the mapping validation fails.

Flatten Map

The transformation flattens a map into two fields for the key and value elements in the map. For a map field that you flattened, you cannot change the value of Occurs from Auto to an integer value.

For example, you want to flatten the following map field emp_sal with a string key and an array of integer values:

```
<emp_name -> [base_sal, bonus, commision]>
```

The following image shows the map field that you want to flatten in the Normalizer view:

Name	Level	Occurs	Type	Type Configuration	Precisi...	Scale	Description
emp_id	1	1	string	N/A	10	0	
emp_sal	1	1	map	< string, string [] >			

The table contains the following values:

emp_id	emp_sal
12200	<Greg -> [4000, 1000, 500]>
12201	<Patricia -> [3800, 1500, 1000]>

When you flatten the map port, the output returns a string field for the map key and an array field for the map value as follows:

emp_id	emp_sal_Key	emp_sal_Value	GCID_emp_salary
12200	Greg	[4000, 1000, 500]	1
12201	Patricia	[3800, 1500, 1000]	1

The following image shows the map field that is flattened to a string key field and an array value field in the Normalizer view:

Name	Level	Occurs	Type	Type Configuration	Precisi...	Scale	Description
emp_id	1	1	string	N/A	10	0	
emp_sal	1	Auto	map	N/A			
emp_sal_Key	2	1	string	N/A	10	0	
emp_sal_Value	2	1	array	string []			

The following image shows the Output group in the Ports view:

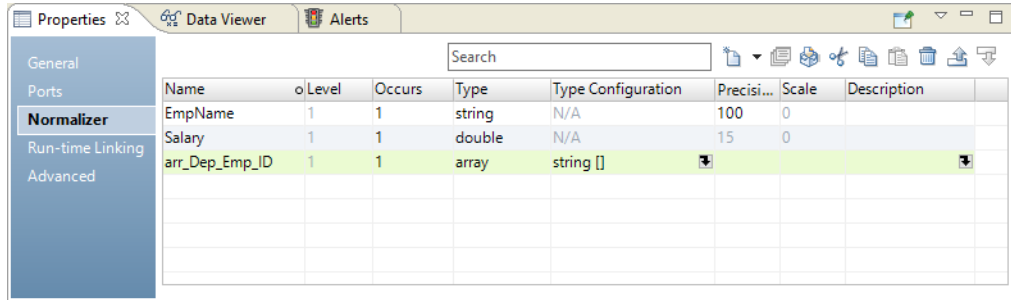
Name	Type	Type Configuration	Precision	Scale	Location	Description
Input						
1 emp_id	string	N/A	10	0	emp_id	
2 emp_sal	map	< string, string [] >			emp_sal	
Output						
1 emp_id	string	N/A	10	0	emp_id	
2 emp_sal_Key	string	N/A	10	0	emp_sal.e...	
3 emp_sal_Value	array	string []			emp_sal.e...	
4 GCID_emp_sal	bigint	N/A	19	0	emp_sal	

Flattening Fields

Flatten fields of a complex data type to modify hierarchical data or to convert to relational data.

1. Click the **Normalizer** view.
2. Select the field of a complex data type.

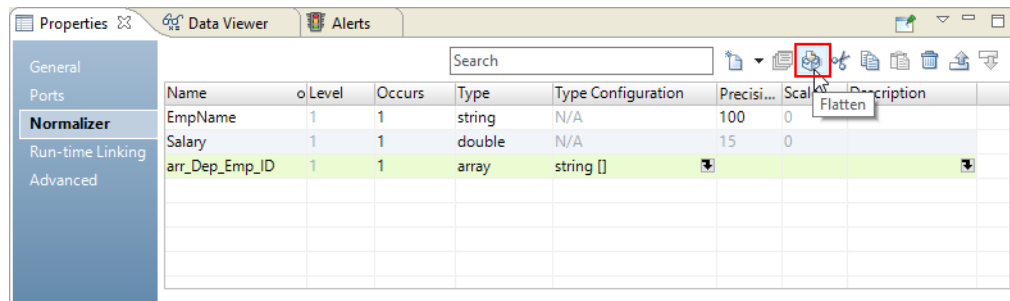
The following image shows a field of type array with string elements:



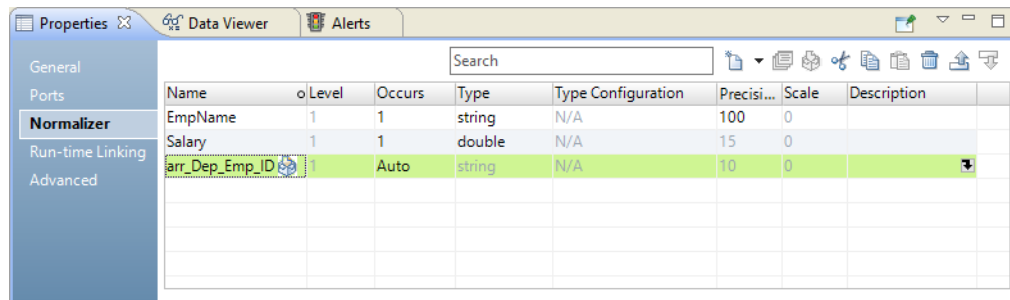
Name	Level	Occurs	Type	Type Configuration	Precisi...	Scale	Description
EmpName	1	1	string	N/A	100	0	
Salary	1	1	double	N/A	15	0	
arr_Dep_Emp_ID	1	1	array	string []			

3. Click the **Flatten** button.

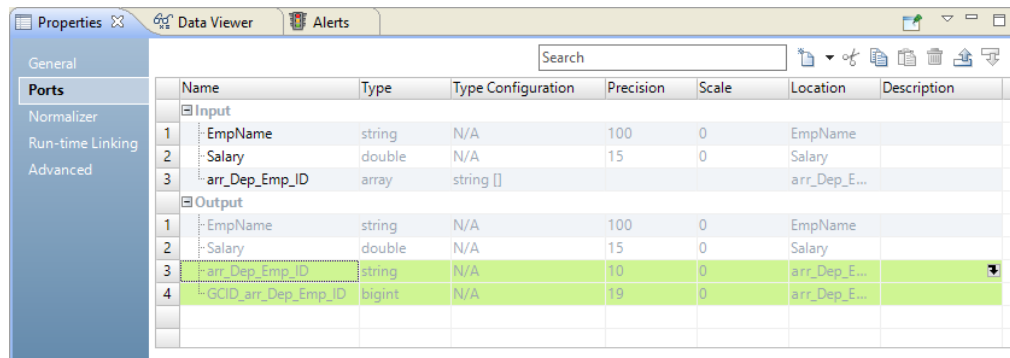
The following image shows the Flatten button:



The flatten action replaces the field of a complex data type with a flattened field and changes the value of **Occurs** to Auto. The data type of the flattened field depends on the complex data type that you flatten. The following image shows the flattened field of type string:



The following image shows the flattened string output port and the GCID output port in the **Ports** view:



Normalizer Transformation Output Groups and Ports

Define output groups and ports in the Overview view of the Normalizer transformation. You can define the output groups after you define the transformation input hierarchy.

The Developer tool generates at least one output group by default. The output group contains all the level 1 fields from the input ports and the first multiple-occurring field. When you define more than one multiple-occurring field in the Normalizer view, the Developer tool creates an output group for each additional multiple-occurring field.

The following source row contains the customer data, the multiple-occurring sales field, and the multiple-occurring phone field:

```
CustomerID
LastName
FirstName
Sales (occurs 4 times)
Phone (occurs 3 times)
```

The Developer tool creates two output groups from the input structure:

```
Output
CustomerID
LastName
FirstName
Sales

Output1
Phone
GCID_Phone
```

The Developer tool creates an Output1 group because the source data contains more than one multiple-occurring field. The Developer tool does not create groups for fields that you define in records. When you define records, you must define the output groups that contain the fields in the records.

The following source row contains customer fields and an address record that occurs twice:

```
CustomerID
  FirstName
  LastName
  Address
    Street
    City
    State
    Country
  Address1
    Street1
    City1
    State1
    Country1
```

The Developer tool creates an output group that contains the following fields.

```
CustomerID
FirstName
LastName
```

The Developer tool creates a default output group for the level 1 customer fields. The default output group does not include the Address record. You must configure how to return the Address data in the output.

Create the output groups based on how you need to structure the output rows. When the source row contains customer data and address data, you can create an output group for the customer fields. You can create another output group for the address fields. Or, you can update the default output group and add the address fields to it. The following examples show different output results based on the output group configuration.

Create an Output Group

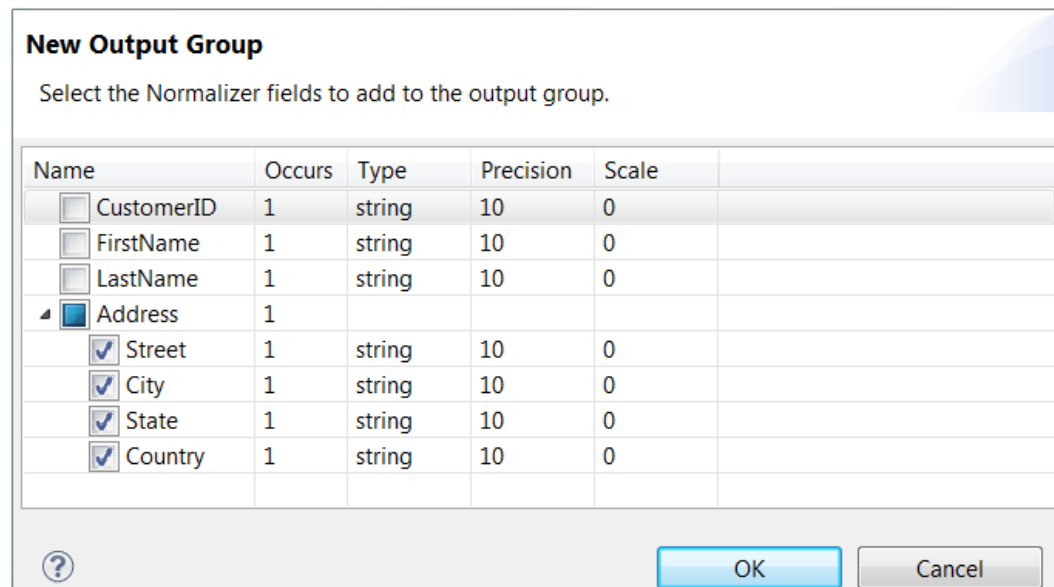
Create an output group in the **Overview** view of the Normalizer transformation.

When you open the **Overview** view of the Normalizer transformation, the Developer tool shows the default groups it creates from the input hierarchy.

When you create a new output group, a dialog box shows a list of the fields and records in the input hierarchy. Select which fields or records to include in the group.

Output Group Example

The following image shows the **New Output Group** dialog box:



When you select the Address record, the Developer tool creates a group of output ports that correspond to the fields in the Address record. The Output1 group contains the Street, City, State, and Country ports. You can change the ports in the output group.

The following image shows the Output group and the Output1 group in the **Overview** view:

	Output				
1	CustomerID	string	10	0	CustomerID
2	FirstName	string	10	0	FirstName
3	LastName	string	10	0	LastName
	Output1				
1	Street	string	10	0	Address.Street
2	State	string	10	0	Address.State
3	Country	string	10	0	Address.Country
4	GCID_Address	bigint	19	0	Address

You might configure the Normalizer transformation to return the rows from the Output group to a Customer table.

The Customer table receives data similar to the following rows:

```
100, Robert, Bold
200, James, Cowan
```

You might return the rows from the Output1 group to an Address table. The Address table receives the Street, City, State, Country, and GCID.

The Address table receives data similar to the following rows:

```
100 Summit Dr, Redwood City, CA, United States,1
41 Industrial Way, San Carlos, CA, United States,2
85 McNulty Way, Los Angeles, CA, United States,1
55 Factory Street, Los Vegas, NV, United States,2
```

The GCID identifies which instance of the customer address is in the output row. In this example, the Normalizer transformation returns two instances of the Address record. Each output row contains a GCID value of 1 or 2.

Update an Output Group

You can update a Normalizer transformation output group. You can add or remove the fields in the group.

By default, the Developer tool creates level 1 output groups when you define an input hierarchy. The Developer tool does not include records in the groups. You can update the default output groups and add records to them.

To update an output group, highlight the group name and click **New > Update Group**. The **Edit Output Group** dialog box shows the fields in the input hierarchy. Choose which fields to include in the group.

Update Output Group Example

In the previous example, the Developer tool created a default output group with the CustomerID, FirstName, and LastName fields.

The following image shows the default output group:

Output					
1	CustomerID	string	10	0	CustomerID
2	FirstName	string	10	0	FirstName
3	LastName	string	10	0	LastName

You can update the default output group and add the Address record to it.

The following image shows the **Edit Output Group** dialog box:

Edit Output Group

Select the Normalizer fields to include in the output group.

Name	Occurs	Type	Precision	Scale
<input checked="" type="checkbox"/> CustomerID	1	string	10	0
<input checked="" type="checkbox"/> FirstName	1	string	10	0
<input checked="" type="checkbox"/> LastName	1	string	10	0
<input checked="" type="checkbox"/> Address	2			
<input checked="" type="checkbox"/> Street	1	string	10	0
<input checked="" type="checkbox"/> State	1	string	10	0
<input checked="" type="checkbox"/> Country	1	string	10	0

In this example, the CustomerID, FirstName, and LastName are level 1 nodes. The Address record is also a level 1 node. The Normalizer transformation can return Address in the same row as the customer data. Since Address is multiple-occurring, the Developer tool adds the GCID_Address index to the output group.

The following image shows the ports in the Output group:

Output					
1	CustomerID	string	10	0	CustomerID
2	FirstName	string	10	0	FirstName
3	LastName	string	10	0	LastName
4	Street	string	10	0	Address.Street
5	State	string	10	0	Address.State
6	Country	string	10	0	Address.Country
7	GCID_Address	bigint	19	0	Address

When the customer fields and the multiple-occurring address fields are in the output group, the Normalizer transformation returns the same customer fields for each instance of the address data.

The following example shows the rows that the Normalizer transformation generates from the output group:

```
100, Robert, Bold, 100 Summit Dr, Redwood City, CA, United States,1
100, Robert, Bold, 41 Industrial Way, San Carlos, CA, United States,2
200, James, Cowan, 85 McNulty Way, Los Angeles, CA, United States,1
200, James, Cowan, 55 Factory Street, Los Vegas, NV, United States,2
```

The GCID port contains the Address instance number. The GCID value is 1 or 2.

Key Generation for Output Groups

You can configure a Sequence Generator transformation to generate keys that link each output row that the Normalizer transformation returns from the same source row.

You can add a Sequence Generator transformation before the Normalizer transformation in a mapping. The Sequence Generator transformation adds a sequence number to each source row. When the Normalizer transformation returns multiple output groups or rows from the same source row, each output row receives the same sequence number. You can use the number as a key in a primary key-foreign key relationship between target tables.

For example, the Normalizer transformation returns customer information in an output group and it returns order information in another output group. You can use the sequence number to link the customer information in a table to the order information in another table.

Normalizer Transformation Advanced Properties

Configure properties for the Normalizer transformation on the **Advanced** tab.

Configure the following properties on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Generate First Level Output Groups

The Developer tool generates the Normalizer transformation first-level output group by default. The Developer tool generates additional output groups when you define more than one multiple-occurring field in the Normalizer view.

The Developer tool creates an output group that contains all level 1 single-occurring fields and the first level 1 multiple-occurring field. The Developer tool creates an additional output group when the transformation contains more than one multiple-occurring field.

The Developer tool does not create an output group for level 1 records. When you define a level 1 record, you must configure where you to put them in the output.

To stop the Developer tool from creating output groups for multiple-occurring fields, disable the **Automatically Generate First-Level Output Groups** advanced property.

Creating a Normalizer Transformation

You can create a reusable or non-reusable Normalizer transformation. Reusable transformations can exist in multiple mappings. Non-reusable transformations exist within a single mapping.

1. To create a transformation, use one of the following methods:

Option	Description
Reusable	Select a project or folder in the Object Explorer view. Click File > New > Transformation . Select the Normalizer transformation and click Next .
Non-reusable	In a mapping or mapplet, drag a Normalizer transformation from the Transformation palette to the editor. You can drag ports from source data objects or from transformations in the mapping to define the transformation.

The **New Normalizer Transformation** wizard appears.

2. Enter a name for the transformation.
3. Click **Next**.
The **Normalizer Definition** page appears.
4. To add a record, click the **New** button and then select **Record**.
5. To add a field, click the **New** button and then select **Field**.
To add a field to a record, you must select the record before you add the field.
6. Optionally, double-click the value in the **Occurs** column to change the occurrence of a field or record.
7. Click **Next**.
The **Normalizer Ports** page appears.
8. To add an output group, click the **New** button and then select **New Output Group**.
9. To edit an output group, select the output group that you want to edit. Click the **New** button and then select **Edit Output Group**.
10. Click **Finish**.

The transformation appears in the editor.

Creating a Normalizer Transformation from an Upstream Source

You can create an empty Normalizer transformation and drag the ports from a source data object or from a transformation into the Normalizer transformation to create input and ports.

1. Create a mapping that includes the source or transformation to pass the source data to the Normalizer transformation.
2. To create the Normalizer transformation, select the Normalizer transformation from the Transformation palette and drag the transformation to the editor.

The Normalizer dialog box appears.

3. Click **Finish** to create an empty transformation.
4. Select ports from a source or transformation in the mapping and drag them to the Normalizer transformation.
The input and output ports appear in the Normalizer transformation. The Developer tool creates one input group and one output group.
5. Open the **Normalizer** view to update the default group and organize fields into records as required.
6. To merge multiple fields into a single multiple-occurring field, select the fields in the **Normalizer** view and click the **Merge** option.

Choose a name for the multiple-occurring field.

Normalizer Mapping Example

A retail organization receives sales totals for the stores in the organization. The organization receives a row of data that contains store information and four sales amounts. Each sales amount represents the total sales for one quarter of the year.

The following example shows how to define a Normalizer transformation to return the sales data to a Store target and a Sales target. The Store target receives one row for each store. The Sales target receives four rows from each store. Each row contains one quarter of sales data.

A Sequence Generator transformation generates a unique ID for each store. The Normalizer transformation returns the StoreID with each output row.

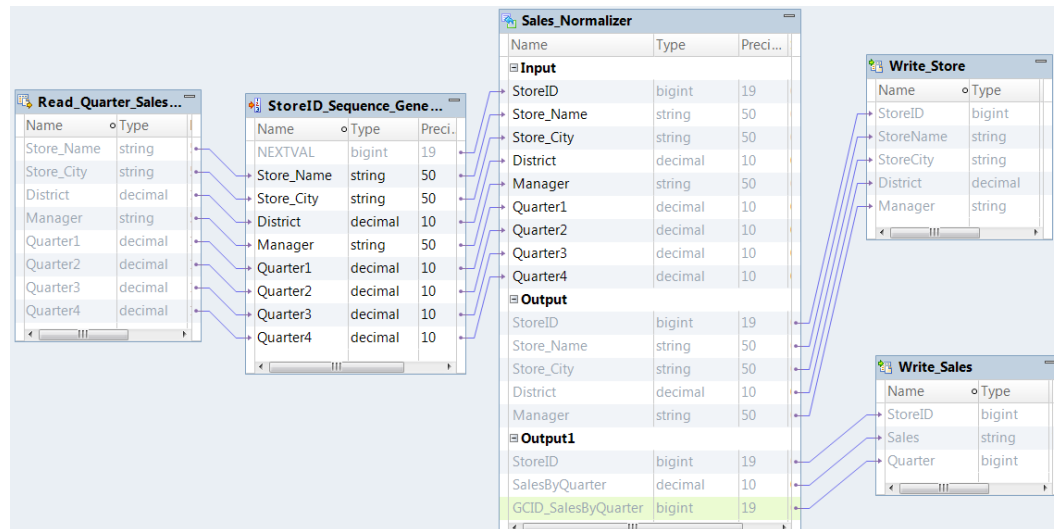
Create a mapping with a Read transformation, a Sequence Generator transformation, the Normalizer transformation, and two Write transformations.

Normalizer Example Mapping

Create a mapping that contains the Normalizer transformation to normalize multiple-occurring quarterly sales data from a flat file source.

The Normalizer transformation generates a separate output row for each quarter of sales and writes the normalized sales amounts to a the Sales target. The Normalizer transformation writes the Store information to a Store target.

The following figure shows the Normalizer transformation mapping:



The mapping contains the following objects:

Read_STORE

A data source that contains multiple-occurring fields.

StoreID Sequence Generator transformation

A Sequence Generator transformation that generates a storeID key to link the Store table to the Sales table.

Sales_Normalizer

A Normalizer transformation that normalizes the multiple-occurring sales data.

Write_Store

A target that receives the store information from the Normalizer transformation.

Write_Sales

A target that receives the sales numbers from the Normalizer transformation.

Normalizer Example Definition

The source is a flat file that contains store information and quarterly sales data. Define the structure of the source data in the **Normalizer** view.

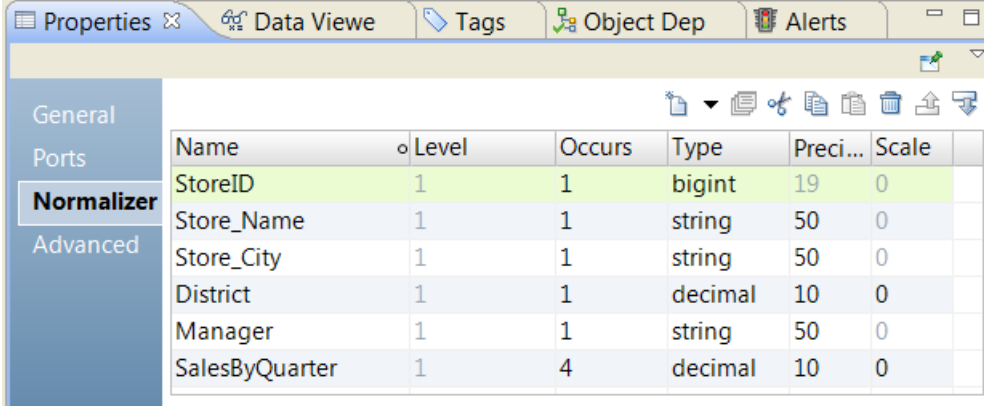
The STORE flat file contains the following source data:

StoreID	Store_Name	Store_City	District	Manager	Quarter1	Quarter2	Quarter3	Quarter4
1	BigStore	New York	East	Robert	100	300	500	700
2	SmallStore	Phoenix	West	Radhika	250	450	650	850

Add the flat file to a mapping as a Read transformation, and then create an empty Normalizer transformation. Drag the ports from the Read_STORE data object to the Normalizer transformation to create the Normalizer definition.

The **Normalizer** view contains one instance of the Store_Name, Store_City, District, and Manager fields. The **Normalizer** view contains four instances of a field called QUARTER. Merge the QUARTER fields to create one SalesByQuarter field that occurs four times.

The following figure shows the Normalizer definition with merged Quarter fields:



Name	Level	Occurs	Type	Preci...	Scale
StoreID	1	1	bigint	19	0
Store_Name	1	1	string	50	0
Store_City	1	1	string	50	0
District	1	1	decimal	10	0
Manager	1	1	string	50	0
SalesByQuarter	1	4	decimal	10	0

Normalizer Example Input and Output Groups

After you modify the input hierarchy, the Normalizer transformation has one input group and one default output group. You need to reorganize the output ports into two groups. You need one group that contains the Store information and one group that contains the Sales information.

The input group contains a port for each field in the source. The output group contains ports for the store fields and port for the multiple-occurring SalesByQuarter field. The output group also contains a generated column ID, `GCID_SalesByQuarter`, that corresponds to the multiple-occurring SalesByQuarter field.

To return the quarterly sales to a different target, create a new group in the **Overview** view. In the Output1 group, add the following fields:

```
StoreID
SalesByQuarter
GCID_SalesByQuarter
```

Update the default output group. Remove the following fields:

```
SalesByQuarter
GCID_SalesByQuarter
```

The following image shows the input group and output groups in the **Overview** view:

The screenshot shows the 'Ports' section of the Data Viewer tool. It contains three groups of ports: 'Input', 'Output', and 'Output1'. Each group lists ports with their names, types, and precision/scale values.

Name	Type	Precision	Scale	Location
Input				
1 StoreID	bigint	19	0	StoreID
2 Store_Name	string	50	0	Store_N...
3 Store_City	string	50	0	Store_City
4 District	decimal	10	0	District
5 Manager	string	50	0	Manager
6 Quarter1	decimal	10	0	SalesBy...
7 Quarter2	decimal	10	0	SalesBy...
8 Quarter3	decimal	10	0	SalesBy...
9 Quarter4	decimal	10	0	SalesBy...
Output				
1 StoreID	bigint	19	0	StoreID
2 Store_Name	string	50	0	Store_N...
3 Store_City	string	50	0	Store_City
4 District	decimal	10	0	District
5 Manager	string	50	0	Manager
Output1				
1 StoreID	bigint	19	0	StoreID
2 SalesByQuarter	decimal	10	0	SalesBy...
3 GCID_SalesByQuarter	bigint	19	0	SalesBy...

The StoreID is the generated key that links the Store information with the Sales information. Verify that both output groups return the StoreID.

Normalizer Example Mapping Output

Add the Write transformation to the mapping and connect the Normalizer transformation output ports to the data objects.

When you run the mapping, the Normalizer transformation writes the following rows to the Store target:

StoreID	Store_Name	Store_City	District	Manager
1	BigStore	New York	East	Robert
2	SmallStore	Phoenix	West	Radhika

The Normalizer transformation writes the following rows to the Sales target:

StoreID	SalesByQuarter	GCID_SalesByQuarter
1	100	1
1	300	2
1	500	3
1	700	4
2	250	1
2	450	2
2	650	3
2	850	4

Normalizer Transformation in a Non-native Environment

The Normalizer transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 33

Merge Transformation

This chapter includes the following topics:

- [Merge Transformation Overview, 519](#)
- [Configuring a Merge Strategy, 519](#)
- [Merge Transformation Advanced Properties, 520](#)
- [Merge Transformation in a Non-native Environment, 520](#)

Merge Transformation Overview

The Merge transformation is a passive transformation that reads the data values from multiple input columns and creates a single output column.

Use the Merge transformation to create data in a preferred format. For example, you can combine Customer_Firstname and Customer_Surname fields to create a field called Customer_FullName.

Within a Merge transformation, you can create multiple merge strategies. The Merge transformation provides a wizard that you use to create strategies.

Configuring a Merge Strategy

To configure a merge strategy, edit the settings in the **Strategies** view of a Merge transformation.

1. Select the **Strategies** view.
2. Click **New**.
The **New Strategy** wizard opens.
3. Click the **Inputs** field to select input ports for the strategy.
4. To define the merge character to place between merged items, click **Choose**. If you do not select a merge character, the Merge transformation uses a space character by default.
5. Optionally, select **Include empty strings in merged output** to include empty input strings in the output.
6. Click **Finish**.

Merge Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Merge transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Merge Transformation in a Non-native Environment

The Merge transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 34

Parser Transformation

This chapter includes the following topics:

- [Parser Transformation Overview, 521](#)
- [Parser Transformation Modes, 522](#)
- [When to Use a Parser Transformation, 522](#)
- [Reference Data Use in the Parser Transformation, 523](#)
- [Token Parsing Operations, 525](#)
- [Token Parsing Ports, 526](#)
- [Token Parsing Properties, 526](#)
- [Pattern-Based Parsing Mode, 529](#)
- [Configuring a Token Parsing Strategy, 530](#)
- [Configuring a Pattern Parsing Strategy, 530](#)
- [Parser Transformation Advanced Properties, 531](#)
- [Parser Transformation in a Non-native Environment, 531](#)

Parser Transformation Overview

The Parser transformation is a passive transformation that parses input data values into new ports. The transformation writes the values to new ports according to the types of information the values contain and the position of the values in the input string.

You use a Parser transformation when you want to change the structure of a data set. The Parser transformation can add columns to a data set and write data values to new columns. Use a Parser transformation when a data column contains multiple values in a single column and you want to write the data values to individual columns based on the type of information they contain.

The Parser transformation parses data values into the output ports that you define. If the transformation can identify an input data value but a defined output port is not available, the transformation writes it to an overflow port. If the transformation cannot identify an input data value, it writes it to an unparsed data port.

Parser Transformation Modes

When you create a Parser transformation, select either token parsing mode or pattern-based parsing mode.

You select one of the following modes:

- Token parsing mode. Use this mode to parse input values that match values in reference data objects such as token sets, regular expressions, probabilistic models, and reference tables. You can use multiple token parsing strategies in a transformation.
- Pattern-based parsing mode. Use this mode to parse input values that match values in pattern sets.

When to Use a Parser Transformation

Use the Parser transformation when the data fields in a column contain more than one type of information and you want to move the field values to new columns. The Parser transformation lets you create new column for each type of information in a data set.

The following examples describe some types of structural change you can perform with a Parser transformation.

Create new columns for contact data

You can create a data structure that parses name data from a single column into multiple columns. For example, you can create columns for salutations, first names, middle names, and surnames.

You configure the transformation with a probabilistic model that represents the structures of the person names on the input port. You use a sample of the input port data to define the model.

You create a token parsing strategy that applies the probabilistic model to the input port and writes the name values to new columns. The transformation writes the name values to the new columns based on the position of each value in the input string and the type of name that the value represents.

Note: You can also use a pattern-based parsing strategy to parse contact data. When you configure a pattern-based parsing strategy, you define the patterns that represents the structures of the names on the input port.

Create address columns

You can create a data structure that parses a single column of address data into multiple columns that describe a deliverable address.

Configure the transformation with reference tables that contain recognizable address elements, such as ZIP Codes, state names, and city names. Create a token parsing strategy that writes each address element to a new port.

You cannot use a reference table to parse street address data from an input string, because street name and number data is too general to be captured in a reference table. However, you can use the Overflow port to capture this data. When you have parsed all city, state, and ZIP data from an address, the remaining data contains street information.

For example, use a token parsing strategy to split the following address into address elements:

```
123 MAIN ST NW STE 12 ANYTOWN NY 12345
```

The parsing strategy can write the address elements to the following columns:

Column Name	Data
Overflow	123 MAIN ST NW STE 12
City	ANYTOWN
State	NY
ZIP	12345

Create product data columns

You can create a data structure that parses a single column of product data into multiple columns that describe the product inventory details.

Configure the transformation with token sets that contain inventory elements, such as dimension, color, and weight. Create a token parsing strategy that writes each inventory element to a new port

For example, use a token parsing strategy to split the following paint description into separate inventory elements:

```
500ML Red Matt Exterior
```

The parsing strategy can write the values to the following columns:

Column Name	Data
Size	500ML
Color	Red
Style	Matt
Exterior	Y

Reference Data Use in the Parser Transformation

Informatica Developer installs with multiple reference data objects that you can use with the Parser transformation. You can also create reference data objects in the Developer tool.

When you add a reference data object to a Parser transformation, the transformation writes the strings that match a value in the object to new columns that you specify.

The following table describes the types of reference data you can use:

Reference Data Type	Description
Pattern sets	Identifies data values based on the relative position of each value in the string.
Probabilistic models	Adds fuzzy match capabilities to token parsing operations. The transformation can use a probabilistic model to infer the type of information in a string. To enable the fuzzy match capabilities, you compile the probabilistic model in the Developer tool.
Reference tables	Finds strings that match the entries in a database table.
Regular expressions	Identifies strings that match conditions that you define. You can use a regular expression to find a string within a larger string.
Token sets	Identifies strings based on the types of information they contain. Informatica installs with token sets different types of token definitions, such as word, telephone number, post code, and product code definitions.

Pattern Sets

A pattern set contains expressions that identify data patterns in the output of a token labeling operation. You can use pattern sets to analyze the Tokenized Data output port and write matching strings to one or more output ports. Use pattern sets in Parser transformations that use pattern parsing mode.

For example, you can configure a Parser transformation to use pattern sets that identify names and initials. This transformation uses the pattern sets to analyze the output of a Labler transformation in token labeling mode. You can configure the Parser transformation to write names and initials in the output to separate ports.

Probabilistic Models

A probabilistic model identifies tokens by the types of information they contain and by the positions that they occupy an input string.

A probabilistic model contains reference data values and label values. The reference data values represent the data on an input port that you connect to the transformation. The label values describe the types of information that the reference data values contain. You assign a label to each reference data value in the model.

To link the reference data values to the labels in a probabilistic model, you compile the model. The compilation process generates a series of logical associations between the data values and the labels. When you run a mapping that reads the model, the Data Integration Service applies the model logic to the transformation input data. The Data Integration Service returns the label that most accurately describes the input data values.

You create a probabilistic model in the Developer tool. The Model repository stores the probabilistic model object. The Developer tool writes the data values, the labels, and the compilation data to a file in the Informatica directory structure.

Note: If you add a probabilistic model to a token parsing operation and you then edit the label configuration in the probabilistic model, you invalidate the operation. When you update the label configuration in a probabilistic model, recreate any parsing operation that uses the model.

Reference Tables

A reference table is a database table that contains at least two columns. One column contains the standard or required version of a data value, and other columns contain alternative versions of the value. When you add a reference table to a transformation, the transformation searches the input port data for values that also appear in the table. You can create tables with any data that is useful to the data project you work on.

Regular Expressions

In the context of parsing operations, a regular expression is an expression that you can use to identify one or more strings in input data. The Parser transformation writes identified strings to one or more output ports. You can use regular expressions in Parser transformations that use token parsing mode.

Parser transformations use regular expressions to match patterns in input data and parse all matching strings to one or more outputs. For example, you can use a regular expression to identify all email addresses in input data and parse each email address component to a different output.

Token Sets

A token set contains expressions that identify specific tokens. You can use token sets in Parser transformations that use token parsing mode.

Use token sets to identify specific tokens as part of parsing operations. For example, you can use a token set to parse all email addresses that use that use an "AccountName@DomainName" format.

Token Parsing Operations

In token parsing mode, the Parser transformation parses strings that match data in token sets, regular expressions, probabilistic models, or reference table entries.

To perform token parsing, add strategies on the **Strategies** view of the transformation. You can add one or more operations to each strategy. The transformation provides a wizard that you use to create strategies.

You can add the following types of operations to a token parsing strategy:

Parse using Token Set

Use predefined or user-defined token sets to parse input data. Token set operations can use custom regular expressions that write data to one or more outputs.

You can also use probabilistic models to identify and parse input data values.

Parse using Reference Table

Use reference tables to parse input data.

The transformation performs the operations in the order in which they appear in the strategy.

Token Parsing Ports

Configure the token parsing ports with settings appropriate for your data.

A Parser transformation in token parsing mode has the following port types:

Input

Contains data that you pass to the Parser transformation. The transformation merges all input ports into a combined data string using the **Input Join Character** specified on the **Strategies** tab. If you do not specify an input join character, the transformation uses a space character by default.

Parsed Output Ports

User-defined output port(s) that contains successfully parsed strings. In cases where multiple parsing strategies use the same output, the transformation merges the output into a combined data string using the **Output Join Character** specified on the **Strategies** tab. If you do not specify an output join character, the transformation uses a space character by default.

Overflow

Contains successfully parsed strings that do not fit into the number of outputs defined in the transformation. For example, if the transformation only has two "WORD" outputs, the string "John James Smith" results in an overflow output of "Smith." The Parser transformation creates an overflow port for each strategy that you add.

When you select the Detailed Overflow option, the transformation creates an overflow port for each label in the model.

Unparsed

Contains strings that the transformation cannot parse successfully. The Parser transformation creates an unparsed port for each strategy that you add.

Output Ports in Probabilistic Matching

When you configure a parsing strategy to use probabilistic matching techniques, the Parser transformation adds a port to store the match scores for each output port.

The following table describes the types of port:

Port Type	Port Created in Probabilistic Matching
Parsed output port	[label name] output [label name] score output
Overflow data port	[overflow data] output [[overflow data] score output
Unparsed data port	[unparsed data] output [unparsed data] score output

Token Parsing Properties

Configure properties for token parsing operations on the **Strategies** view in the Parser transformation.

General Properties

General properties apply to all token parsing operations you define in the strategy. You use the general properties to name the strategy, specify input and output ports, and specify whether the strategy enables probabilistic matching techniques.

The following table describes the general properties.

Property	Description
Name	Provides a name for the strategy.
Inputs	Identifies the input ports that the strategy operations can read.
Outputs	Identifies the output ports that the strategy operations can write to.
Description	Describes the strategy. The property is optional.
Use probabilistic matching techniques	Specifies that the strategy can use a probabilistic model to identify tokens.
Input Join Character	Specifies the character used to join input data ports. The transformation merges all input ports into a combined data string and parses the complete string.
Output Join Character	Specifies the character used to join output data values when multiple parsing operations use the same output.
Reverse Enabled	Configures the strategy to parse data from right to left. This property is disabled for probabilistic matching.
Overflow Reverse Enabled	Configures the strategy to parse the overflow data from right to left. This property is disabled for probabilistic matching.
Detailed Overflow Enabled	Creates a unique overflow field for each parsing operation.
Delimiters	Specifies the delimiter that separates the input data into separate tokens. Default is space.

Probabilistic Model Properties

You can select a probabilistic model in place of a token set when you configure a token parsing strategy. Select the **Parse using Token Set** operation and select the option to use probabilistic matching techniques.

The following table describes the probabilistic model properties:

Property	Description
Name	Provides a name for the operation.
Filter Text	Uses characters or wildcards you enter to filter the list of token sets, probabilistic models, or regular expressions.
Probabilistic Model	Identifies the probabilistic model you select.

Reference Table Properties

Reference table properties apply when you configure a labeling operation to use a reference table.

The following table describes the reference table properties:

Property	Description
Name	Provides a name for the operation.
Reference Table	Specifies the reference table that the operation uses to parse input values.
Case Sensitive	Determines whether input strings must match the case of reference table entries.
Replace Matches with Valid Values	Replaces parsed data with the data from the Valid column in the reference table.
Outputs	Specifies the output ports for the parsed data.

Token Set Properties

Token set properties apply when you configure a parsing operation to use token sets.

Select the **Parse using Token Set** operation to parse input with token sets. Clear the option to use probabilistic matching techniques.

The following tables describe the token set properties:

Property	Description
Name	Provides a name for the operation.
Token Sets (Single Output Only)	Specifies the token set that the operation uses to parse data. The operation writes the data to a single port.
Regular Expression (Single or Multiple Outputs)	Specifies the regular expression that the operation uses to parse data. The operation writes the data to multiple ports if it finds multiple strings in the input field.
Outputs	Identifies the output ports that the operations writes to.

You can add, edit, import, or remove a token set or a regular expression. You can also filter the list of the token sets.

The following table describes the properties that you use to perform the tasks:

Property	Description
Filter text	Filters the list of token sets or regular expressions. Use text characters and wildcard characters as a filter.
Add	Use to define a custom token set or regular expression.
Edit	Edits the contents of a custom token set.

Property	Description
Import	Imports a nonreusable copy of a token set or regular expression from a folder in the Model repository. If you update the source object for the token set or regular expression, the Data Integration Service does not update the nonreusable copy.
Remove	Deletes a custom token set or regular expression.

Pattern-Based Parsing Mode

In pattern-based parsing mode, the Parser transformation parses patterns made of multiple strings.

You can use the following methods to define patterns in pattern-based parsing mode:

- Parse input data using patterns defined in reference tables. You can create a pattern reference table from the profiled output of a Labeler transformation that uses the token labeling mode.
- Parse input data using patterns that you define.
- Parse input data using patterns that you import from a reusable pattern set in the Model repository. Changes to the reusable pattern set do not update the data you add in the Parser transformation.

You can use the "+" and "*" wildcards to define a pattern. Use "*" characters to match any string, and "+" characters to match one or more instances of the preceding string. For example, use "WORD+" to find multiple consecutive instances of a word token, and use "WORD*" to find a word token followed by one or more tokens of any type.

You can use multiple instances of these methods within the Parser transformation. The transformation uses the instances in the order in which they are listed on the **Configuration** view.

Note: In pattern-based parsing mode, the Parser transformation requires the output of a Labeler transformation that uses token labeling mode. Create and configure the Labeler transformation before creating a Parser transformation that uses pattern-based parsing mode.

Pattern-Based Parsing Ports

Configure the pattern-based parsing ports with settings appropriate for your data.

A Parser transformation that uses the pattern-based parsing mode has the following port types:

Label_Data

Connect this port to the `Labeled_Output` port of a Labeler transformation that uses the token labeling mode.

Tokenized_Data

Connect this port to the `Tokenized_Data` output port of a Labeler transformation that uses the token labeling mode.

Parse_Status

If a match is found for the input pattern, this port outputs the value `Matched`. If no match is found, it outputs `Unmatched`.

Overflow

Successfully parsed strings that do not fit into the number of outputs defined in the transformation. For example, if only two "WORD" outputs are defined, the string "John James Smith" results in an overflow output of "Smith" by default.

Parsed

Successfully parsed strings in user-defined ports.

Configuring a Token Parsing Strategy

To configure a token parsing strategy, open a Parser transformation in token parsing mode and select the **Strategies** view.

1. Select the **Strategies** view.
2. Click **New**.
The **New Strategy** wizard opens.
3. Click the **Inputs** field to select ports for the strategy.
4. Configure the strategy properties and click **Next**.
5. Choose an operation and click **Next**.
6. Configure the operation properties and select output ports for successfully parsed data.
7. Optionally, click **Next** to add more operations to the strategy.
8. After you add all operations to the strategy, click **Finish**.
9. Optionally, add more strategies to the transformation.
10. Optionally, change the order in which the transformation processes strategies and operations. Select a strategy or operation and click **Move Up** or **Move Down**.

Configuring a Pattern Parsing Strategy

To configure a pattern parsing strategy, open a Parser transformation in pattern parsing mode and select the **Patterns** view.

Before you configure the transformation to parse patterns, verify that the **Patterns** view shows the output port names you expect. The Parser transformation parses tokens to output ports you select. Create additional output ports if necessary.

1. Select the **Patterns** view.
2. Add one or more patterns to the strategy. You can add patterns in the following ways:
 - Enter data values to create a pattern. Click **New** and select **New Pattern**.
If you select **New Pattern**, click **Enter Patterns Here** and enter one or more token types. The tokens you enter must match the token structure of an input data field. Add patterns need to describe the token structures on the input port.
 - Import data values from a reference table. Click **New** and select **New Reference Table**.

If you select **New Reference Table**, browse the Model repository and select a reference table that contains a list of token structures. The reference table must contain two columns. The second column in the reference table must contain numeric values.

- Import data values from a pattern set. Click **Import** and select a reusable pattern set in the Model repository.

If you select **Import**, browse the content sets in the Model repository and select a reusable pattern set.

Note: You can use the **Filter text** field to filter the lists of reference tables and pattern sets.

You can mix pattern sets and reference tables in the Patterns column.

3. Assign each token in the Patterns column to an output port.
 - To assign a token to an output port, double-click in the port column and select the token name from the menu.
 - To parse multiple tokens to a single output, double-click in the port column and select **Custom**. Assign tokens to the port and select the delimiter to use.

Assign the tokens in each row of the pattern to one or more output ports.

4. Save the transformation.

Parser Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Parser transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Parser Transformation in a Non-native Environment

The Parser transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 35

Python Transformation

The Python transformation provides an interface to define transformation functionality using Python code.

Python is a language that uses simple syntax, dynamic typing, and dynamic binding, making Python an ideal choice to increase productivity or to participate in rapid application development. When you use your Python code in a data engineering mapping, the Python code is embedded into the generated Scala code that the Spark or Databricks Spark engine runs to process large, diverse, and fast-changing data sets.

You can also use the Python transformation for machine learning. In the transformation, you can specify a resource file that contains a pre-trained model and load the pre-trained model in the Python code. For example, you can load a pre-trained model to classify input data or to create predictions.

Before you can use the Python transformation, configure the corresponding Spark advanced properties in the Hadoop connection or Databricks connection properties. Then, ensure that the worker nodes on the cluster contain an installation of Python.

For more information about installing Python, see the *Data Engineering Integration Guide*.

You can only run the Python transformation on the Spark or Databricks Spark engine. You cannot run the Python transformation in the native environment.

For more information about the Python transformation, see the *Data Engineering Integration User Guide*.

CHAPTER 36

Rank Transformation

This chapter includes the following topics:

- [Rank Transformation Overview, 533](#)
- [Rank Transformations in Dynamic Mappings, 534](#)
- [Rank Transformation Ports, 534](#)
- [Rank Port, 536](#)
- [Define Group By Ports, 536](#)
- [Rank Caches, 537](#)
- [Rank Transformation Advanced Properties, 538](#)
- [Rank Transformation in a Non-native Environment, 539](#)

Rank Transformation Overview

The Rank transformation is an active transformation that limits records to a top or bottom range. Use a Rank transformation to return the largest or smallest numeric value in a port or group. Or use a Rank transformation to return the strings at the top or the bottom of a mapping sort order.

During a mapping run, the Data Integration Service caches input data until it can perform the rank calculations.

The Rank transformation differs from the transformation functions MAX and MIN. The Rank transformation returns a group of top or bottom values, not just one value. For example, use a Rank transformation to select the top 10 salespersons in a given territory. Or, to generate a financial report, you might 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. The rows that fall within that rank, based on some measure you set when you configure the transformation, pass through the 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. The top 10 rows pass from the Rank transformation to another transformation.

You can connect ports from one transformation to the Rank transformation. You can also create local variables and write non-aggregate expressions.

Ranking String Values

You can configure the Rank transformation to return the top or bottom values of a string port. The Data Integration Service sorts strings based on the sort order selected for the deployed mapping.

When you configure the application that contains the mapping, you select the sort order that the Data Integration Service uses to run the mapping. You can select binary or a specific language such as French or German. If you select binary, the Data Integration Service calculates the binary value of each string and sorts the strings using the binary values. If you select a language, the Data Integration Service sorts the strings alphabetically using the sort order for the language.

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 that you want to rank.
- Define groups for ranks, such as the 10 least expensive products for each manufacturer.

Rank Transformations in Dynamic Mappings

You can use a Rank transformation in a dynamic mapping. You can configure dynamic ports in the transformation and reference the generated ports.

If you reference a generated port in the Rank transformation and the generated port does not exist at run time, the mapping fails.

If you specify a dynamic port as a Rank port, the dynamic port can have no more than one generated port.

If you specify a dynamic port as the Group By port, the Data Integration service considers all the generated ports as Group By ports. The mapping is not valid if you specify a generated port as a Group By port and you specify the parent dynamic port as a Rank port or as a Group By port.

You can parameterize the Rank port and the Group By ports. Use a port type parameter for the Rank port. Use a port list type parameter for the Group By ports.

Rank Transformation Ports

The Rank transformation includes input, input/output, or output ports that are connected to another transformation in the mapping. The transformation also includes pass-through, and variable ports.

A Rank transformation has the following port types:

Input

Receives data from upstream transformations. You can designate input ports as input/output ports. The transformation must have at least one input port.

Dynamic Port

A port that can receive multiple columns to create a dynamic number of generated ports. A generated port is a port within a dynamic port that represents a single column. You can create input, output, and variable dynamic ports.

Output

Passes data to downstream transformations. You can designate output ports as input/output ports. The transformation must have at least one output port.

Pass-Through

Passes data unchanged.

Variable

Used for local variables. You can use a variable port to store values or calculations to use in an expression. Variable ports cannot be input or output ports. They pass data within the transformation.

Rank Index

The Developer tool creates a RANKINDEX port for each Rank transformation. The Data Integration Service uses the Rank Index port to store the ranking position for each row in a group.

For example, you might create a Rank transformation to identify the 50 highest paid employees in the company. You 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

The RANKINDEX is an output port only. You can pass the rank index to another transformation in the mapping or directly to a target.

Rank Port

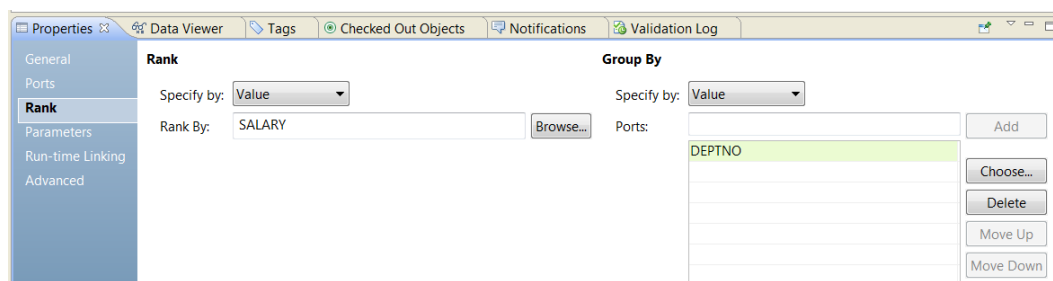
The rank port determines the column to rank values by.

You must designate one input/output or output ports as the rank port. For example, you create a Rank transformation to rank the top employees in each department based on salary. The Salary port contains the salary for each employee. You designate the Salary input/output port as the Rank port.

Select the rank port on the **Rank** tab of the **Properties** view. You can use a parameter for the rank port. To use a parameter, select **Specify By Parameter**. Browse for a port parameter, or create a port parameter. The parameter default value is the name of a port or generated port.

You must link the rank port to another transformation.

The following image shows the **Rank** tab:



Note: The rank port does not support binary data types.

Define Group By Ports

You can configure the Rank transformation to create groups for ranked rows.

For example, if you want to select the 10 most expensive items by manufacturer, you would first define a group for each manufacturer. In the **Group By** panel of the **Rank** tab, you can set one of the input, input/output, or output ports as a group by port.

For each unique value in the group port, the transformation creates a group of rows falling within the rank definition (top or bottom, and a particular number in each rank).

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, if you create a Rank transformation that ranks the top five salespersons grouped by quarter, the rank index numbers the salespeople from 1 to 5 for each quarter:

RANKINDEX	SALES_PERSON	SALES	QUARTER
1	Sam	10,000	1
2	Mary	9,000	1
3	Alice	8,000	1
4	Ron	7,000	1

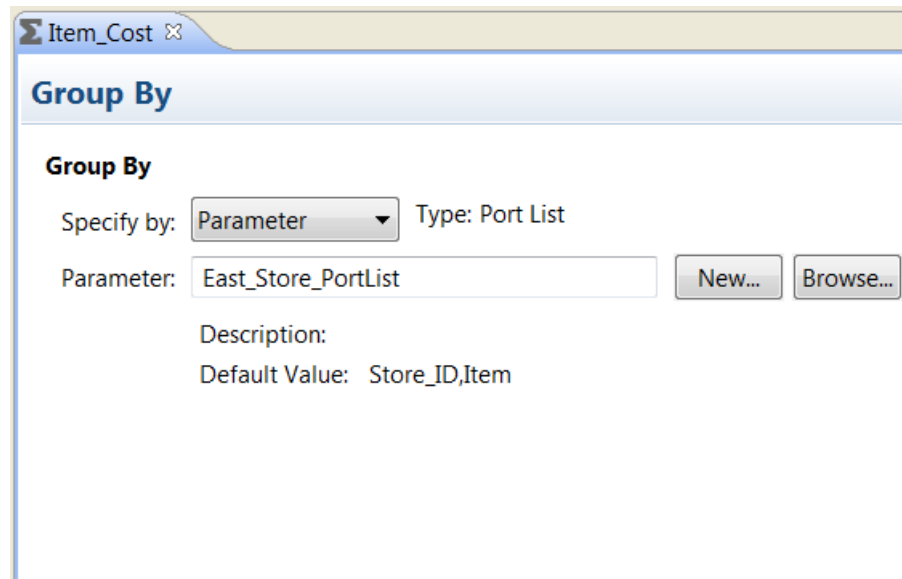
RANKINDEX	SALES_PERSON	SALES	QUARTER
5	Alex	6,000	1

Set the number of rows to include in a ranking on the **Advanced** tab of the **Properties** view.

Group By Parameters

You can configure a port list parameter that contains one or more ports to include in the group. Create a port list parameter by selecting ports from a list of the ports in the transformation.

The following image shows the **Group By** tab when you use a parameter to identify the ports in the group:



You can browse for a port list parameter or click **New** to create a port list parameter. If you choose to create a port list parameter, you can select the ports from a list of the ports in the transformation.

Rank Caches

When you run a mapping that uses a Rank transformation, the Data Integration Service creates an index cache and data cache in memory to run the transformation. If the Data Integration Service requires more space than available in the memory cache, it stores overflow data in cache files.

When you run a mapping that uses a Rank transformation, the Data Integration Service compares an input row with rows in the data cache. If the input row out-ranks a cached row, the Data Integration Service replaces the cached row with the input row. If you configure the Rank transformation to group rows, the Data Integration Service ranks rows within each group.

The Data Integration Service creates the following caches for the Rank transformation:

- Index cache that stores group values as configured in the group by ports.
- Data cache that stores information based on the group by ports.

Rank Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Rank transformation.

Configure the following properties on the **Advanced** tab:

Top/Bottom

Specifies whether you want the top or bottom ranking for a column.

Number of Ranks

Number of rows to include in the top or bottom ranking.

Case-Sensitive String Comparison

Specifies whether the Data Integration Service uses case-sensitive string comparisons when it ranks strings. Clear this option to have the Data Integration Service ignore case for strings. By default, this option is selected.

Cache Directory

Directory where the Data Integration Service creates the index cache files and data cache files. Verify that the directory exists and contains enough disk space for the cache files.

Enter multiple directories separated by semicolons to increase performance during cache partitioning. Cache partitioning creates a separate cache for each partition that processes the transformation.

Default is the CacheDir system parameter. You can configure another system parameter or user-defined parameter for this property.

Rank Data Cache Size

Amount of memory that the Data Integration Service allocates to the data cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.

Rank Index Cache Size

Amount of memory that the Data Integration Service allocates to the index cache for the transformation at the start of the mapping run. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

RELATED TOPICS:

- [“Cache Size” on page 71](#)

Rank Transformation in a Non-native Environment

The Rank transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Rank Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

The data cache for the Rank transformation is optimized to use variable length to store binary and string data types that pass through the Rank transformation. The optimization is enabled for record sizes up to 8 MB. If the record size is greater than 8 MB, variable length optimization is disabled.

When variable length is used to store data that passes through the Rank transformation in the data cache, the Rank transformation is optimized to use sorted input and a pass-through Sorter transformation is inserted before the Rank transformation in the run-time mapping. To view the Sorter transformation, view the optimized mapping or view the execution plan in the Blaze validation environment.

During data cache optimization, the data cache and the index cache for the Rank transformation are set to Auto. The sorter cache for the Sorter transformation is set to the same size as the data cache for the Rank transformation. To configure the sorter cache, you must configure the size of the data cache for the Rank transformation.

Rank Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- Case sensitivity is disabled.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

Rank Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- A Rank transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.
- A Rank transformation is upstream from a Joiner transformation.
- A streaming pipeline contains more than one Rank transformation.
- A streaming pipeline contains an Aggregator transformation and a Rank transformation.

Rank Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- Case sensitivity is disabled.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

CHAPTER 37

Read Transformation

This chapter includes the following topics:

- [Read Transformation Overview, 541](#)
- [Read Transformation Properties , 541](#)
- [Synchronize Relational Data Objects, 545](#)
- [Change the Source Data Object, 545](#)
- [Read Transformation Parameters, 547](#)
- [Constraints , 548](#)
- [Create a Read Transformation, 548](#)

Read Transformation Overview

The Read transformation is a passive transformation that reads data from a source. The Read transformation is nonreusable.

You can create a Read transformation from a physical data object or a logical data object. If you want to create a Read transformation from a physical data object that you imported from a PowerExchange® adapter source, the mapping editor might prompt you to specify a read operation before you can create a Read transformation from the data object.

You can configure different properties for a Read transformation based on the type of data object that you used to create the transformation. For example, if you create a Read transformation from a relational data object, you can configure SQL overrides and define constraints. The properties that you can configure also depend on whether you have configured parameters for the transformation.

Read transformations can contain dynamic sources. You can configure a Read transformation to dynamically update its ports, metadata, and other properties. For information about configuring dynamic sources, read the "Dynamic Mappings" chapter in the *Informatica Developer Mapping Guide*.

Read Transformation Properties

After you create a Read transformation, you can configure properties for the transformation.

Configure Read transformation properties on property tabs. The tabs you can use depend on the type of source that the Read transformation represents.

The following table describes each property tab and identifies the source type you use the tab for:

Property Tab	Description	Source Type
General	Specify transformation properties and behavior. For relational and customized data object sources, synchronize transformation input ports with source.	All
Data Object	Specify the transformation data source.	- Relational - Flat file - Customized data object
Ports	Set port definition by the associated data object.	- Relational - Customized data object - Logical data object
Format	Input settings for a flat file data source	Flat file
Query	Specify a query to the source.	- Relational - Customized data object - Logical data object
Run-time	Define run-time behavior.	- Relational - Flat file - Customized data object
Sources	Select source tables and configure source details.	- Relational - Customized data object
Data Object Parameters	Set parameter properties.	- Flat file - Customized data object - Logical data object
Run Time Linking	Configure a group-to-group link between transformations that uses a parameter, a link policy, or both to determine which ports to link at run time.	All
Advanced	Set tracing level and row order. For a relational source, set option to create or replace the target table at run time.	All

General Properties

You can configure the name and description of the Read transformation. You can also configure the following properties:

When Column Metadata Changes

Available for relational sources. Select one of the following options:

- Synchronize output ports. The Developer tool updates Read transformation output ports with metadata changes that the Model repository stores for the data object.
- Do not synchronize. The Read transformation does not show metadata changes in the data object.

Physical Data Object

Available for flat file and customized sources. The object used to create the transformation.

You can select the data object name and configure its properties.

Data Object Properties

On the Data Object tab, you can specify or change the Read transformation source, and make relational, flat file, and customized data object sources dynamic.

You can configure the following properties:

Specify By

To specify source columns and metadata for the Read transformation, select one of the following options:

- **Value.** The Read transformation uses the associated data object to specify source columns and metadata.
- **Parameter.** The Read transformation uses a parameter to specify source columns and metadata.

Data Object

If you created the Read transformation from an existing data object, the field displays the name of the object. Click **Browse** to change the data object to associate with the Read transformation.

At runtime, get data object columns from data source

When you enable this option, the Data Integration Service fetches metadata and data definition changes from source tables to the Read transformation.

Query Properties

Configure an SQL query to a relational resource or customized data object.

When you configure properties on the **Query** tab, you choose to configure simple or advanced properties.

In the **Simple** properties view, you configure the default SQL statement as a Define Distinct statement, and to edit the hints, join, filter and sort conditions for the statement.

In the **Advanced** properties view, you can define a custom SQL query. You can select from columns in the associated data object, or from parameters, or create a new parameter to represent a data object.

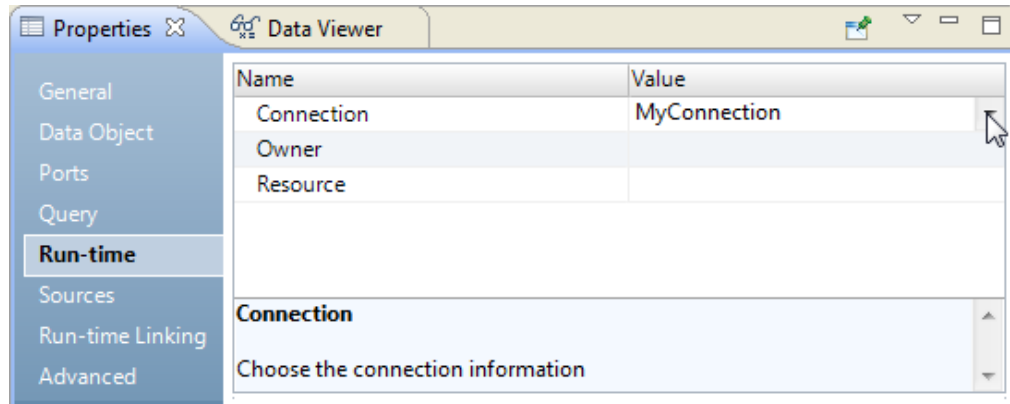
Run-Time Properties

You can configure the following Read transformation properties on the **Run-time** tab:

Connection

Available for relational sources. Connection used by the transformation. Click in the right side of the field to change the connection.

The following image shows the location of the dropdown button to click:



Sources Properties

Configure the details of sources for relational resources and customized data objects. You can change the relational data object definition after you import it to the repository. You can add and remove ports, define primary keys, and configure relationships between multiple relational data objects in the repository.

The **Sources** tab enables you to configure the following settings:

All sources

Use the Add and Remove buttons to add and remove additional sources for the transformation.

General tab

Change the name and description for the selected source. Click the source name to change other details.

Keys tab

Designate resource columns as keys.

Relationships tab

Add and remove relationships between multiple relational resources.

Advanced Properties

Configure advanced properties to determine how the Data Integration Service processes data for the Read transformation.

Configure the following properties on the Advanced tab:

Tracing level

Control the amount of detail in the mapping log file.

PreSQL

SQL command the Data Integration Service runs against the source database before it reads the source.

The Developer tool does not validate the SQL.

PostSQL

SQL command that the Data Integration Service runs against the source database after it writes to the target.

The Developer tool does not validate the SQL.

Constraints

SQL statements for table-level referential integrity constraints. Applies to relational sources only.

Synchronize Relational Data Objects

You can synchronize physical data objects when their sources change. When you synchronize a physical data object, the Developer tool reimports the object metadata from the source you select.

You can synchronize all physical data objects. When you synchronize relational data objects or customized data objects, you can retain or overwrite the key relationships you define in the Developer tool.

Choose from among several methods to synchronize mapping objects:

Synchronize a relational resource.

To synchronize any physical data object, right-click the object in the **Object Explorer** view, and select **Synchronize**.

Synchronize transformation ports with the physical data object.

In the Data Object tab of a transformation, select the option **At run time, get data object columns from the data source**.

At run time, the Data Integration Service fetches metadata and data definition changes from the data source and refreshes the data object definition in the Model repository.

To preview how the Data Integration Service fetches metadata and data definition changes, view the mapping with resolved parameters.

Synchronize ports when metadata changes

In the General tab of a transformation, select the option to synchronize ports. The exact label of this option depends on the type of transformation you configure. For example, for a Read transformation, the option is **When metadata changes, synchronize output ports**.

When the mapping runs, the Data Integration Service synchronizes column metadata in the transformation with the metadata in the data source.

Change the Source Data Object

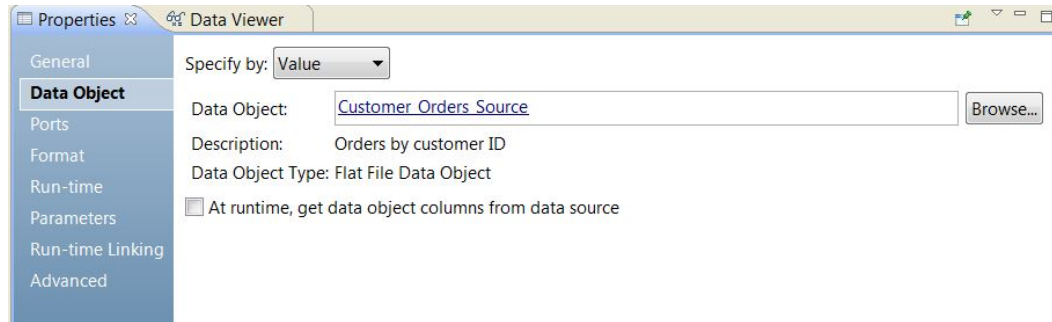
A Read transformation is based on a physical data object or a logical data object in the Model repository. You can change the data object when you configure a Read transformation. You can parameterize the data object to change the data object at run time. For example, you might test a mapping with a different source file than the source file you use for a production mapping run.

When you create a transformation from a physical data object, information about the data object appears on the **Data Object** tab of the transformation properties. You can click the data object name to view the physical data object definition from the Model repository.

You can change the data object for the transformation by browsing for a different physical data object in the Model repository. When you change the data object, the transformation uses the run time properties and the advanced properties of the data object that you select.

You can update the structure of the data object at run time based on changes in the data source. The data source is the physical file or the database table that the data object represents. When you enable the Data Integration Service to get data columns from the data source, the Data Integration Service examines the structure of the data source. The Data Integration Service updates the data object ports in the transformation instance based on the data source. The Data Integration Service does not change the physical data object definition in the Model repository.

The following image shows the **Data Object** tab:



The **Data Object** tab has the following fields:

Specify by

Choose **Value** to enter a specific data object name. Choose **Parameter** to parameterize the data object.

Data object

The name of the data object in the Model repository. You can click the **Data Object** link to open the data object definition from the repository. You can also Browse for a different data object in the Model repository.

Description

The description of the data object in the repository. Read-only.

Data object type

Describes the type of data object, such as a flat file data object, a relational table object, or a customized data object.

At run time, get data object columns from the data source

The Data Integration Service fetches metadata and data definition changes from the data file or the table that the data object refers to and updates the structure of the data object for the transformation instance at run time.

To preview how the Data Integration Service fetches metadata and data definition changes at run time, view the mapping with resolved parameters.

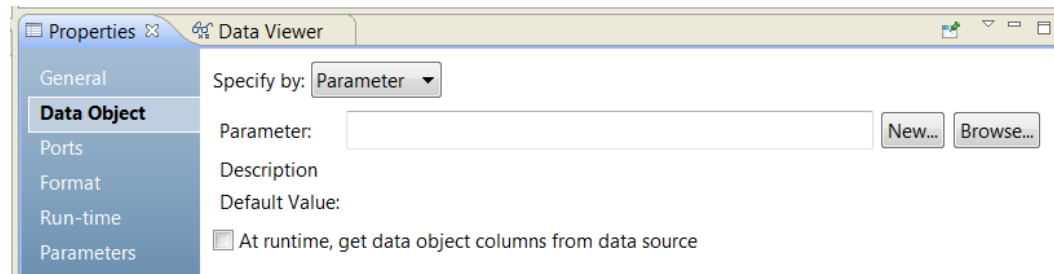
Parameterize the Read Transformation

You can parameterize the Read transformation and change the data object at run time.

To parameterize a data object, choose **Specify by Parameter** in the **Data Object** tab. The properties in the **Data Object** tab change.

To parameterize the data object, create a resource type parameter or browse for a resource parameter that you already created. The parameter default value is the name of physical data object in the Model repository. When you create a default parameter value, you select a physical data object name from a list of data objects in the repository.

The following image shows the **Data Object** tab when you specify the data object by a parameter:



The **Data Object** tab has the following options by parameter:

Parameter

The name of a resource parameter that you configured as the data object. Read-only.

Description

The description of the parameter. Read-only.

New

Create a resource parameter. Browse for and select a data object in the Model repository for the parameter default value.

Browse

Browse for a resource parameter and select the parameter.

Default value

The default value of the resource parameter that you configured for the data object. The default value is a physical data object name and the path to the object in the Model repository. Read-only.

Read Transformation Parameters

You can parameterize some of the properties of a Read transformation and some of the properties of the reusable physical data object that you create the Read transformation from.

When you create a physical data object, you configure the read and write properties. The parameters that you configure for read properties in a physical data object appear in the Read transformation **Data Object Parameters** tab when you add the data object to a mapping.

You can configure parameters for the following Read properties in the physical data object:

- Control file directory
- Control file name
- Default scale
- Delimiter
- Flat file delimiter
- Merge file directory
- Source file name
- Source file directory

After you add the physical data object to a mapping, you can view the parameters in the **Data Object Parameters** tab of the Read transformation. You can expose these parameters as mapping parameters to override the parameter values at run time.

Note: You cannot nest user-defined parameters within a parameterized source. If the source data object is parameterized, you cannot expose a user-defined parameter as a mapping parameter to override the parameter values at run time. The mapping uses the default value instead.

You can configure the following mapping parameters for the Read transformation:

- Connection (relational)
- Data object
- Link resolution order
- Resource name (relational)
- Table owner name (relational)

You can view these parameters in the mapping **Data Object Parameters** tab.

Constraints

A constraint is a conditional expression that the values on a data row must satisfy.

When you set a constraint, you enter an expression that evaluates to TRUE for each data row.

The Data Integration Service can read constraints from relational sources, logical data objects, physical data objects, or virtual tables. To set a constraint on a reusable physical data object, create a customized data object.

When the Data Integration Service reads constraints, it might drop the rows that do not evaluate to TRUE for the data rows based on the optimization method applied.

Before you set a constraint, you must verify that the source data satisfies the condition set by the constraint. For example, a source database has an AGE column that appears to have rows with AGE < 70. You can set a constraint with AGE < 70 on the source database. The Data Integration reads records from the source database with the constraint AGE < 70. If the Data Integration Service reads records with AGE >= 70, it might drop the rows with AGE >= 70.

In the database, you can use SQL commands to set constraints on the database environment when you connect to the database. The Data Integration Service runs the connection environment SQL each time it connects to the database.

Create a Read Transformation

When you create a Read transformation, you choose one of the following methods based on the resource you create the transformation from:

Create the transformation from a data object in the Model repository.

Perform the following steps to create a Read transformation from a data object in the Model repository:

1. Open the mapping in the editor.

2. Drag a data object from the **Object Explorer** to the editor view.
3. Select **Read** and click **OK**.
The Read transformation in the mapping contains the ports and properties of the data object.

Create the transformation using the mapping editor.

Use this method if you want to configure detailed Read transformation settings. You can also use this method if you want to base a Read transformation on a parameter.

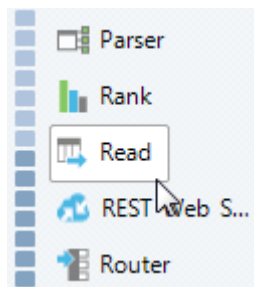
To create a Read transformation in the mapping editor, see [“Creating a Read Transformation in the Mapping Editor” on page 549](#).

Creating a Read Transformation in the Mapping Editor

You can create a Read transformation to represent the source of data and column metadata and properties in a mapping.

Perform the following steps:

1. Choose one of the following methods to create a Read transformation:
 - Right-click in the mapping editor and select **Add Transformation**.
The **Add Transformation** dialog box opens.
Select Read transformation and click **Next**.
 - Scroll down in the mapping palette to locate and double-click the Read transformation icon.
The following image shows the Read transformation icon:



The **New Read Transformation** dialog box opens.

2. To use a flat file, a relational resource, or a customized data object as the source, perform the following steps:
 - a. Select **Physical Data Object** as the data object type.
 - b. Click **Browse** to select a flat file, a relational resource, or a customized data object.
The **Select Data Object** window opens.
 - c. Select a data object and click **OK**.
 - d. Optionally, configure the transformation to fetch data object columns from the source at run time.
Select **At run-time, get data object columns from data source**.
The Data Integration Service refreshes column metadata for the Read transformation when the mapping runs.
3. To use a parameter as the source, perform the following steps:
 - a. Select **Create using a parameter**.
 - b. Click **New** to create a new parameter, or click **Browse** to select an existing parameter.

- c. Select a parameter and click **OK**.
 - d. Optionally, configure the transformation to fetch data object columns from the source at run time. Select **At run-time, get data object columns from data source**.
The Data Integration Service refreshes column metadata for the Read transformation when the mapping runs.
4. To use a logical data object as the source, perform the following steps:
 - a. Select **Logical Data Object** as the data object type.
 - b. Click **Browse** to select a data object, and then click **OK**.
 5. Optionally, type a name for the Read transformation.
 6. Click **Finish**.

CHAPTER 38

Relational to Hierarchical Transformation

This chapter includes the following topics:

- [Relational to Hierarchical Transformation Overview, 551](#)
- [Example - Relational to Hierarchical Transformation, 552](#)
- [Input Relational Ports and the Overview View, 554](#)
- [Relational to Hierarchical Transformation Ports, 554](#)
- [Schema References, 555](#)
- [Relational to Hierarchical Transformation Development, 555](#)

Relational to Hierarchical Transformation Overview

The Relational to Hierarchical transformation processes relational input and transforms it into hierarchical output. A Relational to Hierarchical transformation reads relational input from input ports and transforms the data to hierarchical output at the transformation output port. To transform relational input to hierarchical output, use a schema object to define the hierarchical structure.

You can use the Relational to Hierarchical transformation wizard to create a hierarchical structure that reflect the relational input ports. You can view the mapping for the hierarchical output ports in the transformation **Overview** view.

After you create the transformation, you can pass the data from the hierarchical output port to another transformation in a mapping.

Note: The Relational to Hierarchical transformation can process up to 10,000 schema elements in an .xsd file. To process more than 10,000 elements, split the data into multiple files.

In the relational model, each table schema identifies a column, called the primary key, to uniquely identify each row. You identify the relationship between each row in the table and a row in another table with a foreign key. The wizard generates keys when it creates the transformation. You can change an auto-generated transformation, and add, edit, or delete ports.

You can link an input relational port to a node in the hierarchy. Link a primary key from the relevant element or attribute in the hierarchy to a relational group in the input. The primary key identifies each row in the relational tables.

Link a foreign key from the relevant element or attribute in the hierarchy to a relational group in the input. A foreign key in the relational input is a column in one table that points to the primary key of another table.

The input relational port and the hierarchical node must have compatible data types.

Example - Relational to Hierarchical Transformation

The Finance department of the Electronics Superstore company must process paychecks for company employees. They need to transform employee data stored in a relational database into a hierarchical format that their payment system can process.

The mapping needs to use a Relational to Hierarchical transformation that inputs employee details such as employee name, employee ID, employee address, and employee bank account data, and outputs the details in a usable hierarchical format.

In the relational input, the Bank_ID element is a primary key in the Employee table, and a Foreign key in the Bank table:

Employee_ID	Last_Name	First_Name	Address	Bank_ID	Bank_Account
9173327437	Sandrine	Jacques	74 Mobile Avenue	74845	8723487234
9174562342	Race	Tom	266 Crouse St.	9234734	45324734
8484526471	Jones	Charles	3815 LaValle Boulevard	389236	234638437
7023847265	Smith	Delilah	193 Short Drive	74845	8723463432
9174596725	Frederick	George	17 Serenity Road	9234734	6342636699

Bank_ID	Bank_Name	SWIFT_Code
74845	National Bank	9173327
9234734	International Bank	9174562
389236	Star National Bank	8484526

In the Payment output in hierarchical format, the elements are combined from the tables:

```
<banks>
  <bank name="National Bank" SWIFT="9173327">
    <account id="8723487234">
      <employee_id>9173327437</employee_id>
      <fname>Sandrine</fname>
      <lname>Jacques</lname>
      <address>74 Mobile Avenue</address>
    </account>
    <account id="8723463432">
      <employee_id>9082745558</employee_id>
      <fname>Delilah</fname>
      <lname>Smith</lname>
      <address>193 Short Drive</address>
    </account>
  </bank>
  <bank name="International Bank" SWIFT="9174562">
    <accounts>
      <account id="45324734">
        <employee_id>5534398889</employee_id>
        <fname>Race</fname>
        <lname>Tom</lname>
        <address>266 Crouse St.</address>
      </account>
      <account id="6342636699">
        <employee_id>9174596725</employee_id>
        <fname>Frederick</fname>
        <lname>George</lname>
        <address>17 Serenity Road</address>
      </account>
    </accounts>
  </bank>
  <bank name="Star National Bank" SWIFT="8484526">
    <accounts>
```

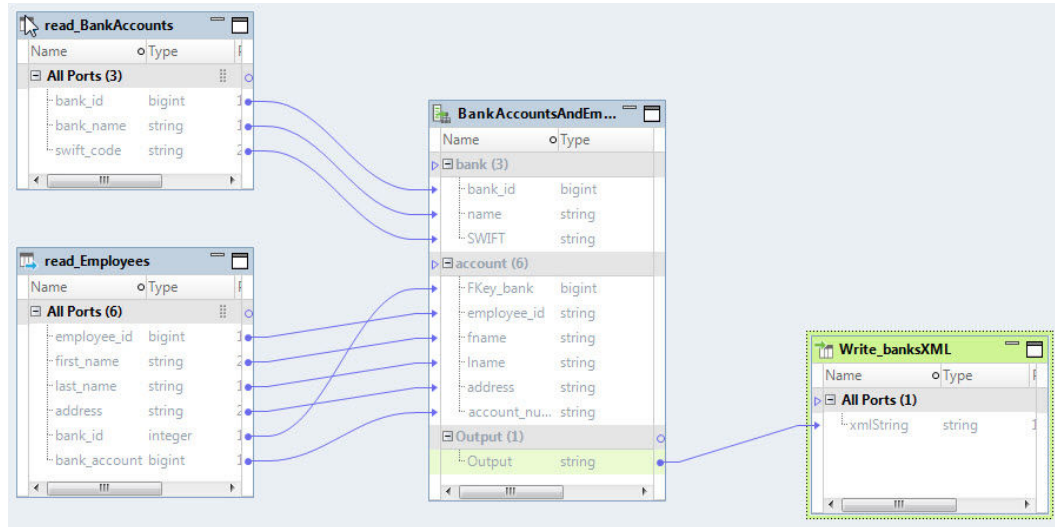


```

<account id="234638437">
  <employee_id>8484526471</employee_id>
  <fname>Jones</fname>
  <lname>Charles</lname>
  <address>3815 LaValle Boulevard</address>
</account>
</accounts>
</bank>
</banks>

```

The following image shows the mapping in this example:



The mapping contains the following objects:

Read_BankAccounts

The source that contains the bank data.

Read_Employees

The source that contains the employee data.

BankAccountsAndEmployees_To_PaymentsSystemXML

A Relational to Hierarchical transformation that transforms relational input that contains employee and bank account information into an XML format that the payment system consumes.

Write_BanksXML

A target path to the file that stores the transformed data every time you run the mapping.

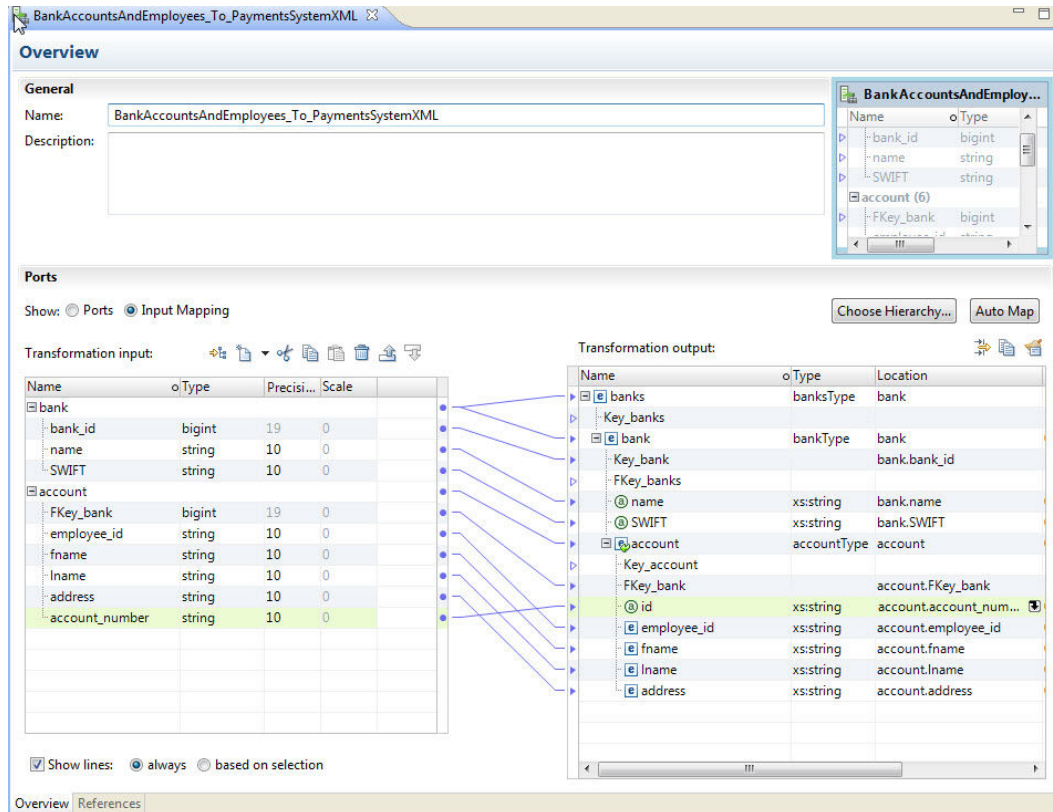
The mapping uses the Read_BankAccount and Read_Employees files to provide relational input. The mapping processes and transforms the data with the BankAccountsAndEmployees_To_PaymentsSystemXML transformation. Then the mapping stores the output in the target path listed in the Write_BanksXML flat file.

Input Relational Ports and the Overview View

To transform relational data to hierarchical data, the wizard creates a hierarchical structure that reflects the relational input ports. You can use the **Overview** View to link relational ports to hierarchical ports.

To view the links between relational input to hierarchal output, use the **Overview** view. Select **Input Mapping**. The **Ports** panel appears in the **Overview** view.

The following image shows the **Ports** panel:



To the left of the **Ports** panel is the **Transformation input** area that contains the relational elements and groups. To the right is the **Transformation output** area that contains the hierarchical schema nodes.

You can create ports in the **Transformation input** area, and link relational elements to the schema nodes. You can also drag the pointer from a node in the schema to an empty field in the **Transformation input** area to create a port. When you connect a relational port to a schema node, the Developer tool shows a link between them.

Relational to Hierarchical Transformation Ports

The Relational to Hierarchical transformation ports are defined on the transformation **Overview** view.

A Relational to Hierarchical transformation can read relational data output from a buffer. The output ports return hierarchical data to a buffer.

Schema References

A Relational to Hierarchical transformation requires a hierarchical schema to define the output hierarchy in the transformation. To use the schema in the transformation, you define a schema reference.

You can define transformation schema references in the transformation **References** view.

The Relational to Hierarchical transformation references schema objects in the Model repository. The schema objects can exist in the repository before you create the transformation.

A schema can reference additional schemas. The **References** view shows the namespace and prefix for each schema that the Relational to Hierarchical transformation references.

Relational to Hierarchical Transformation Development

Use the New Transformation wizard to auto-generate a Relational to Hierarchical transformation. Choose a schema or hierarchical sample file to define the output hierarchy.

Creating the Relational to Hierarchical Transformation

1. In the Developer tool, click **File > New > Transformation**.
2. Select the Relational to Hierarchical transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model repository location to put the transformation and click **Next**.
4. To select a schema, select one of the following methods:
 - To use a schema from the Model repository to define the output hierarchy, near the **Schema Object** field, browse to select the schema file from the repository.
 - To import a schema file, click **Create a new schema object**. In the **New schema object** window, you can browse to and select a schema file, or you can select to create a schema from a sample hierarchical file.
5. Choose the root for the output hierarchy. In the **Hierarchy root** dialog box, select the element in the schema that is the root element for the output hierarchical file. To help select the root object, you can add a sample hierarchical file. To add a sample file, near the **Sample File** field, browse for and select the file from the file system.
6. Click **Finish**.

The wizard creates the transformation in the repository.

Creating the Ports

Configure the ports in the **Overview** view.

1. To view the mapping, in the **Overview** view **Ports** area, select **Input Mapping**.
2. Select the input port type, precision and scale.
3. Expand the trees in the **Ports** grid. To the left, the **Transformation input** panel shows the relational input, and to the right, the **Transformation output** panel shows the expected hierarchical output.

4. To define a node as a root, click **Choose Hierarchy**.
The Developer tool displays only the nodes from the root level and below the root level in the **Transformation input** area.
5. To view lines that connect the ports with the hierarchical nodes, click **Show Lines**. Select to view all the connection lines or just the lines for selected ports.
6. To add an input group or port to the **Transformation input** area, use one of the following methods:
 - Drag a simple or complex element in the **Transformation output** area to an empty column in the **Transformation input** area. If the node is a group node, the Developer tool adds a relational group without ports.
 - To add a relational group, select a row and right-click to select **New > Group**.
 - To add a relational port, right-click to select **New > Field**.
7. To clear the hierarchical node settings for locations of ports, use one of the following methods:
 - Select one or more nodes in the **Transformation output** area, right-click and select **Clear**.
 - Select one or more lines that connect the relational input ports to the hierarchical nodes, right-click and select **Delete**.
8. To display the output ports in a hierarchy, click **Show As Hierarchy**. Each child group appears underneath the parent group.

CHAPTER 39

REST Web Service Consumer Transformation

This chapter includes the following topics:

- [REST Web Service Consumer Transformation Overview, 557](#)
- [REST Web Service Consumer Transformation Configuration, 559](#)
- [HTTP Methods, 560](#)
- [REST Web Service Consumer Transformation Ports, 563](#)
- [REST Web Service Consumer Transformation Input Mapping, 566](#)
- [REST Web Service Consumer Transformation Output Mapping, 568](#)
- [REST Web Service Consumer Transformation Advanced Properties, 570](#)
- [REST Web Service Consumer Transformation Creation, 571](#)
- [Parsing a JSON Response Message that Contains Arrays, 572](#)

REST Web Service Consumer Transformation Overview

The REST Web Service Consumer Transformation is an active transformation that connects to a REST web service as a web service client to access or transform data. Use a REST Web Service Consumer transformation to connect to a REST web service. The REST Web Service Consumer transformation can send a request to a REST web service and receive a response from a REST web service.

The REST Web Service Consumer transformation connects to a web service through a URL that you define in the transformation or in an HTTP connection. You can also use an HTTPS connection. REST Web Service Consumer transformations can use TLS 1.2, TLS 1.1, or TLS 1.0.

A REST web service contains an HTTP method for each action that the web service supports. When the Data Integration Service connects to a REST web service, it can send a request to get, post, put, or delete data. The request can act upon individual resources or collections of resources. After the Data Integration Service sends a request message, it receives a response message from the web service.

The request and the response messages contain XML or JSON data with elements that can form a hierarchy. When a request or a response message contains multiple-occurring elements, groups of elements form levels in the XML or JSON hierarchy. Groups are related when one level is nested within another.

In the REST Web Service Consumer transformation, the method input and method output define the structure of the request and the response messages. The method input and method output include mappings that define how to map the message elements to the input and output ports.

The REST Web Service Consumer transformation supports proxy server. You can also connect to a Microsoft SharePoint application with the REST Web Service Consumer transformation.

Example

An online store defines resources for a products database. The database identifies each product by part number.

Web service clients access the product details through a REST web service. The web service uses the following URL:

```
http://www.HypoStores.com/products/ProductDetails
```

You need to retrieve details about a specific product, such as the description and the unit price, and pass the details to a transformation downstream in a mapping. Create a REST Web Service Consumer transformation to retrieve details about a product and pass them to another transformation..

The following table shows the transformation details that you configure:

Transformation Detail	Value
HTTP method	Get
Base URL	http://www.HypoStores.com/products/ProductDetails
Input argument port	Part_No
Output ports	Description, Unit_Price
Method output	<The structure of the response message.>

The method output includes an output mapping that defines how the elements in the response message map to the output ports.

When the Data Integration Service sends the request to the web service, it appends the value in the argument port to the base URL. For example, to retrieve details about part 0716, the Data Integration Service uses the following URL:

```
http://www.HypoStores.com/products/ProductDetails?Part_No=0716
```

When the Data Integration Service receives a response, it converts the product description and the unit price in the response message to data for the output ports.

You can also pass Part_No as a parameter and substitute the value mid-stream when you run the mapping.

REST Web Service Consumer Transformation Process

The REST Web Service Consumer transformation creates a request message based on the data in the input ports and the method input. It converts elements in the response message to data for the output ports based on the method output.

The input ports of the REST Web Service Consumer transformation contain relational data from upstream transformations in a mapping. The Data Integration Service uses the method input to convert data from the input ports to elements in the request message.

To connect to the web service, the Data Integration Service reads the base URL that you configure in the transformation properties or the HTTP connection. It identifies the resource that you want to get, post, put, or delete by appending values from the URL ports or argument ports to the base URL.

When the Data Integration Service receives a response, it passes the data in the response message to the output ports of the transformation. The Data Integration Service passes data based on how you configure the method output. The output ports contain relational data. The Data Integration Service sends the data in the output ports to downstream transformations in the mapping or to the target.

REST Web Service Consumer Transformation Configuration

When you create a REST Web Service Consumer transformation, you select the HTTP method and define the method input and output. If you select the Get method, you do not define method input.

The input elements in the HTTP request message map to input ports. The output elements in the HTTP response message map to output ports. The Developer tool creates ports for the first level elements.

When you configure the transformation, you complete the following tasks:

1. Select the HTTP method.
2. Configure ports to represent elements in the header and body of the request and response messages.
3. Configure the input mapping.
4. Configure the output mapping.
5. Configure advanced properties such as the connection and the base URL for the web service.

If the REST web service requires authentication, create an HTTP connection object.

Message Configuration

The Data Integration Service generates request messages and interprets response messages based on the method input and output and on the ports that you configure in the REST Web Service Consumer transformation.

Input ports represent different parts of the request message. You can add input ports that identify the resource that you want to retrieve or change. You can also add input ports that represent HTTP headers, cookie information, and elements in the request message.

Output ports represent elements in the response message that you want to send to downstream transformations or to the target in a mapping. You can add output ports that represent HTTP headers, cookie information, the response code, and elements in the response message.

Resource Identification

To identify the resource in an HTTP request, the Data Integration Service appends values in specific input ports to the base URL. You define the base URL in the HTTP connection or in the transformation properties. Use URL or argument ports to identify a particular resource.

Use URL ports when the web service identifies a resource through a unique string of characters.

For example, the HypoStores REST web service identifies parts by the part number through the following URL:

```
http://www.HypoStores.com/products/ProductDetails/<Part_No>
```

To identify a part, define the following transformation details:

1. Set the base URL to the following URL:

```
http://www.HypoStores.com/products/ProductDetails
```

2. Define a URL port, and pass the part number to the transformation through the URL port.

If the mapping passes part number 500 to the URL port, the Data Integration Service uses the following URL in the request message:

```
http://www.HypoStores.com/products/ProductDetails/500
```

Use argument ports when the web service identifies the location of a resource through arguments.

For example, you want to pass a part number to the HypoStores REST web service through the "Part_No" argument.

To identify a part, define the following transformation details:

1. Set the base URL to the following URL:

```
http://www.HypoStores.com/products/ProductDetails
```

2. Create an argument port with argument name "Part_No," and pass the part number to the transformation through the argument port.

If the mapping passes part number 600 to the argument port, the Data Integration Service uses the following URL in the request message:

```
http://www.HypoStores.com/products/ProductDetails?Part_No=600
```

Create multiple argument ports to define multiple arguments. The Data Integration Service separates each argument with an ampersand character (&).

For example, you want to retrieve employee details from a REST web service and pass the employee first name and last name through the "First_Name" and "Last_Name" arguments. Create argument ports with the argument names "First_Name" and "Last_Name." If the mapping passes the name "John Smith" to the transformation, the Data Integration Service uses a URL like the following in the request message:

```
http://www.HypoStores.com/employees/EmpDetails?First_Name=John&Last_Name=Smith
```

If you do not specify a URL or an argument port, the Data Integration Service uses the base URL from the transformation properties or HTTP connection to identify the resource. The base URL in the HTTP connection overrides the base URL in the transformation.

HTTP Methods

When you create a REST Web Service Consumer transformation, you select the HTTP method that the Data Integration Service uses in the request message. You cannot change the HTTP method after you create the transformation.

You configure the transformation to use one of the following HTTP methods:

Get

Retrieves a resource or a collection of resources from the web service. For example, you can retrieve a table of products or retrieve information about one product.

Post

Sends data to a web service. Use the Post method to create a resource or collection of resources. For example, you can add the details of a new store transaction.

Put

Replaces a resource or a collection of resources. If the data does not exist, the Put method posts the data. For example, you can update a customer shipping address.

Delete

Deletes a resource or a collection of resources. For example, you can delete the record of an employee that no longer works for an organization.

HTTP Get Method

The Data Integration Service uses the HTTP Get method to retrieve data from a REST web service. Use the Get method to retrieve a resource or a collection of resources.

When you configure the REST Web Service Consumer transformation to use the Get method, you configure the input ports, the method output, and the output ports. You do not configure method input.

Example

You want to retrieve the description and price for part number 500 in the HypoStores products database. The web service uses the following URL to identify a part:

```
http://www.HypoStores.com/products/ProductDetails?Part_No=<Part_No>
```

Enter the following base URL:

```
http://www.HypoStores.com/products/ProductDetails
```

The following table shows the input port that you might define:

Port Type	Argument Name	Input Value
Argument	Part_No	500

The following table shows the output ports that you might define:

Port Type	Port Name	Return Value
Output	Part_Desc	...<desc>ACME ball point pens, 12-pk, black, 0.7 mm</desc>...
Output	Price_USD	...<price>9.89</price>...

HTTP Post Method

The Data Integration Service uses the HTTP Post method to send data to a REST web service. The web service determines the actual function that the Post method performs. You might use the Post method to create a resource or a collection of resources.

When you configure the REST Web Service Consumer transformation to use the Post method, you configure the input ports, the method input, the method output, and the output ports.

Example

You want to post new part 501 to the HypoStores products database. The web service uses the following URL for part 501:

```
http://www.HypoStores.com/products/ProductDetails/501
```

Enter the following base URL:

`http://www.HypoStores.com/products/ProductDetails`

The following table shows the input ports that you might define:

Port Type	Port Name	Input Value
URL	URL_Part_No	501
Input	Part_Desc	ACME ball point pens, 12-pk, black, 0.5 mm
Input	Price_USD	9.89

The following table shows the output ports that you might define:

Port Type	Port Name	Return Value
Output	Response	<Response returned by the web service>

HTTP Put Method

The Data Integration Service uses the HTTP Put method to update data through a REST web service. Use the Post method to update a resource or a collection of resources. If the data does not exist, the Data Integration Service creates the resource or collection of resources.

When you configure the REST Web Service Consumer transformation to use the Put method, you configure the input ports, the method input, the method output, and the output ports.

Example

You want to update the unit price for part 501 in the HypoStores products database. The web service uses the following URL for part 501:

`http://www.HypoStores.com/products/ProductDetails/501`

Enter the following base URL:

`http://www.HypoStores.com/products/ProductDetails`

The following table shows the input ports that you might define:

Port Type	Port Name	Input Value
URL	URL_Part_No	501
Input	Price_USD	9.99

The following table shows the output ports that you might define:

Port Type	Port Name	Return Value
Output	Response	<Response returned by the web service>

HTTP Delete Method

The Data Integration Service uses the HTTP Delete method to remove data through a REST web service. Use the Delete method to remove a resource or a collection of resources.

When you configure the REST Web Service Consumer transformation to use the Delete method, you configure the input ports, the method input, the method output, and the output ports.

Example

You want to delete part number 502 from the HypoStores products database. The web service uses the following URL to identify a part:

```
http://www.HypoStores.com/products/ProductDetails?Part_No=<Part_No>
```

Enter the following base URL:

```
http://www.HypoStores.com/products/ProductDetails
```

The following table shows the input port that you might define:

Port Type	Argument Name	Input Value
Argument	Part_No	502

The following table shows output ports that you might define:

Port Type	Port Name	Return Value
Output	Response	<Response returned by the web service>

REST Web Service Consumer Transformation Ports

A REST Web Service Consumer transformation can have multiple input ports and multiple output ports. You create ports in groups based on the structure of the XML or JSON hierarchy.

When you view the transformation ports, show the ports if you do not need to view the XML or JSON hierarchy. When you show the ports, you can define groups, define ports, and map elements from the method input and output to input and output ports.

A REST Web Service Consumer transformation can have multiple input groups and multiple output groups. When you create ports, create groups and add the ports to the groups. Define the ports in a group hierarchy based on the structure of the input or output hierarchy in the XML or JSON . Add a key to relate a child group to a parent group.

All groups except the lowest group in the hierarchy must have primary keys. All groups in the hierarchy except the root group must have foreign keys.

The transformation has a root input group named RequestInput. You must add a primary key to the root input group. The key must be string, bigint, or integer. You can configure any port in the root input group as a pass-through port.

To map an element to a port, click the field in the **Location** column and expand the hierarchy in the **Select Location** dialog box. Then, choose an element from the hierarchy.

Input Ports

Input ports represent data from an upstream transformation or source that you want to pass to the web service. You can configure multiple input ports. Each input port maps to an element in the request message.

To add an input port, select an input group, click the arrow next to the **New** button, and select **Field**.

Output Ports

Output ports represent elements in the response message that you want to pass to a downstream transformation or to the target. You can configure multiple output ports. Each output port maps to an element in the response message.

To add an output port, select an output group, click the arrow next to the **New** button, and select **Field**.

Pass-through Ports

Pass-through ports pass data through the transformation without changing the data. You can configure any port in the root input group as a pass-through port.

To add a pass-through port, add a port to the root input group. Then right-click the port and select **Map**.

Argument Ports

Argument ports allow you to identify a resource when the URL for the resource takes an argument. Add argument ports to the root input group.

An argument port has a port name and an argument name. If an argument name contains a character that is not allowed in a port name, enter an argument name that differs from the port name. For example, you want to pass the argument "Cust-ID" to the web service, but the Data Integration Service does not allow the dash character (-) in port names. Enter "Cust-ID" as the argument name, but enter "CustID" as the port name.

The Data Integration Service appends the argument names and values for each argument port to the base URL as name=value pairs. You can configure multiple argument ports. The Data Integration Service separates multiple arguments in the request with an ampersand character (&).

For example:

```
http://www.HypoStores.com/customers/CustDetails?Last_Name=Jones&First_Name=Mary
```

If you define argument ports and URL ports in the transformation, the Data Integration Service appends the URL port values to the base URL, followed by the argument names and values.

To add an argument port, right-click on the root input group and select **New > Argument Ports**. Enter the argument name and the port name.

URL Ports

URL ports allow you to identify a resource through a static URL. To identify a resource, the Data Integration Service appends the value of the URL port to the base URL.

For example:

```
http://www.HypoStores.com/products/ProductDetails/<URL_port_value>
```

Add URL ports to the root input group.

You can configure multiple URL ports. The Data Integration Service separates the values in each URL port with a slash character (/). If you define URL ports and argument ports in the transformation, the Data

Integration Service appends the URL port values to the base URL, followed by the argument names and values.

To add a URL port, right-click on the root input group and select **New > URL Ports**.

HTTP Header Ports

HTTP header ports represent HTTP headers in the request message. You can configure multiple HTTP header ports.

To pass header information to the web service in the request, add the port to the root input group. You can configure one HTTP header port for the root input group. If you add an HTTP header to the root input group, you can configure it as a pass-through port.

An HTTP header port has a port name and an HTTP header name. If an HTTP header name contains a character that is not allowed in a port name, enter an HTTP header name that differs from the port name. For example, you want to pass the header name "Content-Type" to the web service, but the Data Integration Service does not allow the dash character (-) in port names. Enter "Content-Type" as the HTTP header name, but enter "ContentType" as the port name.

To add an HTTP header port, right-click on the root input group and select **New > HTTP Header**. Enter a header name and port name.

Cookie Ports

You can configure the REST Web Service Consumer transformation to use cookie authentication. The remote web server tracks the web service consumer users based on the cookies. You can increase performance when a mapping calls a web service multiple times.

To pass cookie information to the web service in the request, add the port to the root input group. You can configure one cookie port for the root input group. If you add a cookie port to the root input group, you can configure it as a pass-through port.

To extract cookie information from the response, add a cookie port to an output group. You can configure one cookie port for each output group.

When you project the cookie port to a web service request message, the web service provider returns a cookie value in the response message. You can pass the cookie value to another transformation downstream in the mapping, or you can save the cookie value in a file. When you save the cookie value in a file, you can configure the cookie as input to the REST Web Service Consumer transformation.

To add a cookie port, right-click on the root input group and select **New > Other Ports**. Then, select **Cookie**, and click **OK**.

Output XML Ports

Output XML ports represent responses from the web service. Output XML ports are string ports.

Add an output XML port to an output group. You can configure one output XML port for each output group. right-click on the root input group and select **New > Other Ports**. Then, select **Output XML**, and click **OK**.

Response Code Ports

Response code ports represent the HTTP response codes from the web service. Response code ports are integer ports.

Add a response code port to an output group. You can configure one response code port for each output group.

To add a response code port, select an output group, right-click on the root input group and select **New > Other Ports**. Then, select **Response Code**, and click **OK**.

REST Web Service Consumer Transformation Input Mapping

When you view the transformation ports, show the input mapping to view the method input hierarchy. When you show the input mapping, you can define input groups, define input ports, and map input ports to method input elements.

The input mapping includes the following areas:

Ports

Create the transformation input groups and input ports in the **Ports** area.

Method Input

The **Method Input** area shows the elements in the request message that the REST Web Service Consumer transformation sends to the web service. If you use a schema object to create the transformation, the schema object defines the method input hierarchy.

After you create input ports, map the input ports from the **Ports** area to the elements in the **Method Input** area. When you map an input port to an element in the method input, the location of the port appears in the Location column in the **Method Input** area.

The Developer tool maps elements in the first level of the method input to input ports when you choose to map the first level of the input hierarchy. The Developer tool also creates the ports to perform the mapping. If the first level of the hierarchy contains a multiple occurring parent element with one or more multiple occurring child elements, the Developer tool does not map the first level of the hierarchy.

You can choose to view the lines that connect the input ports to the elements in the method input.

Rules and Guidelines to Map Input Ports to Elements

Review the following rules when you map input ports to elements in the method input hierarchy:

- You can map an input port to one element in the hierarchy. You can map the same port to any number of keys in the hierarchy.
- The input port and the element must have compatible datatypes.
- You can map ports from one input group to multiple hierarchy levels in the method input.
- You must map input ports to the keys in the method input. Any port that you map to a key must be of string, integer, or bigint datatype. Map data to the keys in all levels in the method input above the hierarchy level that you are including in the request message. Include the foreign keys for all levels above and including the level you are mapping.

Note: You do not have to map input ports to keys if you are mapping only the lowest level of the method input hierarchy.

- You must map the RequestInput root element to the child element of Rest_Consumer_input group for method input definition.
- You can map multiple string, bigint, or integer input ports to a key in the **Method Input** area to create a composite key. When you click the **Location** field for a composite key, you can reorder the input ports or remove one of the ports.
- If the web service produces a JSON document, ensure that xmlRoot is the first node in the response hierarchy. If xmlRoot is not the first node for a web service with JSON response, null values may appear.

Review the following rule when you map the input ports to elements with the method input as JSON:

- Ensure that the data you pass to an input port does not contain leading zeroes followed by a number. If the data contains leading zeros, the zeroes are truncated in the corresponding values of the generated JSON request and the data type of the value changes from string to the numeric data type.

Mapping Input Ports to the Method Input

When you show the transformation input mapping, you can define input groups, define input ports, and map input ports to method input elements.

1. Open a REST Web Service Consumer transformation.
2. In the **Ports** view, show the input mapping.
3. Define a primary key for the root input group.
4. To add an input group or port to the **Ports** area, use one of the following methods:

Method	Description
Drag an element.	Drag a group or a child element from the Method Input area to an empty column in the Ports area. If you drag a group to the Ports area, the Developer tool adds a group without ports.
Manually add a group or port.	To add a group, click the arrow next to the New button, and then click Group . To add a port, click the arrow next to the New button, and then click Field .
Drag a port from another transformation.	In the editor, drag a port from another transformation to the REST Web Service Consumer transformation.
Copy a port.	Select ports from another transformation and copy them to the Ports area. To copy ports, you can use keyboard shortcuts or you can use the Copy and Paste buttons in the Developer tool.
Select Map first level of hierarchy .	The Developer tool maps elements in the first level of the method input to input ports and groups. The Developer tool also creates the input ports and groups to perform the mapping.

5. If you manually create a port or you copy a port from another transformation, click the **Location** column in the **Method Input** area and choose a port from the list.

6. To map input ports as a composite key, use one of the following methods:

Method	Description
Drag input ports.	Select two or more input ports, and drag them to a key in the method input hierarchy.
Select input ports from the Select Location dialog box.	Click the Location column of a key in the method input hierarchy and then select the input ports.

7. To clear the locations of elements, use one of the following methods:

Method	Description
Click Clear .	Select one or more elements in the Method Input area and click Clear .
Delete the lines that connect ports to elements.	Select one or more lines that connect the input ports to the elements in the method input, and press Delete .

REST Web Service Consumer Transformation Output Mapping

When you view the transformation ports, show the output mapping to view the method output hierarchy. When you show the output mapping, you can define output groups, define output ports, and map method output elements to output ports.

The output mapping includes the following areas:

Method Output

The **Method Output** area shows the elements in the response message that the web service returns to the REST Web Service Consumer transformation. If you use a schema object to create the transformation, the schema object defines the method output hierarchy.

Ports

Create the transformation output groups and ports in the **Ports** area.

After you create output ports, map the elements from the **Method Output** area to the ports in the **Ports** area. When you map an element from the method output to an output port, the location of the element appears in the **Location** column in the **Ports** area.

The Developer tool maps elements in the first level of the method output to output ports when you choose to map the first level of the output hierarchy. The Developer tool also creates the ports to perform the mapping. If the first level of the hierarchy contains a multiple occurring parent element with one or more multiple occurring child elements, the Developer tool does not map the first level of the hierarchy.

You can choose to display the output ports in a hierarchy. Each child group appears under the parent group. You can also choose to view the lines that connect the elements in the method output to the output ports.

If the associated schema object is deleted from the repository, the Developer tool retains the location of the method elements in the output mapping. When you show the output mapping, the **Ports** area still displays the location of the method elements in the **Location** column for the output ports. If you associate another schema with the transformation, the Developer tool checks whether each location is valid. The Developer tool

clears the location of method elements in the **Ports** area of the output mapping if the location is no longer valid.

Rules and Guidelines to Map Elements to Output Ports

Review the following rules when you map elements in the method output hierarchy to output ports:

- The method output element and the output port must have compatible datatypes.
- You cannot map an element to more than one output port in a group.
- Each output port must have a valid location, unless the port is a pass-through port.
- If you drag a multiple-occurring child element to an empty output port, you must relate the group to other output groups. When you select a group, the Developer tool creates keys to relate the groups.
- When you drag a multiple-occurring element into a group that contains the parent element, you can configure the number of child element occurrences to include. Or, you can replace the parent group with the multiple-occurring child group in the transformation output.
- If the web service produces a JSON document, ensure that `xmlRoot` is the first node in the response hierarchy. If `xmlRoot` is not the first node for a web service with JSON response, null values may appear in the output ports.

Customize View Options

You can change the method output hierarchy to show the cookie ports, pass-through ports, and keys in the **Method Output** area. You can also show grouping constructs that define how to order elements.

Click **Customize View** in the **Method Output** area. Enable any of the following options:

Sequence, Choice, and All

Show a line that indicates whether an element definition is sequence, choice, or all.

Elements in a sequence group must be in the order specified in the hierarchy.

At least one element in a choice group must appear in the response message.

Elements in an all group must all be included in the response message.

Keys

View the keys in the **Method Output** area. The **Method Output** area includes keys for each group. You can add a key to an output port in the **Ports** area.

Pass-through Ports

The **Method Output** area shows the pass-through ports. Pass-through ports are ports that pass data through the transformation without changing the data. You can project pass-through ports from the method output to any of the REST Web Service Consumer transformation output groups. A pass-through port receives data one time, so the port is at the root level in the response messages.

Mapping the Method Output to Output Ports

When you show the transformation output mapping, you can define output groups, define output ports, and map method output elements to output ports.

1. Open a REST Web Service Consumer transformation.
2. In the **Ports** view, show the output mapping.

3. To add an output group or port to the **Ports** area, use one of the following methods:

Method	Description
Drag an element.	Drag a group or a child element in the Method Output area to an empty column in the Ports area. If you drag a group to the Ports area, the Developer tool adds a group without ports.
Manually add a group or port.	To add a group, click the arrow next to the New button and then click Group . To add a port, click the arrow next to the New button, and then click Field .
Drag a port from another transformation.	In the editor, drag a port from another transformation to the REST Web Service Consumer transformation.
Copy a port.	Select ports from another transformation and copy them to the Ports area. To copy ports, you can use keyboard shortcuts or you can use the Copy and Paste buttons in the Developer tool.

4. If you manually create a port or you copy a port from another transformation, click the **Location** column in the **Ports** area, and choose an element from the list.
5. To clear the locations of ports, use one of the following methods:

Method	Description
Click Clear .	Select one or more ports in the Ports area, and click Clear .
Delete the lines that connect elements to ports.	Select one or more lines that connect the elements in the method output to the output ports, and press Delete .

REST Web Service Consumer Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the REST Web Service Consumer transformation.

Configure the following properties on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Connection

Identifies the HTTP connection object to connect to the web service. Create and edit the HTTP connection in the Developer tool. When you configure an HTTP connection, configure the base URL, the type of security that the web service requires, and a connection timeout period.

The REST Web Service Consumer transformation connects to a web service using a URL. You can define the URL in the transformation properties or in the HTTP connection.

Configure an HTTP connection in the following circumstances:

- You do not use a URL input port.

- The web service requires HTTP authentication or SSL certificates.
- You want to change the default connection timeout period.

XML Schema Validation

Validates the response message at run time. **Select Error on Invalid XML** or **No Validation**.

Sorted Input

Enables the Data Integration Service to generate output without processing all of the input data. Enable sorted input when the input data is sorted by the keys in the XML input hierarchy.

URL

The base URL for the REST web service. The base URL in the HTTP connection overrides this value.

Format

The format of the web service response. Select **XML** or **JSON** depending on the web service response..

REST Web Service Consumer Transformation Creation

You can create a reusable or non-reusable REST Web Service Consumer transformation. Reusable transformations can exist in multiple mappings. Non-reusable transformations exist within a single mapping.

When you create a REST Web Service Consumer transformation, you can define the elements and XML hierarchy manually or you can import the elements and hierarchy from a schema object. The schema object can be an XML file or a text file.

Creating a REST Web Service Consumer Transformation

When you create a REST Web Service Consumer transformation, you select a method and define the method input and the method output based on the method that you choose.

1. To create a REST Web Service Consumer transformation, use one of the following methods:

Method	Description
Reusable	Select a project or folder in the Object Explorer view. Click File > New > Transformation . Select the REST Web Service Consumer transformation and click Next .
Non-reusable	In a mapping or mapplet, drag a REST Web Service Consumer transformation from the transformation palette to the mapping or mapplet editor.

2. Enter the transformation name, and select the location and the HTTP method.
3. Click **Next**.
4. To define the method input, use one of the following methods:

Method	Description
Create as empty	Define the XML elements and hierarchy manually.

Method	Description
Create from an element in a Schema Object	Import the XML elements and hierarchy from a schema object.

The **Method input definition** area shows the transformation input groups and input ports. The **Input mapping** area shows the request message hierarchy.

5. Define the input groups and input ports, and map the input ports to input elements.
6. Click **Next**.
7. To define the method output, select **Create as empty** or **Create from an element in a Schema Object**.

The **Method output definition** area shows the transformation input groups and input ports. The **Output mapping** area shows the request message hierarchy.

8. Define the output groups and output ports, and map the elements to output ports.
9. Click **Finish**.

Parsing a JSON Response Message that Contains Arrays

When the element is a child element of complex type and the maximum occurrence of this element is unbounded, the schema is not valid. The JSON parser restricts you from extracting multiple instances of an element.

The maximum occurrence of child elements under complex type must be 0 or 1 with the order indicator as choice for the complex type in a schema. When you change the maximum occurrence as 1 to validate the schema, you can extract one instance of the element at a time.

You can use the maximum occurrence as unbounded in the choice order indicator of a complex type in the schema.

Example JSON Response Message

You have the following schema where the complex type element `xmlRoot` has element name `Likes` whose maximum occurrence is unbounded:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="xmlRoot">
    <xs:complexType>
      <xs:all>
        <xs:element type="xs:byte" name="Age"/>
        <xs:element type="xs:string" name="FirstName"/>
        <xs:element type="xs:string" name="Likes" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element type="xs:string" name="FamilyName"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

You can change the JSON response in the following format:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="xmlRoot">
    <xs:complexType>
```

```

<xs:choice maxOccurs="unbounded">
  <xs:element type="xs:byte" name="Age"/>
  <xs:element type="xs:string" name="FirstName"/>
  <xs:element type="xs:string" name="Likes" />
  <xs:element type="xs:string" name="FamilyName"/>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>

```

<xs:choice maxOccurs="unbounded"> allows the contents to be repeated one or more times, in any order.

Unnamed Arrays in a Response Message

A REST Web Service Consumer transformation supports unnamed arrays only in a response message but not in a request message. To parse an unnamed array schema defined in the Method Output Definition, the parent element of complexType or simple type array elements must be of name `xmlRoot`.

In a Rest Web Service Consumer transformation, you must define `xmlRoot` as the child element of the `xmlRoot` element with maximum occurs set to unbounded and the elements in unnamed array as child elements of the `xmlRoot` element.

The following image shows the defined method output for the unnamed array:

Ports Method input Method output

Show: Method output definition Output mapping

Method output definition

Name	Type	Min...	Ma...	Description	>>
Rest_Consume...	(Rest_Cons...				
xmlRoot	(xmlRoot)	1	1		
xmlRoot	(xmlRoot)	1	Un...		
emp	xs:string	1	1		
empid	xs:string	1	1		

CHAPTER 40

Router Transformation

This chapter includes the following topics:

- [Router Transformation Overview, 574](#)
- [Router Transformations in Dynamic Mappings, 575](#)
- [Working with Groups, 575](#)
- [Working with Ports, 579](#)
- [Connecting Router Transformations in a Mapping, 580](#)
- [Router Transformation Advanced Properties, 580](#)
- [Router Transformation in a Non-native Environment, 580](#)

Router Transformation Overview

The Router transformation is an active transformation that routes data into multiple output groups based on one or more conditions. Route the output groups to different transformations or to different targets in the mapping.

A Router transformation is similar to a Filter transformation because both transformations 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. A Router transformation tests data for one or more conditions and can 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 can use one Router transformation instead of three Filter transformations. When you use a Router transformation in a mapping, the Data Integration Service processes the incoming data once. When you use multiple Filter transformations in a mapping, the Data Integration Service processes the incoming data for each transformation.

A Router transformation consists of input and output groups, input and output ports, group filter conditions, and advanced properties that you configure in the Developer tool.

When the Spark engine runs a mapping with a Router transformation, the Spark engine processes the upstream mapping pipeline once and stages the data on the HDFS so it can be used by each downstream branch.

The following figure shows a sample Router transformation and its components:

Name	Type	Lengthy...
Input (4)		
COUNTRY	string	13
CUSTOMER_NO	decimal	10
FIRSTNAME	string	10
LASTNAME	string	10
Default (4)		
COUNTRY	string	13
CUSTOMER_NO	decimal	10
FIRSTNAME	string	10
LASTNAME	string	10
France (4)		
COUNTRY	string	13
CUSTOMER_NO	decimal	10
FIRSTNAME	string	10
LASTNAME	string	10
Japan (4)		
COUNTRY	string	13
CUSTOMER_NO	decimal	10
FIRSTNAME	string	10
LASTNAME	string	10
USA (4)		
COUNTRY	string	13
CUSTOMER_NO	decimal	10
FIRSTNAME	string	10
LASTNAME	string	10

1. Input group
2. Input ports
3. Default output group
4. User-defined output groups

Router Transformations in Dynamic Mappings

You can use an Router transformation in a dynamic mapping. You can configure dynamic ports in the transformation and reference the generated ports in the group filter condition.

When you use a dynamic port in the group filter condition, the dynamic port might contain more than one generated port. The group filter condition expands to include each generated port. Each generated port must be a valid type for the expression.

You can parameterize the group filter condition. Use an expression type parameter to specify the filter.

Working with Groups

A Router transformation has the following types of groups:

- Input
- Output

Input Group

The Router transformation includes a single input group. The input group includes all input ports that you add to the transformation.

Output Groups

The Rank transformation includes the following types of output groups:

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 Developer tool. Create one user-defined group for each condition that you want to specify.

The Data 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 Data Integration Service determines the order of evaluation for each condition based on the order of the connected output groups. The Data Integration Service processes user-defined groups that are connected to a transformation or a target in a mapping.

If a row meets more than one group filter condition, the Data Integration Service passes this row multiple times.

The Default Group

The Developer tool creates the default group after you create one user-defined group. The Developer tool 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 group conditions evaluate to FALSE, the Data Integration Service passes the row to the default group. If you want the Data Integration Service to drop all rows in the default group, do not connect it to a transformation or a target in a mapping.

The Developer tool deletes the default group when you delete the last user-defined group from the list.

The Developer tool copies property information from the input ports of the input group to create a set of output ports for each output group. You cannot change or delete output ports or their properties.

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, based on whether a row satisfies the specified condition. Zero (0) is the equivalent of FALSE. Any non-zero value is the equivalent of TRUE. You can use a single numeric port as a filter condition. The Data Integration Service passes the rows of data that evaluate to TRUE to each transformation or target that is associated with each user-defined group.

Note: You cannot use a single dynamic port to return a boolean value.

For example, you have customers from nine countries, and you want to perform different calculations on the data from three countries. You can use a Router transformation in a mapping to filter this data to three different Expression transformations.

You can use parameters as elements in the group filter condition. You can use system parameters or user-defined parameters. You can create parameters from the Expression Editor and add them to the expression.

In the mapping, the Data Integration Service passes the rows of data that evaluate to TRUE to each transformation or target associated with each user-defined group, such as Japan, France, and USA. The Data Integration Service passes the row to the default group if all of the conditions evaluate to FALSE. The Data Integration Service then passes the data of the other six countries to the transformation or target that is associated with the default group. If you want the Data Integration Service to drop all rows in the default group, do not connect it to a transformation or a target in a mapping.

The Router transformation passes data through each group that meets the condition. If data meets three output group conditions, the Router transformation passes the data through three output groups.

For example, you configure the following group conditions in a Router transformation:

Group Name	Group Filter Condition
Output Group 1	employee_salary > 1000
Output Group 2	employee_salary > 2000

When the Router transformation processes an input row data with employee_salary=3000, it routes the data through output groups 1 and 2.

Dynamic Ports in Group Filter Conditions

You can use a dynamic port in a group filter condition. The Data Integration Service applies the filter condition to each generated port in the dynamic port.

For example, a dynamic port, MyDynamicPort, contains three decimal ports:

```
Salary  
Bonus  
Stock
```

You might configure the following group filter condition:

```
MyDynamicPort > 100
```

The group filter condition expands to the following expression:

```
Salary > 100 AND Bonus > 100 AND Stock > 100
```

Each generated port must be a valid type for the expression.

Parameterize the Group Filter

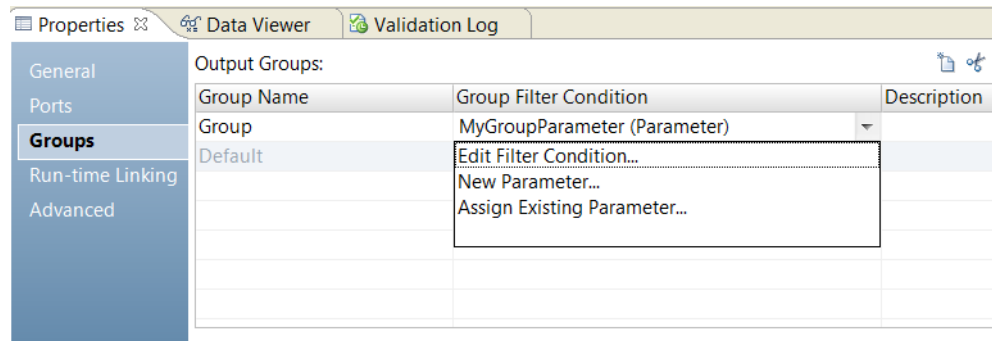
You can use an expression parameter to define a group filter. An expression parameter is a parameter that contains a complete expression.

For example, an expression parameter might contain the expression

```
Employee_Salary > 1000.
```

To parameterize the group filter, choose **New Parameter** in the **Group Filter Condition** field of the **Groups** tab. Define the parameter and enter the expression on the Expression Editor. An expression parameter cannot contain other parameters.

The following image shows the **Groups** tab in the transformation properties:



When you use an expression parameter, the Developer tool cannot validate the expression. If the expression is not valid at run time, the mapping might fail.

Adding Groups

When you add a group, the Developer tool copies property information from the input ports to the output ports.

1. Click the **Groups** tab.
2. Click the **New** button.
3. Enter a name for the group in the **Group Name** section.
4. Click the **Group Filter Condition** field to 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.

The Developer tool 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 Developer tool updates the output ports to reflect these changes. You cannot edit or delete output ports.

The Developer tool creates output port names based on the input port names. For each input port, the Developer tool creates a corresponding output port in each output group.

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.

Router Transformation Advanced Properties

Configure advanced properties to help determine how the Data Integration Service processes data for the Router transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Router Transformation in a Non-native Environment

The Router transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 41

Sequence Generator Transformation

This chapter includes the following topics:

- [Sequence Generator Transformation Overview, 581](#)
- [Sequence Generator Ports, 581](#)
- [Sequence Generator Transformation Properties, 584](#)
- [Sequence Generator Advanced Properties, 585](#)
- [Sequence Data Object, 586](#)
- [Creating a Sequence Generator Transformation, 589](#)
- [Frequently Asked Questions, 590](#)
- [Sequence Generator Transformation in a Non-native Environment, 591](#)

Sequence Generator Transformation Overview

The Sequence Generator transformation is a passive transformation that generates numeric values. Use the Sequence Generator transformation to create unique primary key values, replace missing primary keys, or cycle through a sequential range of numbers.

The Sequence Generator contains pass-through ports and an output port. You connect the NEXTVAL port to the input ports of other transformations. The Integration Service increments the sequence when the mapping runs.

You can create a Sequence Generator transformation based on a new sequence or a sequence data object. A sequence data object is an object that creates and maintains a sequence of values.

Sequence Generator Ports

The Sequence Generator transformation has pass-through ports and one output port, NEXTVAL. You cannot edit or delete the output port.

Pass-Through Ports

You can add a port to the Sequence Generator transformation as a pass-through port. Pass-through ports are input and output ports that receive input data and return the same data to a mapping without changing it.

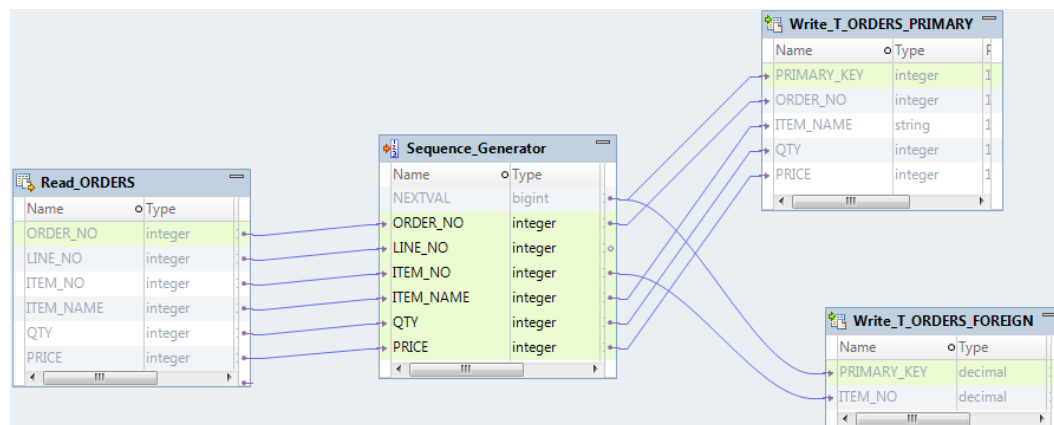
You must add at least one input port to the transformation and connect it to an upstream source or transformation before you link the NEXTVAL output port to targets. To add a pass-through port to the transformation, drag a port from an upstream source or transformation in the mapping to the Sequence Generator transformation.

NEXTVAL Port

You can connect NEXTVAL to a transformation to generate unique values for each row in the transformation. Connect the NEXTVAL port to a downstream transformation or target to generate a sequence of numbers. If you connect NEXTVAL to multiple transformations, the Integration Service generates the same sequence of numbers for each transformation.

You connect the NEXTVAL port to generate the sequence based on the Start Value and Increment Value properties. If the Sequence Generator is not configured to cycle through the sequence, the NEXTVAL port generates sequence numbers up to the configured end value.

The following image shows a mapping with the Sequence Generator transformation NEXTVAL port connected to a source and two targets to generate primary and foreign key values:



When you configure the Sequence Generator transformation with a Start Value = 1 and an Increment Value = 1, the Integration Service generates the same primary key values for the T_ORDERS_PRIMARY and T_ORDERS_FOREIGN target tables.

Create Keys

You can create primary or foreign key values with the Sequence Generator transformation by connecting the NEXTVAL port to a target or downstream transformation. You can use a range of values from 1 to 9,223,372,036,854,775,807 with the smallest interval of 1. You can also create composite keys to identify each row in the table.

To create a composite key, you can configure the Integration Service to cycle through a smaller set of values. For example, if three stores generate order numbers, configure a Sequence Generator transformation to cycle through values from 1 to 3, incrementing by 1. When you connect the ORDER_NO port to the Sequence Generator transformation, the generated values create unique composite keys.

The following example shows composite keys and order numbers:

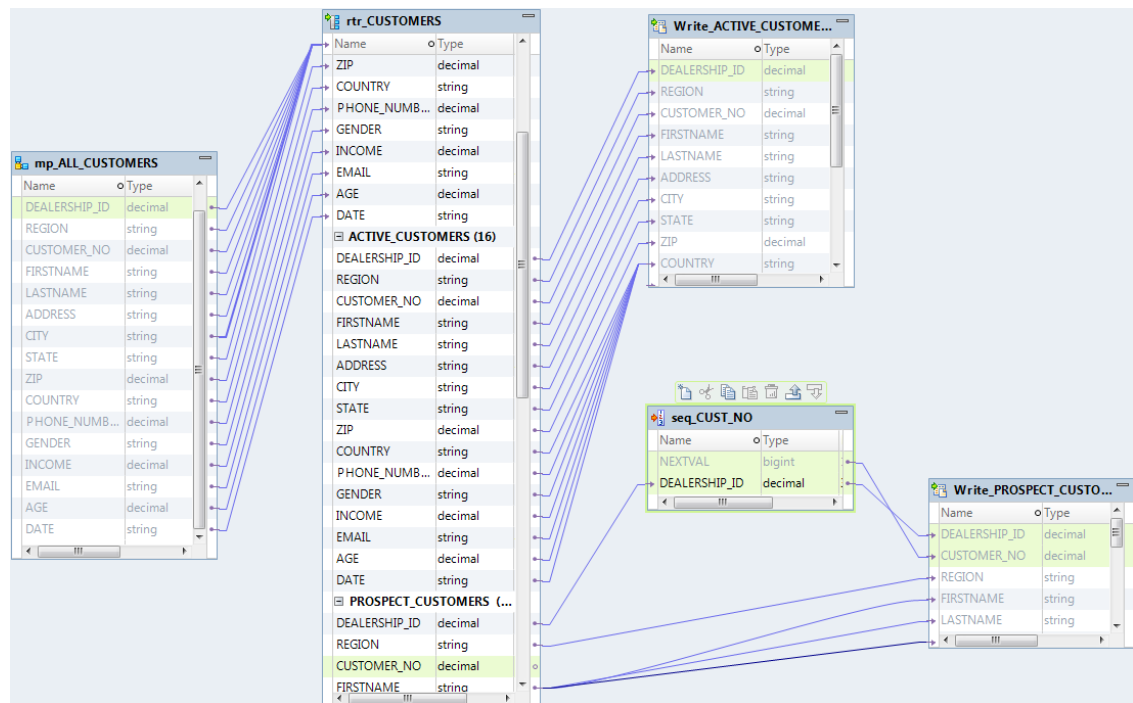
COMPOSITE_KEY	ORDER_NO
1	12345
2	12345
3	12345
1	12346
2	12346
3	12346

Replace Missing Values

When you use a Sequence Generator transformation to replace missing keys, you can also use a Router transformation to filter null values from columns that have values assigned. You connect the Router transformation to the Sequence Generator transformation, and use NEXTVAL to generate a sequence of numeric values to populate the null values.

For example, to replace null values in a CUSTOMER_NO column, you create a mapping with a source that contains customer data. You add a Router transformation to filter customers with customer numbers assigned from those with null values. You add a Sequence Generator transformation to generate unique CUSTOMER_NO values. You add customer targets to write the data to.

The following image shows a mapping that replaces null values in the CUSTOMER_NO column:



Sequence Generator Transformation Properties

Configure transformation properties that the Integration Service uses to generate sequential values.

The following table lists the properties that you configure for a sequence data object and a new sequence:

Property	Description
Start Value	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. Maximum value is 9,223,372,036,854,775,806.
End Value	Maximum value that 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. Maximum value is 9,223,372,036,854,775,807.
Increment Value	Difference between two consecutive values from the NEXTVAL port. Default is 1. Must be a positive integer. Maximum value is 2,147,483,647.
Cycle	If enabled, the Integration Service cycles through the sequence range and starts over with the start value. If disabled, the Integration Service stops the sequence at the configured end value. The Integration Service fails the session with overflow errors if it reaches the end value and still has rows to process.

Start Value

Use Cycle to generate a repeating sequence, such as numbers 1 through 12 to correspond to the months in a year.

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.

End Value

The end value is the maximum value that 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 mapping run fails with an overflow error.

Set the end value to any integer between 1 and 9,223,372,036,854,775,807. If you connect the NEXTVAL port to a downstream integer port, set the end value to a value no larger than the integer maximum value. For example, if you connect the NEXTVAL port to a Small Integer port, set the end value to a maximum of 32,767. If the NEXTVAL exceeds the datatype maximum value for the downstream port, the mapping fails.

Increment Value

The Integration Service generates a sequence in the NEXTVAL port 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
```

Cycle Through a Range of Values

You can establish a range of values for the Sequence Generator transformation. If you use the cycle option, the Sequence Generator transformation repeats the range when it reaches the end value.

For example, if you set the sequence range to start at 10 and end at 50, and you set an increment value of 10, the Sequence Generator transformation creates the values 10, 20, 30, 40, 50. The sequence starts over again at 10.

Sequence Generator Advanced Properties

Configure advanced properties that the Integration Service uses to generate sequential values.

The following table lists the advanced properties that you can configure for a sequence data object and a new sequence:

Property	Description
Reset	If enabled, the Integration service resets the sequence data object to the start value when the mapping run completes. If disabled, the Integration Service increments the current value after the mapping run completes, and uses that value in the next mapping run. This property is disabled for reusable Sequence Generator transformations and for non-reusable Sequence Generator transformations that use a reusable sequence data object.
Tracing Level	Level of detail about the transformation that the Integration Service writes into the mapping log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Maintain Row Order	Maintain the row order of the input data to the transformation. Select this option if the Integration Service should not perform any optimization that can change the row order. Default is false.

Reset

If you configure a non-reusable Sequence Generator transformation to use the reset property, the Integration Service uses the original start value for each mapping run. Otherwise, the Integration Service increments the current value and uses that value in the next mapping run.

For example, you configure a Sequence Generator transformation to create values from 1 to 1,000 with an increment of 1, a start value of 1, and choose reset. During the first mapping run, the Integration Service generates numbers 1 through 234. In each subsequent mapping run, the Integration Service again generates numbers beginning with the start value of 1.

If you do not reset, the Integration Service updates the current value to 235 at the end of the first 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.

Maintain Row Order

Maintain the row order of the input data to the transformation. Select this option if the Integration Service should not perform any optimization that can change the row order.

When the Integration Service performs optimizations, it might lose an order established earlier in the mapping. You can establish order in a mapping with a sorted flat file source, a sorted relational source, or a Sorter transformation. When you configure a transformation to maintain row order, the Integration Service considers this configuration when it performs optimizations for the mapping. The Integration Service performs optimizations for the transformation if it can maintain the order. The Integration Service does not perform optimizations for the transformation if the optimization would change the row order.

Sequence Data Object

A sequence data object creates and maintains a sequence of numeric values. The Sequence Generator transformation uses the sequence data object to generate the values for the transformation.

You can use a reusable sequence data object in multiple Sequence Generator transformations. All Sequence Generator transformations that use the same sequence data object use the same sequence of values if they run on the same Integration Service. You can also use a reusable sequence data object in a non-reusable Sequence Generator transformation. You can use a non-reusable sequence data object a non-reusable Sequence Generator transformation.

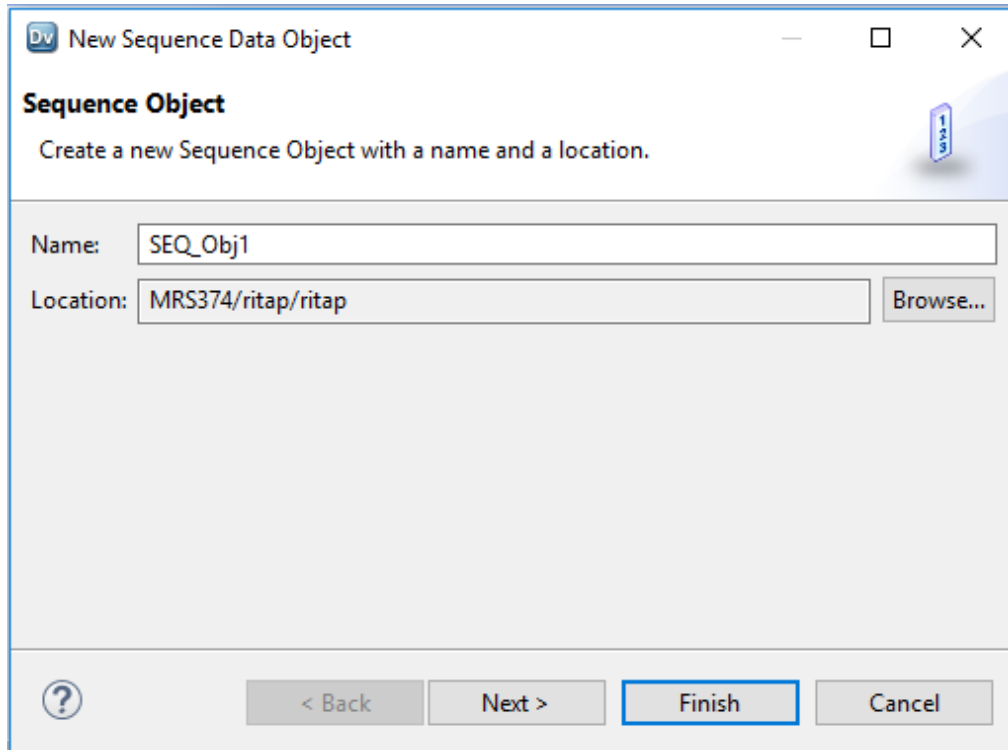
For example, you create multiple mappings that write to the same primary key field in a relational table. Each mapping uses the same reusable Sequence Generator transformation, which uses the same reusable sequence data object and runs on the same Integration Service. Each mapping write unique values to the primary key field.

Creating a Sequence Data Object

To use a Sequence data object to create a Sequence Generator transformation, create the Sequence data object, configure the object properties, and select the object in the Sequence Generator transformation dialog box.

1. In the mapping editor, scroll down in the mapping palette to locate the Sequence Generator transformation and drag it to the mapping.
The **New Transformations** wizard opens.
2. Click **New Sequence Data Object**.

The **New Data Object** wizard opens.



3. Enter a name for the Sequence data object.
The naming convention for Sequence data objects is SEQ_<data object name>.
4. Click **Next** to configure the Sequence data object properties.

If you create a Sequence Generator transformation from the object, the transformation will use the properties you enter for the data object. The following image shows the properties that you can configure:

Dv New Sequence Data Object

Sequence Object
Enter the Sequence

Start Value:

End Value:

Increment Value:

Cycle:

5. After you configure the data object properties, you can create a Sequence Generator transformation using the Sequence data object. When you create the transformation, name the Sequence Generator transformation, and select **Choose an existing Sequence object**. Navigate to the data object and click **OK**.

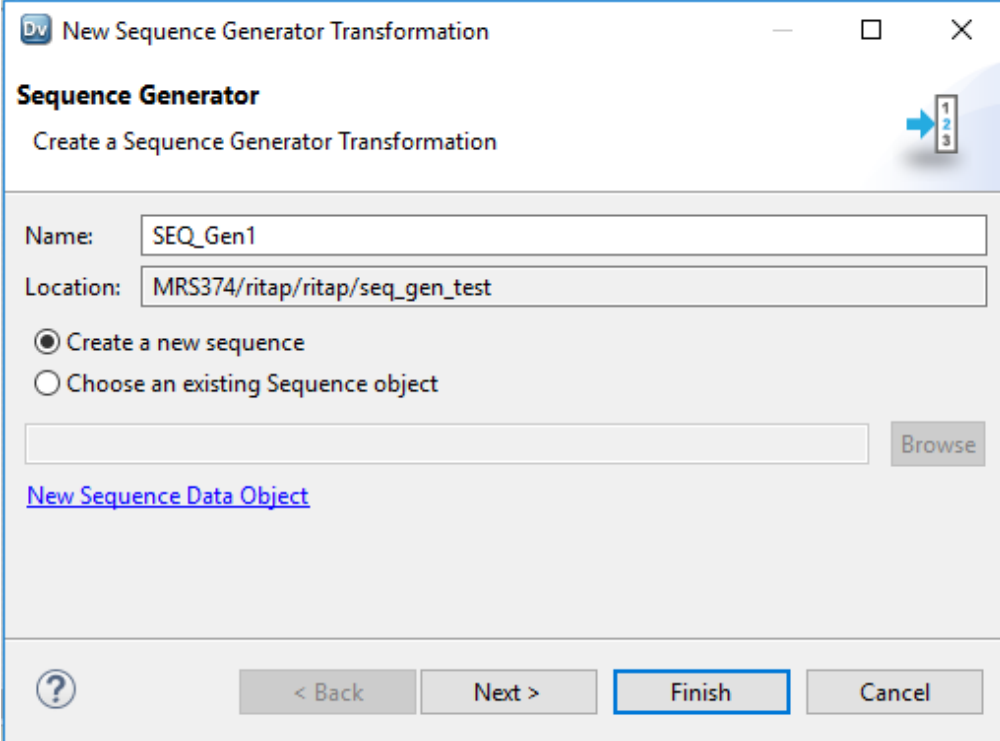
The Sequence Generator transformation appears in the mapping editor with a NEXTVAL output-only port. You can connect the NEXTVAL port to a downstream transformation or target to generate a sequence of numbers.

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 to one or more transformations.

1. In the mapping editor, scroll down in the mapping palette to locate the Sequence Generator transformation and drag it to the mapping.

The **New Transformations** wizard opens.



The screenshot shows a wizard dialog box titled "New Sequence Generator Transformation". The main heading is "Sequence Generator" with the subtitle "Create a Sequence Generator Transformation". There is a blue arrow icon pointing right with the numbers 1, 2, and 3 stacked vertically next to it. Below the heading, there are two text input fields: "Name:" with the value "SEQ_Gen1" and "Location:" with the value "MRS374/ritap/ritap/seq_gen_test". There are two radio button options: "Create a new sequence" (which is selected) and "Choose an existing Sequence object". Below these options is a text input field and a "Browse" button. At the bottom of the dialog, there is a question mark icon, a "< Back" button, a "Next >" button, a "Finish" button (which is highlighted with a blue border), and a "Cancel" button. A blue link labeled "New Sequence Data Object" is located below the "Browse" button.

2. Enter a name for the Sequence Generator transformation.
The naming convention for Sequence Generator transformations is SEQ_<transformation name>.
3. Choose to create a new sequence or to use an existing Sequence object.

- To create a new sequence, select **Create a new sequence**. Click **Next** to configure the sequence properties. The following image shows the properties that you can configure:

- To use an existing Sequence object, select **Choose an existing Sequence object**. Navigate to the Sequence object and click **OK**.

The Sequence Generator transformation appears in the mapping editor with a NEXTVAL output-only port. You can connect the NEXTVAL port to a downstream transformation or target to generate a sequence of numbers.

Frequently Asked Questions

Can I change a non-reusable Sequence Generator transformation to make it reusable?

You cannot make the transformation reusable, but you can change the transformation to use a sequence data object. The sequence data object maintains the integrity of the sequence, regardless of the number of transformations that use it.

Can I place a non-reusable Sequence Generator transformation in a mapplet?

No, you cannot. Mapplets are reusable objects, so all objects in the mapplet must also be reusable. Use a reusable Sequence Generator transformation instead.

Sequence Generator Transformation in a Non-native Environment

The Sequence Generator transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported when the domain and the Databricks cluster reside in the same virtual network in AWS or Azure Databricks environments.

Sequence Generator Transformation on the Blaze Engine

A mapping with a Sequence Generator transformation consumes significant resources when the following conditions are true:

- You configure the transformation to maintain row order.
- The mapping runs in a single partition.

Sequence Generator Transformation on the Spark Engine

The Sequence Generator transformation does not maintain row order in output data. If you enable the **Maintain Row Order** property on the transformation, the Data Integration Service ignores the property.

Note: Spark sequence generator with maintain row order works fine if the data set is relatively small and can run or forced to run on a single partition.

CHAPTER 42

Sorter Transformation

This chapter includes the following topics:

- [Sorter Transformation Overview, 592](#)
- [Sorter Transformations in Dynamic Mappings, 593](#)
- [Developing a Sorter Transformation, 593](#)
- [Sorter Transformation Ports, 593](#)
- [Sort Tab, 594](#)
- [Configure Sort Keys, 594](#)
- [Sorter Transformation Advanced Properties, 596](#)
- [Sorter Cache, 597](#)
- [Creating a Sorter Transformation, 597](#)
- [Sorter Transformation Example, 598](#)
- [Sorter Transformation in a Non-native Environment, 600](#)

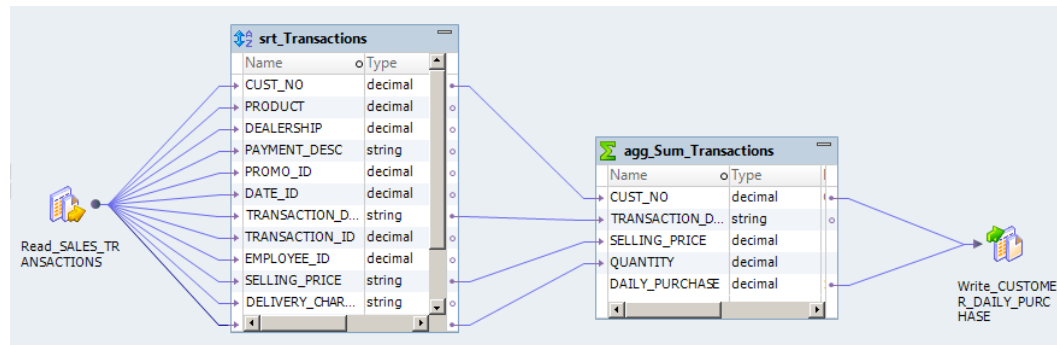
Sorter Transformation Overview

Use a Sorter transformation to sort data in ascending or descending order according to a specified sort key. You can configure the Sorter transformation for case-sensitive sorting and for distinct output. The Sorter transformation is an active transformation.

When you create a Sorter transformation, you specify ports as sort keys and configure each sort key port to sort in ascending or descending order. The Data Integration Service sorts each port sequentially when you specify multiple ports for sort key.

For example, you need to create an invoice for the overall customer sales from a customer database. Use a Sorter transformation on the customer sales table to sort the data in descending order according to the customer number. Use the result of the Sorter transformation as an input to the Aggregator transformation. You can increase Aggregator transformation performance with the sorted input option.

The following figure shows the mapping:



Sorter Transformations in Dynamic Mappings

You can use a Sorter transformation in a dynamic mapping. You can configure dynamic ports in the transformation and reference the generated ports.

You can reference a dynamic port or a generated port in the Sorter transformation. However, if the generated port does not exist at run time, the mapping fails.

If you use a dynamic port as a sort key, the Data Integration service considers all the generated ports in the dynamic port and the generated port order.

You can parameterize the sort keys. Use a sort list parameter for the sort key.

Developing a Sorter Transformation

When you develop a Sorter transformation, you need to consider factors such as the sort key ports, distinct output rows, and the case-sensitive sort criteria.

Consider the following factors when you develop a Sorter transformation:

- The ports that you want to configure as sort keys and the sort direction.
- Whether you want a case-sensitive sort.
- Whether you want to consider null values as sort priority.
- Whether you want distinct output rows.
- The sorter cache size value you want to set.

Sorter Transformation Ports

When you create ports in the Sorter transformation, you create input and output ports by default. The Sorter transformation returns the same output ports as input ports.

You can define dynamic ports in a Sorter transformation. A dynamic port can receive different columns of data from an upstream transformation in a mapping. This enables the Sorter transformation to sort rows that contain different columns.

Define the sort keys on the **Sort** tab of the **Properties** view.

Sort Tab

Define the sort key on the **Sort** tab of the Sorter transformation **Properties** view. Choose one or more ports that you want to use as the sort criteria.

The Data Integration Service sorts data according to the order of the ports on the Sorter tab. Configure to sort data in ascending or descending order. The default is ascending.

If you configure the Sorter transformation for distinct output rows, the Developer tool configures all ports as part of the sort key. The Data Integration Service discards duplicate rows during the sort operation

Configure Sort Keys

Define the sort key on the **Sort** tab of the transformation **Properties** view.

This is the start of the concept.

The following image shows the **Sort** tab:

Sort

Output: All rows Distinct rows only

Sort Keys

Specify by: Value

Ports:

Department	Ascending (A)
Employee	Ascending (A)

Buttons: Add, Choose..., Delete, Move Up, Move Down

The **Sort** tab contains the following options:

Output

Choose whether to return all the sorted rows or to discard duplicate rows. Duplicate rows are rows where all the column values are the same.

Specify by

Select **Value** or **Parameter**. Select **Value** to use port names. Choose **Parameter** to use a sort list parameter.

Add

Accepts a port name that you type in manually. You must type a valid port name before you click **Add**.

Choose

Click **Choose** to select ports to add to the sort key. The Developer tool provides a list of ports from the transformation to choose from.

Move Up and Move Down

You can change the order of the ports in the group. Select the port name and then click one of the move buttons to move it up or down in the sort order.

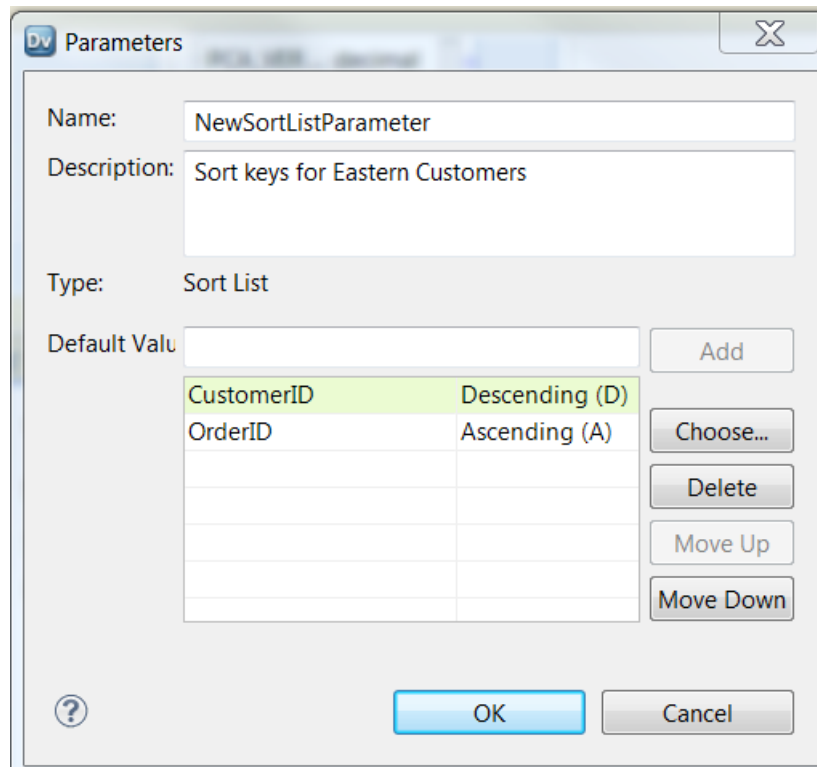
Parameterize the Sort Keys

You can create a sort list parameter that contains a list of ports for the sort keys.

If the Sort transformation is in a dynamic mapping, the Sorter transformation might contain generated ports. You can parameterize the sort keys. Create a sort list parameter that contains a list of ports to sort by.

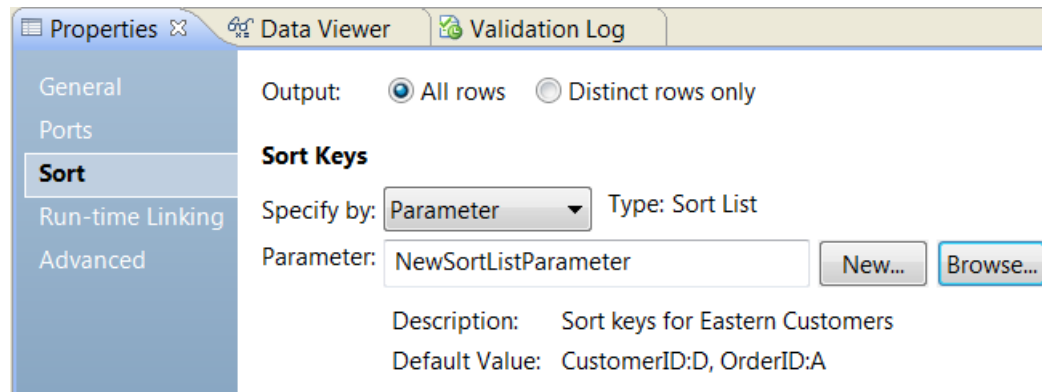
On the **Sort** tab of the transformation properties, choose **Specify by Parameter**. Click **New** to create a parameter.

The following image shows the **Parameters** dialog box:



Choose ports or generated ports for the sort keys. You can choose an ascending or descending sort type. You can manually enter port names. Type the port name in the **Default Value** field and click **Add**. The Developer tool adds the port name to the sort list.

The following image shows the **Sort** tab after you configure a parameter for the sort keys:



Sorter Transformation Advanced Properties

You can specify additional sort criteria in the Sorter transformation advanced properties. The Data Integration Service applies the properties to all sort key ports. The Sorter transformation properties also determine the system resources that the Data Integration Service allocates when it sorts data.

The following section describes the advanced properties for a Sorter transformation:

Case Sensitive

Determines whether the Data Integration Service considers case when sorting data. When you enable the Case Sensitive property, the Data Integration Service sorts uppercase characters higher than lowercase characters. The Developer tool sets the Case Sensitive by default.

Null Treated Low

Treats a null value as lower than any other value. Enable the property if you want the Data Integration Service to treat a null value as lower than any other value when it performs the sort operation.

Sorter Cache Size

Amount of memory that the Data Integration Service allocates at the start of the mapping run to perform the sort operation. The Data Integration Service passes all incoming data into the Sorter transformation before it performs the sort operation. Select Auto to have the Data Integration Service automatically calculate the memory requirements at run time. Enter a specific value in bytes when you tune the cache size. Default is Auto.

Work Directory

Directory where the Data Integration Service temporarily stores data if the amount of incoming data is greater than the Sorter cache size. After the Data Integration Service sorts data, it deletes the temporary files.

Enter multiple directories separated by semicolons to increase performance during cache partitioning. Cache partitioning creates a separate cache for each partition that processes the transformation.

Default is the TempDir system parameter. You can configure another system parameter or user-defined parameter in this field.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

RELATED TOPICS:

- [“Cache Size” on page 71](#)

Sorter Cache

The Data Integration Service creates a cache in memory to run the Sorter transformation. The Data Integration Service passes all incoming data into the Sorter transformation before it performs the sort operation. If the Data Integration Service requires more space than available in the memory cache, it temporarily stores data in the Sorter transformation work directory.

If you do not configure the cache size to sort all of the data in memory, a warning appears in the session log stating that the Data Integration Service made multiple passes on the source data. The Data Integration Service makes multiple passes on the data when it has to page information to disk to complete the sort. The message specifies the amount of memory required for a single pass, which is when the Data Integration Service reads the data once and performs the sort in memory without paging to disk. To optimize mapping performance, configure the cache size so that the Data Integration Service makes one pass on the data.

If the amount of incoming data is greater than the Sorter cache size, the Data Integration Service temporarily stores data in the Sorter transformation work directory. The Data Integration Service requires disk space of at least twice the amount of incoming data when storing data in the work directory.

For best performance, configure the Sorter cache size with a value less than or equal to the amount of available physical memory on the machine that runs the mapping. To sort data using a Sorter transformation, allocate at least 16 MB (16,777,216 bytes) of physical memory. Sorter cache size is set to Auto by default.

Optimizing the Sorter Cache

The sorter cache is optimized to use variable length to store binary and string data types that pass through the Sorter transformation.

Variable length reduces the amount of data that the Data Integration Service stores in the sorter cache and the disk space consumption on the Data Integration Service machine.

For example, you store data on customers. Some customers have longer names than other customers. If the Data Integration Service uses fixed length to store the data on customer names, the Data Integration Service might store data on 20 characters for each name. If the Data Integration Service uses variable length, the Data Integration Service might store data with an average length of 10 characters.

Creating a Sorter Transformation

You can create reusable or non-reusable sorter transformations.

Creating a Reusable Sorter Transformation

Create a reusable Sorter transformation to use in multiple mappings or mapplets.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
The **New** dialog box appears.
3. Select the Sorter transformation.
4. Click **Next**.
5. Enter a name for the transformation.
6. Click **Finish**.
The transformation appears in the editor.
7. Click **New** to add a port to the transformation.
8. Edit the port to set the name, datatype, and precision.
9. On the **Sort** tab, choose the ports to sort by or select a sort list parameter.
10. Click the **Advanced** view and edit the transformation properties.

Creating a Non-Reusable Sorter Transformation

Create a non-reusable Sorter transformation in a mapping or mapplet.

1. In a mapping or mapplet, drag a Sorter transformation from the Transformation palette to the editor.
The transformation appears in the editor.
2. In the **Properties** view, edit the transformation name and the description.
3. On the **Ports** tab, click **New** to add ports to the transformation.
4. Edit the ports to set the name, datatype, and precision.
5. Select **Key** to indicate the port as sort key.
6. Click the **Advanced** tab and edit the transformation properties.

Sorter Transformation Example

You have a database table PRODUCT_ORDERS that contains information about all the orders which were placed by the customer.

ORDER_ID	ITEM_ID	ITEM	QUANTITY	PRICE
43	123456	ItemA	3	3.04
41	456789	ItemB	2	12.02
43	000246	ItemC	6	34.55
45	000468	ItemD	5	0.56
41	123456	ItemA	4	3.04
45	123456	ItemA	5	3.04

ORDER_ID	ITEM_ID	ITEM	QUANTITY	PRICE
45	456789	ItemB	3	12.02

Use the Sorter transformation on PRODUCT_ORDERS and specify the ORDER_ID as the sort key with direction as descending.

After sorting the data, the Data Integration Service passes the following rows out of the Sorter transformation:

ORDER_ID	ITEM_ID	ITEM	QUANTITY	PRICE
45	000468	ItemD	5	0.56
45	123456	ItemA	5	3.04
45	456789	ItemB	3	12.02
43	123456	ItemA	3	3.04
43	000246	ItemC	6	34.55
41	456789	ItemB	2	12.02
41	123456	ItemA	4	3.04

You need to find out the total amount and item quantity for each order. You can use the result of the Sorter transformation as an input to an Aggregator transformation. Use sorted input in Aggregator transformation to improve performance.

When you do not use sorted input, the Data Integration Service performs aggregate calculations as it reads. The Data Integration Service stores data for each group until it reads the entire source to ensure that all aggregate calculations are accurate. If you use sorted input and do not presort data correctly, you receive unexpected results.

The Aggregator transformation has the ORDER_ID group by port, with the sorted input option selected. When you pass the data from the Sorter transformation, the Aggregator transformation groups ORDER_ID to calculate the total amount for each order.

ORDER_ID	SUM
45	54.06
43	216.42
41	36.2

Sorter Transformation in a Non-native Environment

The Sorter transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Sorter Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The target is configured to maintain row order and the Sorter transformation is not connected directly to a flat file target.

Parallel Sorting

The Data Integration Service enables parallel sorting with the following restrictions:

- The mapping does not include another transformation between the Sorter transformation and the target.
- The data type of the sort keys does not change between the Sorter transformation and the target.
- Each sort key in the Sorter transformation must be linked to a column in the target.

Global Sort

The Blaze engine can perform global sorts in the following situations:

- The Sorter transformation is connected directly to flat file targets.
- The target is configured to maintain row order.
- The sort key is not a binary data type.

If any of the conditions are not true, the Blaze engine performs a local sort.

Data Cache Optimization

If a Sorter transformation is inserted before an Aggregator or Rank transformation to optimize the Aggregator or Rank data cache, the size of the sorter cache is the same size as the data cache for the Aggregator or Rank transformation. To configure the sorter cache, you must configure the size of the data cache for the Aggregator or Rank transformation.

Sorter Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails when case sensitivity is disabled.

The Data Integration Service logs a warning and ignores the Sorter transformation in the following situations:

- There is a type mismatch in between the target and the Sorter transformation sort keys.
- The transformation contains sort keys that are not connected to the target.
- The Write transformation is not configured to maintain row order.
- The transformation is not directly upstream from the Write transformation.

Null Values

The Data Integration Service treats null values as low even if you configure the transformation to treat null values as high.

Data Cache Optimization

You cannot optimize the sorter cache to store data using variable length.

Sorter Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situation:

- The Sorter transformation in a streaming mapping does not have an upstream Aggregator transformation.

General Guidelines

Consider the following general guidelines:

- The mapping runs in complete output mode if it contains a Sorter transformation.
- To maintain global sort order, ensure that the target is single-partitioned. Source can be single- or multi-partitioned.

Sorter Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails when case sensitivity is disabled.

The Data Integration Service logs a warning and ignores the Sorter transformation in the following situations:

- There is a type mismatch in between the target and the Sorter transformation sort keys.
- The transformation contains sort keys that are not connected to the target.
- The Write transformation is not configured to maintain row order.
- The transformation is not directly upstream from the Write transformation.

Null Values

The Data Integration Service treats null values as low even if you configure the transformation to treat null values as high.

Data Cache Optimization

You cannot optimize the sorter cache to store data using variable length.

Parallel Sorting

The Data Integration Service enables parallel sorting with the following restrictions:

- The mapping does not include another transformation between the Sorter transformation and the target.
- The data type of the sort keys does not change between the Sorter transformation and the target.
- Each sort key in the Sorter transformation must be linked to a column in the target.

CHAPTER 43

SQL Transformation

This chapter includes the following topics:

- [SQL Transformation Overview, 603](#)
- [SQL Transformation Ports, 604](#)
- [SQL Transformation Advanced Properties, 608](#)
- [SQL Transformation Query, 610](#)
- [Input Row to Output Row Cardinality, 612](#)
- [Filter Optimization with the SQL Transformation, 616](#)
- [SQL Transformation Example with an SQL Query, 617](#)
- [Stored Procedures, 621](#)
- [SQL Transformation Connection, 626](#)
- [Manually Creating an SQL Transformation, 627](#)
- [Creating an SQL Transformation from a Stored Procedure, 627](#)

SQL Transformation Overview

The SQL transformation processes SQL queries midstream in a mapping. You can run SQL queries from the SQL transformation or you can configure the SQL transformation to run stored procedures from a database.

You can pass input port values to parameters in the query or stored procedure. The transformation can insert, delete, update, and retrieve rows from a database. You can run SQL DDL statements to create a table or drop a table midstream in a mapping. The SQL transformation is an active transformation. The transformation can return multiple rows for each input row.

You can import a stored procedure from a database to the SQL transformation. When you import the stored procedure, the Developer tool creates the transformation ports that correspond to the parameters in the stored procedure. The Developer tool also creates the stored procedure call for you.

To configure an SQL transformation to run a stored procedure, perform the following tasks:

1. Define the transformation properties including the database type to connect to.
2. Import a stored procedure to define the ports and create the stored procedure call.
3. Manually define ports for result sets or additional stored procedures that you need to run.
4. Add the additional stored procedure calls in the SQL Editor.

You can configure an SQL query in the transformation SQL Editor. When you run the SQL transformation, the transformation processes the query, returns rows, and returns any database error.

To configure an SQL transformation to run a query, perform the following tasks:

1. Define the transformation properties including the database type to connect to.
2. Define the input and output ports.
3. Create an SQL query in the SQL Editor.

After you configure the transformation, configure the SQL transformation in a mapping and connect the upstream ports. Preview the data to verify the results.

SQL Transformation Ports

When you create an SQL transformation, the Developer tool creates the `SQLException` port by default. Add input ports, output ports, and pass-through ports in the **Ports** view.

The SQL transformation has the following types of ports:

Input

Receives source data that you can use in an SQL query.

Output

Returns database data from an SQL `SELECT` query.

Pass-through

Input-output ports that pass source data through the transformation without changing it.

SQLException

Returns SQL errors from the database. If no errors occur, returns `NULL`.

NumRowsAffected

Returns the total number of database rows affected by `INSERT`, `DELETE`, and `UPDATE` query statements for an input row. The Developer tool creates this port when you choose to include the update statistics in the output row.

Return Value

Receives the return value from a stored procedure.

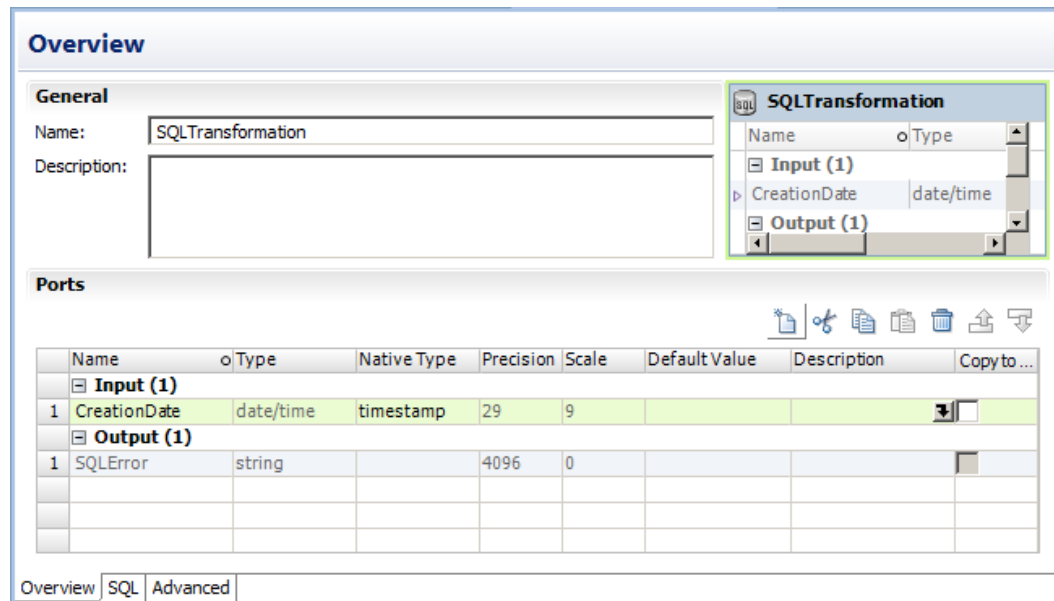
Input Ports

You can reference SQL transformation input ports with parameter binding in any type of SQL statement or stored procedure. You can create input ports in the SQL transformation for data that you do not intend to pass to output ports.

You must manually add ports if you are configuring an SQL query that has input parameters. When you import a stored procedure to the SQL transformation, the SQL transformation creates the input ports. You can add pass-through ports to pass data through the transformation without changing it.

You can add ports in the **Overview** view. When you add a port, enter the native datatype for the port. The native datatype is a datatype that is valid for the database that you are connecting to. When you configure a native datatype, a transformation datatype appears. If you drag rows to the SQL transformation, the Developer tool sets the native datatype based on datatypes that are valid for the database you are connecting to. Verify that the datatypes for columns that you use in the query are the same datatypes as the columns in the database.

The following figure shows the **CreationDate** input port in a reusable SQL transformation:



To add input ports, click **Input** in the **Ports** panel. Click **New**.

Note: If you select **Copy to Output** for a port, the input port becomes a pass-through port. Pass-through ports appear in the **Input** and **Output** sections of the **Ports** view.

Output Ports

SQL transformation output ports return values from a query statement or from a stored procedure.

You must define the output ports when you manually configure a SQL transformation. Define an output port for each stored procedure output parameter or for each port that a SELECT statement returns.

When you import a stored procedure, the Developer tool creates an output port for each output parameter that the procedure returns. If the procedure returns a result set, you must manually define the output ports in the result set. A stored procedure can return a result set and it can return output parameters that are not part of the result set in the same run. You must define the output ports for the result set fields and for the output parameters.

When you configure an output port, choose the native datatype for the port. The output port native datatype must match the datatype of the corresponding column in the database. When you configure the native datatype, the Developer tool defines the transformation datatype for the port.

For example, the SQL transformation contains the following SQL query for an Oracle database:

```
SELECT FirstName, LastName, Age FROM EMPLOYEES
```

You might configure the following output ports and the native datatypes in the SQL transformation:

Output Port	Native Datatype	Transformation Datatype
FirstNm	varchar2	string
LastNm	varchar2	string
Age	number	double

The number of the output ports and the order of the output ports must match the number and the order of the columns that the query or stored procedure returns. When the number of output ports is more than the number of columns in the query or stored procedure, the extra ports return a null value. When the number of output ports is less than the number of columns in the SQL, the Data Integration Service generates a row error.

If you change the database type that the transformation connects to, the Developer tool changes the native types of the output ports. The Developer tool might not choose the correct datatype for all the ports. If you change the database type, verify that the native datatype for each output port is the same datatype as the column in the database. For example, the Developer tool might choose nVarchar2 for a Oracle database column. You might need to change the datatype to varchar2.

Configure output ports in the SQL transformation **Overview** view.

Pass-through Ports

Pass-through ports are input-output ports that pass data through the transformation without changing the data. The SQL transformation returns data in the pass-through ports whether an SQL query returns rows or not.

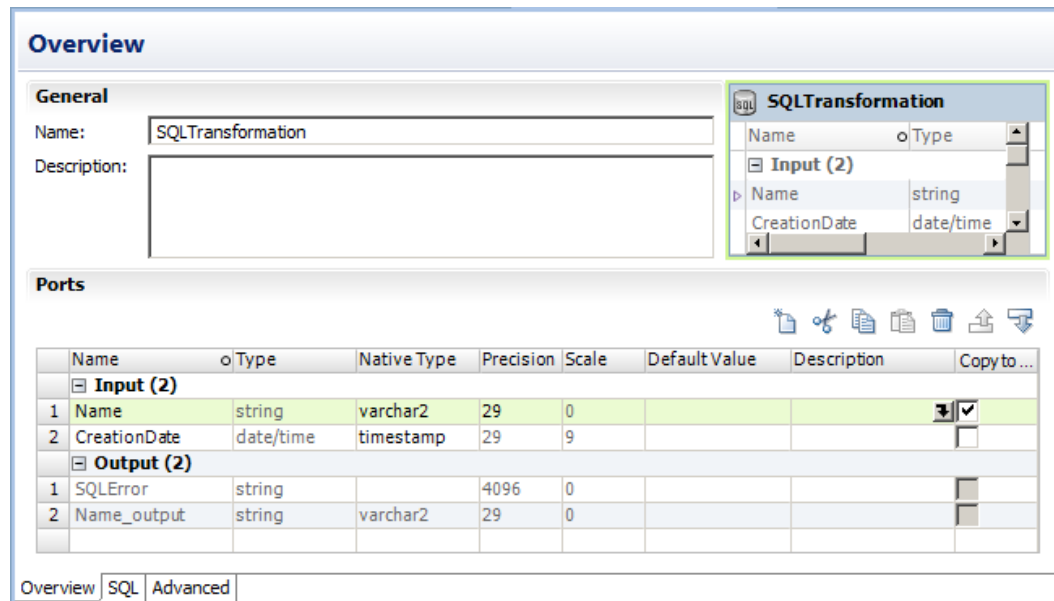
When the input row contains a SELECT query statement, the SQL transformation returns the data in the pass-through port for each row it returns from the database. If the query result contains multiple rows, the SQL transformation repeats the pass-through data in each row.

When a query returns no rows, the SQL transformation returns the pass-through column data with null values in the output columns. For example, queries that contain INSERT, UPDATE, and DELETE statements return no rows. When the query has errors, the SQL transformation returns the pass-through column data, the SQLError message, and null values in the output ports.

You cannot configure a pass-through port to return data from a SELECT query.

To create a pass-through port, create an input port and select **Copy to Output**. The Developer tool creates an output port and adds an "_output" suffix to the port name. You cannot change the output port that the Developer tool creates for a pass-through port. You cannot create an output port with the "_output" suffix.

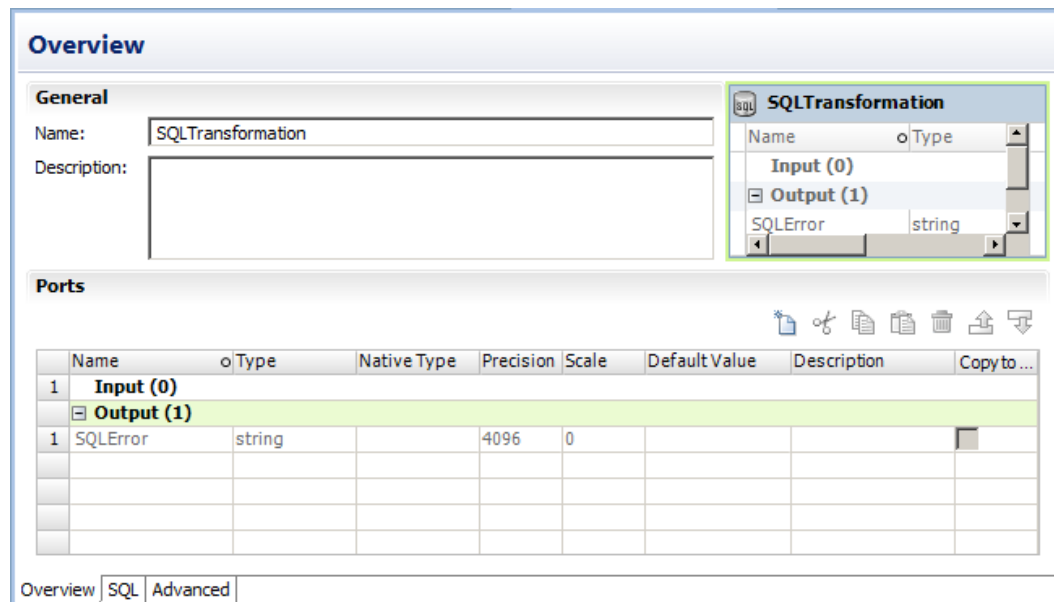
The following figure shows a Name pass-through port in a reusable SQL transformation:



SQLError Port

The SQLError port returns SQL errors from the database from stored procedures or SQL queries.

The following figure shows the SQLError port in a reusable SQL transformation:



When the SQL query contains syntax errors, the SQLError 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 Data 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
```

You can configure multiple query statements in the SQL query or you can call multiple stored procedures. When you configure the SQL transformation to continue on SQL error, the SQL transformation might return rows for one query statement, but return database errors for another query statement. The SQL transformation returns any database error in a separate row.

Number of Rows Affected

Enable the NumRowsAffected output port to return the number of rows that the INSERT, UPDATE, or DELETE query statements change for each input row. You can configure the NumRowsAffected output port for SQL queries.

The Data Integration Service returns the NumRowsAffected for each statement in the query. NumRowsAffected is disabled by default.

When you enable NumRowsAffected and the SQL query does not contain an INSERT, UPDATE, or DELETE statement, NumRowsAffected is zero in each output row.

When the SQL query contains multiple statements, the Data Integration Service returns the NumRowsAffected for each statement. NumRowsAffected contains the sum of the rows that the INSERT, UPDATE, and DELETE statements change for 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 Data Integration Service returns one row from the DELETE statement. NumRowsAffected is equal to one. The Data Integration Service returns one row from the SELECT statement, NumRowsAffected is zero. The Data Integration Service returns one row from the INSERT statement with NumRowsAffected equal to one.

SQL Transformation Advanced Properties

You can change the SQL transformation properties at any time. The default database type is Oracle. If the database you need to connect to is another database type, change the database type before you add ports to the transformation.

Configure the following properties on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal. When you configure the SQL transformation tracing level to Verbose Data, the Data Integration Service writes each SQL query it prepares to the mapping log.

Connection type

Describes how the Data Integration Service connects to the database. The connection type is static. The Data Integration Service connects one time to the database. Select a database connection object in the SQL transformation. Read only.

DB Type

Type of database that the SQL transformation connects to. Choose a database type from the list. You can choose Oracle, Microsoft SQL Server, or IBM DB2. Ignore the ODBC database type for the SQL transformation. The database type affects the datatypes that you can assign on the **Ports** tab. When you change the database type, the Developer tool changes the port datatypes for input, output, and pass-through ports.

Continue on Error Within Row

Continues processing the remaining SQL statements in a query after an SQL error occurs.

Include Statistics as Output

Adds a NumRowsAffected output port. The port returns the total number of database rows that INSERT, DELETE, and UPDATE query statements update 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.

Query Description

Description of the SQL query that you define in the transformation.

SQL Mode

Determines whether the SQL query is an external script or whether the query is defined in the transformation. The SQL Mode is Query. The SQL transformation runs a query that you define in the SQL Editor. Read only.

SQL Query

Displays the SQL query that you configure in the SQL Editor.

Has Side Effects

Indicates that the SQL transformation performs a function besides returning rows. The SQL transformation has a side effect when the SQL query updates a database. Enable **Has Side Effects** when the SQL query contains a statement such as as CREATE, DROP, INSERT, UPDATE, GRANT, or REVOKE.

The SQL transformation also has a side effect if the transformation returns NULL rows for SELECT statements that return no results. The rows might contain pass-through port values, SQL error information, or the NUMRowsAffected field.

Disable the **Has Side Effects** property in order to allow push-into optimization or early selection optimization. Default is enabled.

Return Database Output Only

The SQL transformation does not generate rows for SELECT statements that return 0 results, rows for other statements such as INSERT, UPDATE, DELETE, or COMMIT, or null rows.

Enable Push-Into Optimization

Enables the Data Integration Service to push the logic from a Filter transformation in the mapping to the SQL in the SQL transformation.

Maintain Row Order

Maintain the row order of the input data to the transformation. Select this option if the Data Integration Service should not perform any optimization that can change the row order.

When the Data Integration Service performs optimizations, it might lose an order established earlier in the mapping. You can establish order in a mapping with a sorted flat file source, a sorted relational source, or a Sorter transformation. When you configure a transformation to maintain row order, the Data Integration Service considers this configuration when it performs optimizations for the mapping. The Data Integration Service performs optimizations for the transformation if it can maintain the order. The Data Integration Service does not perform optimizations for the transformation if the optimization would change the row order.

Partitionable

The transformation can be processed with multiple threads. Clear this option if you want the Data Integration Service to use one thread to process the transformation. The Data Integration Service can use multiple threads to process the remaining mapping pipeline stages.

Disable partitioning for an SQL transformation when the SQL queries require that the transformation be processed with one thread. Or, you might want to disable partitioning for an SQL transformation so that only one connection is made to the database.

SQL Transformation Query

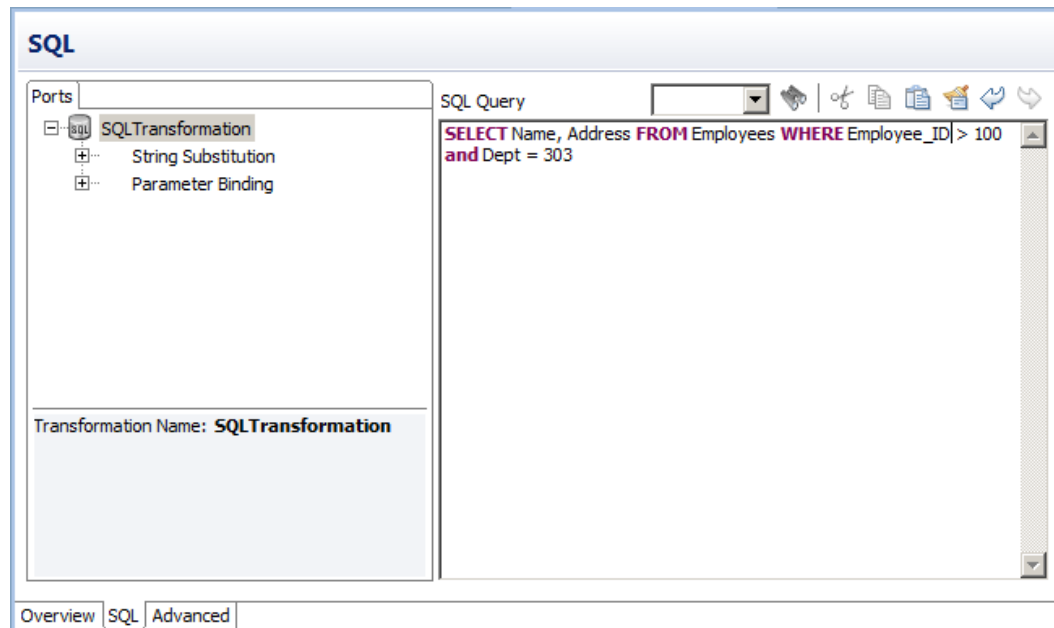
Create an SQL query in the SQL Editor to retrieve rows from a database or to update the database.

To create a query, type the query statement in the SQL Editor in the SQL view. You can enter up to 32767 characters in an SQL query statement. The SQL Editor provides a list of the transformation ports that you can reference in the query. You can double-click a port name to add it as a query parameter.

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 use constants in the SQL query. Enclose each string in a single quote (').

The following figure shows a sample SQL query:



You can create a static SQL query. The query statement does not change, but you can include parameters to change values. The Data Integration Service runs the query for each input row.

Define the SQL Query

Define an SQL query that runs the same query statements for each input row. You can change the query columns or table based on input port values in the row. You can also change the values in the WHERE clause based on input port values.

To change the data values in the WHERE clause for each input row, configure parameter binding.

To change the query columns or to change the table based on input port values, use string substitution.

Parameter Binding

To change the data in the query, configure the query parameters and bind the query parameters to input ports in the transformation. When you bind a parameter to an input port, identify the port by name in the query. The SQL Editor encloses the port name in question marks (?). The query data changes based on the value of the data in the port.

The following 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 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

Employee_ID	Dept
123	HR
130	Accounting

The Data 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'
```

String Substitution

Use string variables to replace components of query statements. For example, you can use the string variable to replace the table name in a query. Or, you can substitute the column names in a SELECT statement.

To substitute the table name, configure an input port to receive the table name from each input row. In the SQL Editor, select the port from the **String Substitution** list of ports. The Developer tool identifies the input port by name in the query and encloses the name with the tilde (~).

The following 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 Data Integration Service replaces the ~Table_Port~ variable with the table name value 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'
```

Input Row to Output Row Cardinality

When the Data 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.

Query statement processing

When a SELECT query is successful, the SQL transformation might retrieve multiple rows. When the query contains other statements, the Data Integration Service might generate a row that contains SQL errors or the number of rows affected.

Port configuration

The NumRowsAffected output port contains the number of rows that an UPDATE, INSERT, or DELETE statement changes for one input row. The SQL transformation returns the number of rows affected for

each statement in a query. When the SQL transformation contains pass-through ports, the transformation returns the column data at least one time for each source row.

The maximum row count configuration

The Max Output Row Count limits the number of rows the SQL transformation returns from SELECT queries.

Error rows

The Data Integration Service returns row errors when it encounters connection errors or syntax errors. The SQL transformation returns errors to the SQLError port.

Continue on SQL Error

You can configure the SQL transformation to continue processing when there is an error in an SQL statement. The SQL transformation does not generate a row error.

Query Statement Processing

The type of SQL query determines how many rows the SQL transformation returns. The SQL transformation 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.

The following table lists the output rows that the SQL transformation generates for different types of query statements when no errors occur in query mode:

Query Statement	Output Rows
UPDATE, INSERT, DELETE only	One row for each statement in the query.
One or more SELECT statements	Total number of database rows retrieved.
DDL queries such as CREATE, DROP, TRUNCATE	One row for each statement in the query.

Port Configuration

When you enable Include Statistics as Output, the Developer tool creates the NumRowsAffected port. The Data Integration Service returns at least one row with the NumRowsAffected based on the statements in the SQL query.

The following table lists the output rows the SQL transformation generates if you enable NumRowsAffected:

Query Statement	Output Rows
UPDATE, INSERT, DELETE only	One row for each statement with the NumRowsAffected for the statement.
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.

Maximum Output Row Count

You can limit the number of rows that 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. The SQL transformation returns no rows from the second statement.

To configure unlimited output rows, set **Max Output Row Count** to zero.

Error Rows

The Data Integration Service returns row errors when it encounters a connection error or syntax error. The SQL transformation returns SQL errors to the SQLError port.

When you configure a pass-through port or the NumRowsAffected port, the SQL transformation returns at least one row for each source row. When a query returns no data, the SQL transformation returns the pass-through data and the NumRowsAffected values, but it returns null values in the output ports. You can remove rows with null values by passing the output rows through a Filter transformation.

The following table describes the rows that the SQL transformation generates for UPDATE, INSERT, or DELETE query statements:

NumRowsAffected Port or Pass-Through Port Configured	SQLError	Rows Output
Neither port configured	No	One row with NULL in the SQLError port.
Neither port configured	Yes	One row with the error in the SQLError port.
Either port configured	No	One row for each query statement with the NumRowsAffected or the pass-through column data.
Either port configured	Yes	One row with the error in the SQLError port, the NumRowsAffected port, or the pass-through port data.

The following table describes the number of output rows that the SQL transformation generates for SELECT statements:

NumRowsAffected Port or Pass-Through Port Configured	SQLError	Rows Output
Neither port configured	No	One or more rows, based on the rows returned from each SELECT statement.
Neither port configured	Yes	One row greater than the sum of the output rows for the successful statements. The last row contains the error in the SQLError port.

NumRowsAffected Port or Pass-Through Port Configured	SQLError	Rows Output
Either port configured	No	One or more rows, based on the rows returned for each SELECT statement: <ul style="list-style-type: none"> - If NumRowsAffected is enabled, each row contains a NumRowsAffected column with a value zero. - If a pass-through port is configured, each row contains the pass-through column data. When the query returns multiple rows, the pass-through column data is duplicated in each row.
Either port configured	Yes	One or more rows, based on the rows returned for each SELECT statement. The last row contains the error in the SQLError port: <ul style="list-style-type: none"> - When NumRowsAffected is enabled, each row contains a NumRowsAffected column with value zero. - If a pass-through port is configured, each row contains the pass-through column data. When the query returns multiple rows, the pass-through column data is duplicated in each row.

The following table describes the number of output rows that the SQL transformation generates for DDL queries such as CREATE, DROP, or TRUNCATE:

NumRowsAffected Port or Pass-Through Port Configured	SQLError	Rows Output
Neither port configured	No	One row with NULL in the SQLError port.
Neither port configured	Yes	One row that contains the error in the SQLError port.
Either port configured	No	One row that includes the NumRowsAffected column with value zero and the pass-through column data.
Either port configured	Yes	One row with the error in the SQLError port, the NumRowsAffected column with value zero, and the pass-through column data.

Continue on SQL Error

You can choose to ignore an SQL error that occurs in a query statement. Enable **Continue on SQL Error within a Row**. The Data Integration Service continues to run the rest of the SQL statements for the row.

The Data Integration Service does not generate a row error. However, the SQLError port contains the failed SQL statement and error messages.

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 Data Integration Service continues processing the INSERT statement.

Disable the **Continue on SQL Error** option to troubleshoot database errors and to associate errors with the query statements that caused the errors.

Filter Optimization with the SQL Transformation

The Data Integration Service can apply filter optimization with an SQL transformation if the filter condition references only pass-through ports and the SQL transformation does not have side effects.

The SQL transformation has side effects under the following circumstances:

- The SQL query updates a database. The SQL query contains a statement such as as CREATE, DROP, INSERT, UPDATE, GRANT, or REVOKE.
- The transformation returns NULL rows for SELECT statements that return no results. The rows might contain pass-through port values, SQL error information, or the NUMRowsAffected field.

The Data Integration Service can apply the early selection and push-into optimization methods with the SQL transformation.

Early Selection Optimization with the SQL Transformation

The Data Integration Service can perform early selection optimization with an SQL transformation if the filter condition references only pass-through ports and the SQL transformation does not have side effects.

The SQL transformation has side effects under the following circumstances:

- The SQL query updates a database. The SQL query contains a statement such as as CREATE, DROP, INSERT, UPDATE, GRANT, or REVOKE.
- The transformation returns NULL rows for SELECT statements that return no results. The rows might contain pass-through port values, SQL error information, or the NUMRowsAffected field.

Enabling Early Selection Optimization with the SQL Transformation

Enable early selection optimization in the SQL transformation if the SQL transformation has no side effects.

1. Enable the **Return Database Output Only** option in the SQL transformation **Advanced Properties**.
2. Clear **Has Side Effects** in the transformation **Advanced Properties**.
3. If the transformation has a **NumAffectedRows** port, remove the port.

Push-Into Optimization with the SQL Transformation

With push-into optimization, the Data Integration Service pushes the filter logic from a Filter transformation in the mapping to the query in the SQL transformation.

Use the following rules and guidelines when you enable push-into optimization with the SQL transformation:

- The transformation SQL query must only contain SELECT statements.
- The transformation SQL query must be a valid subquery.
- The filter condition cannot refer to the SQL Error or NumRowsAffected fields.
- The names of the output ports must match the names of the columns in the SQL SELECT statement. When you reference an output port in a filter condition, the Data Integration Service pushes the corresponding port name to the SQL query. You can add aliases to the SQL if the columns in the query do not match the output port names. For example, `SELECT mycolname1 AS portname1, mycolname2 AS portname2`.
- The transformation cannot have side effects.

Push-Into Optimization with the SQL Transformation Example

An SQL transformation retrieves orders by customer ID. A Filter transformation that appears after the SQL transformation returns only the rows where the order amount is greater than 1000.

The Data Integration Service pushes the following filter into a SELECT statement in the SQL transformation:

```
orderAmount > 1000
```

Each statement in the SQL query becomes a separate subquery of the SELECT statement that contains the filter.

The following query statement shows the original query statement as a subquery in the SELECT statement:

```
SELECT <customerID>, <orderAmount>, ... FROM (original query statements) ALIAS WHERE  
<orderAmount> > 1000
```

If the SQL query has multiple statements, each statement is included in a separate subquery. The subquery has the same syntax, including the WHERE clause.

The ports *customerID* and *orderAmount*, are the names of the output ports in the SQL transformation. The subquery does not include pass-through ports, the SQL error, or the SQL statistics ports. If you push multiple filters into the SQL transformation, the WHERE clause contains all the filters.

Enabling Push-Into Optimization with the SQL Transformation

Enable push-into optimization by configuring properties on the SQL transformation **Advanced Properties** tab.

1. Clear **Has Side Effects**.
2. Enable **Return Database Output Only**.
3. Set **Max Out Row Count** to zero.
4. Enable push-into optimization.

SQL Transformation Example with an SQL Query

You are a developer in the HR department of Hypostores corporation. Hypostores maintains employee payroll information in a separate database from the human resources employee data. The Human Resources department needs to query a single view of the employees and salaries across regions.

You want to create a logical data object mapping that shows a single view of the employee data and salary data in an employee logical data object.

Create a logical data object mapping with the employee data source. Include an SQL transformation to retrieve the salary and hire date from the payroll database.

Logical Data Object Mapping

The logical data object mapping contains the following objects:

Employee table

Input relational table of employee data from the Human Resources database.

Salary table

A table in the Payroll database that contains the employee salary and hire date. The database is an Oracle database.

SQL transformation

Transformation that retrieves the hire date and salary for each employee row. The transformation connects to a Payroll database and runs an SQL query against the Salary table in the database.

Logical data object

Contains the combined view of the employee and the salary data. The logical data object receives the output from the SQL transformation.

SQLErrors file

The SQLErrors file is flat file that contains any SQL error from the database. The Data Integration Service writes at least one row to the SQLErrors file for each input row. If no SQL errors occur, the SQLError column contains NULL. Review the SQLErrors file to troubleshoot errors.

Salary Table

The Salary table is a relational table in the Payroll database. The table contains employee data that the Payroll department maintains. The SQL transformation retrieves the hire date and the employee salary from the Salary table.

The following table shows some rows from the Salary table:

Employee_Num	HireDate	Salary
10	3-May-97	232000
11	11-Sep-01	444000
12	17-Oct-89	656000
13	13-Aug-07	332100

Employee Table

The source is the Employee table from the Human Resources database.

The following table shows sample rows from the Employee table:

EmpID	LastName	FirstName	DeptId	Phone
10	Smith	Martha	FIN	(415) 552-1623
11	Jones	Cynthia	ENG	(415) 552-1744
12	Russell	Cissy	SLS	(415) 552-1656
13	Goyal	Girish	FIN	(415) 552-1656

SQL Transformation

The SQL transformation retrieves the employee hire date and salary from the Salary table of the Payroll database. The Salary table is in an Oracle database.

Use the following steps to configure the SQL transformation:

1. Configure the SQL transformation properties.
2. Define the ports.
3. Create the SQL query.
4. Configure the database connection for the SQL transformation.

Define SQL Transformation Properties

Configure the SQL transformation properties in the **Advanced Properties** view.

Configure the following properties:

Database type

The database type is Oracle. When you define the ports, you can choose port datatypes that are applicable for Oracle.

Continue on Error Within Row

Disable. Stop processing if an SQL error occurs in the row.

Include Statistics as Output

Disable. Do not create the NumRowsAffected output port.

Define the Ports

Define input ports for each column in the employee source table. Select **Copy to Output** to change the input ports to pass-through ports for the columns. When you select **Copy to Output**, the Developer tool creates the corresponding output port for each port that you copy.

Create the following input pass-through ports:

Name	Type	Native Type	Precision	Scale	Copy to Output
EmpID	decimal	number(p,2)	4	0	x
LastName	string	varchar2	30	0	x
FirstName	string	varchar2	20	0	x
DeptID	string	varchar2	4	0	x
Phone	string	varchar2	16	0	x

The SQL transformation has the following output ports:

Name	Type	Native Type	Precision	Scale
EmpID	decimal	number(p,s)	4	0
LastName	string	varchar2	30	0
FirstName	string	varchar2	20	0
DeptID	string	varchar2	4	0
Phone	string	varchar2	16	0
HireDate	date/time	timestamp	29	0
Salary	decimal	number(p,s)	8	2

The Developer tool adds the "_output" suffix to each output port that it creates when you select **Copy to Output**.

Manually define the output ports for the hire date and salary columns. The SQL transformation returns the hire date and salary columns from the Salary table in the ports.

Define the SQL Query

Create an SQL query to select the hiredate and salary for each employee from the Salary table.

Define the query in the SQL transformation SQL view.

Enter the following query in the SQL Editor:

```
select HIREDATE,SALARY,from Salary where EMPLOYEE_NUM =?EmpID?
```

Hiredate, Salary, and Employee_Num are column names in the Salary table.

?EMPID? is a parameter that contains the value of the EmpID port.

Define the Database Connection

In the **Runtime** view, select a database connection object for the database that the SQL transformation connects to. Select an Oracle database connection object.

Output

Connect the SQLError port and the EmpID_output port to the SQLErrors flat file. The SQLError port contains null values unless an SQL error occurs.

Connect EmpID and the other output ports to the logical data object.

The SQL transformation returns a row that contains data from the Employee table and includes the hire date and salary from the Salary table.

The following table shows some rows from the logical data object:

EmpID	LastName	FirstName	DeptId	Phone	HireDate	Salary
10	Smith	Martha	FIN	(415) 552-1623	19970303 00:00:00	2320.00
11	Jones	Cynthia	ENG	(415) 552-1744	20010911 00:00:00	4440.00
12	Russell	Cissy	SLS	(415) 552-1656	19891017 00:00:00	6560.00
13	Goyal	Girish	FIN	(415) 552-1660	20070813 00:00:00	3210.00

Stored Procedures

You can call a stored procedure from an SQL transformation. You can use a stored procedure to automate tasks in a relational database. Stored procedures accept user-defined variables, conditional statements, and other features that standard SQL statements do not support.

The SQL transformation connects to a relational database to run the stored procedure. The SQL transformation can call stored procedures from Oracle, IBM DB2, Microsoft SQL Server, Sybase, and ODBC. A stored procedure is kept in the database, and it runs in the database.

Create an ODBC connection to call a stored procedure from a Sybase database. You must also create an ODBC connection to call a stored procedure from a Microsoft SQL Server database on non-Windows operating systems.

A stored procedure is a pre-compiled collection of Transact-SQL, PL-SQL, or other database procedural statements. Stored procedure syntax varies based 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.
- Retrieve data by a value.
- Drop and re-create indexes.

You can use a stored procedure to perform a query or calculation that you would otherwise include in a transformation. For example, if you have a well-tested stored procedure for calculating sales tax, you can perform that calculation with the stored procedure instead of recreating the same calculation in an Expression transformation.

A stored procedure can accept input and then return a result set of rows. A stored procedure can run a DDL task that requires no input and then returns no output.

You can configure the SQL transformation to run more than one stored procedure. For each stored procedure that you configure, configure transformation ports to match the stored procedure parameters. Each stored procedure can pass data back to output ports.

The database that contains the stored procedure has user permissions. You must have permissions to run the stored procedure on the database.

Note: A stored function is similar to a stored procedure, except that a function returns a single value. The SQL transformation can run stored functions.

SQL Transformation Ports for Stored Procedures

The SQL transformation input and output ports correspond to the input and output parameters in the stored procedure.

When you import a stored procedure, the Developer tool determines the database type from the database connection. It generates input and output ports in the SQL transformation from the parameters in the stored procedure. The Developer tool determines the native datatype of each port from the parameter of the stored procedure.

When you manually configure the SQL transformation, you need to configure the input and the output ports in the transformation. When you configure the database type, the SQL transformation changes the native datatype for each port based on the database type that you enter.

The following types of data can pass between the SQL transformation and the stored procedure:

Input and output parameters

The SQL transformation sends parameters to the stored procedure and the SQL transformation receives parameters from the stored procedure in the input and output ports.

Return value

If the stored procedure passes a return value, the Developer tool creates a Return Value port.

SQL errors

The SQL transformation returns errors from the stored procedure in the SQLError port.

Input and Output Parameters

When you call a stored procedure from an SQL transformation, each field that the call statement references identifies an input or output port. When you import a stored procedure, the Developer tool generates the stored procedure call statement. Otherwise, you need to manually configure the call statement.

You can edit the call statement in the **SQL** view of the transformation.

The call statement has the following format:

```
?RETURN_VALUE? = call <stored proc name>( ?Field1?, ?Field2?, . . . )
```

Enclose the port names in question marks. The port names do not have to match the parameter names in the stored procedure. The output ports must be in the same order as parameters in a SELECT query.

You can use a stored procedure that contains INOUT parameters. The SQL transformation identifies INOUT parameters by the input port name. The output port has the prefix `output_`. The Data Integration service binds the input port and output port to the same parameter.

You can configure an SQL transformation to return a result set. When the stored procedure returns a result set, the Developer tool cannot create output ports for the columns in the result set. When you import the stored procedure, you must manually enter the ports and configure the stored procedure call.

Return Value

A return value is a code or text string that defines that status of the stored procedure. When the stored procedure has a return value, the SQL transformation has a **Return Value** port.

Most databases can pass a return value after running a stored procedure. The return value can contain an integer value, or it can contain a value that you define in the stored procedure. For example, a stored procedure might return "Success" when the procedure succeeds.

If a stored procedure returns a result set instead of a single return value, the SQL transformation receives the first return value from the procedure.

Stored Procedure Result Sets

You can configure a SQL transformation to receive a result set from a stored procedure. A stored procedure returns multiple rows in a result set. The SQL transformation can return each row to the mapping.

The following stored procedure returns a result set:

```
CREATE OR REPLACE FUNCTION fetchEMPinfo
(p_State IN VARCHAR2 )
return types.cursorType
AS
my_cursor types.cursorType;
BEGIN
OPEN my_cursor FOR SELECT EMP_ID, NAME, CITY FROM EMP WHERE STATE = p_State ORDER BY
EMP_ID;
RETURN my_cursor;
END;
```

When you import the stored procedure, the Developer tool creates a stored procedure call statement similar to the following syntax:

```
call FETCHEMPINFO (?P_STATE?)
```

The input parameter is p_state. The Developer tool does not create the output ports for you. You must manually create the output ports with the same datatypes as the stored procedure parameters.

For example, the resultSet contains the EMP_ID, EMPNAME, and CITY columns. Create output ports for these columns.

You must also manually update the SQL call with the output columns using the following syntax:

```
(?EMP_ID?,?EMPNAME?,?CITY?) = call FETCHEMPINFO (?P_STATE?)
```

Result Sets with Different Databases

Configure stored procedures to return result sets using different syntax based on the database type.

Oracle

An Oracle stored function returns results with a cursor:

```
create or replace function sp_ListEmp return types.cursorType
as
l_cursor types.cursorType;
begin
open l_cursor for select ename, empno from emp order by ename;
return l_cursor;
end;
```

Oracle also accepts cursors as input parameters. You cannot configure cursors as input parameters with the SQL transformation.

Microsoft SQL Server

A Microsoft SQL Server stored procedure returns a result set stored procedure with a select statement in the procedure body or with the return type explicitly declared as a table.

```
Create PROCEDURE InOut(
@inout varchar(100) OUT
)
AS
BEGIN
set @inout = concat(@inout, '__')
select * from mytable;
END
```

IBM DB2

An IBM DB2 stored procedure can return a resultset with an open cursor. The number of result sets it returns are declared in the RESULT SET clause. The stored procedure opens a cursor and returns it. The below example returns 2 open cursors.

```
CREATE PROCEDURE TESTMULTIRS
  (IN i_cmacct CHARACTER(5))
RESULT SETS 2
LANGUAGE SQL
BEGIN

  DECLARE csnum INTEGER;

  --Declare serial cursors to consume less resources
  --You do not need a rollable cursor.

  DECLARE getDeptNo CHAR(50); --Be careful with the estimated length.
  DECLARE getDeptName CHAR(200);
  DECLARE c1 CURSOR WITH RETURN FOR s1;
  SET getDeptNo = 'SELECT DEPTNO FROM DEPT';
  SET getDeptName = 'SELECT DEPTNAME FROM DEPT';

  PREPARE s1 FROM getDeptNo;
  OPEN c1;

  END;
```

Sybase

A Sybase stored procedure returns a result set stored procedure with a select statement in the procedure body or with the return type explicitly declared as a table.

```
CREATE PROCEDURE FETCHEMPINFO
(
  @p_State VARCHAR(5),
  @e_id INT OUTPUT,
  @e_name VARCHAR(50) OUTPUT
)
AS
BEGIN
  SET NOCOUNT ON
  SELECT EMP_ID, NAME FROM EMP WHERE STATE = @p_State ORDER BY EMP_ID
  SET NOCOUNT OFF
  SELECT @e_id AS EMP_ID, @e_name AS NAME
  RETURN
  END
  GO

  --Configure the following variables to execute the procedure.

  DECLARE @p_State VARCHAR(5)
  DECLARE @EMPID int
  DECLARE @EMPNAME varchar(50)

  SET @p_State = 'CA'
  exec FETCHEMPINFO @p_State, @e_id = @EMPID, @e_name = @EMPNAME
  GO
```


Result Set Rows

Some stored procedures return output parameters in addition to the result set rows. The SQL transformation returns output parameters in the last row. It does not include the single-occurring output parameters in the result set rows.

For example, you write a stored procedure that receives an employee ID and returns the employee name in output parameter 1 and department in output parameter 2. The stored procedure also returns a row for each sick day that the employee took throughout the year. The row contains the date, the number of hours, and the reason for the absence.

The result set contains a different number of rows for each employee. Each row in the result set contains an empty employee name and department. The SQL transformation returns the employee name and department after the result set. The employee name and department appear in the last row.

Stored Procedure Example

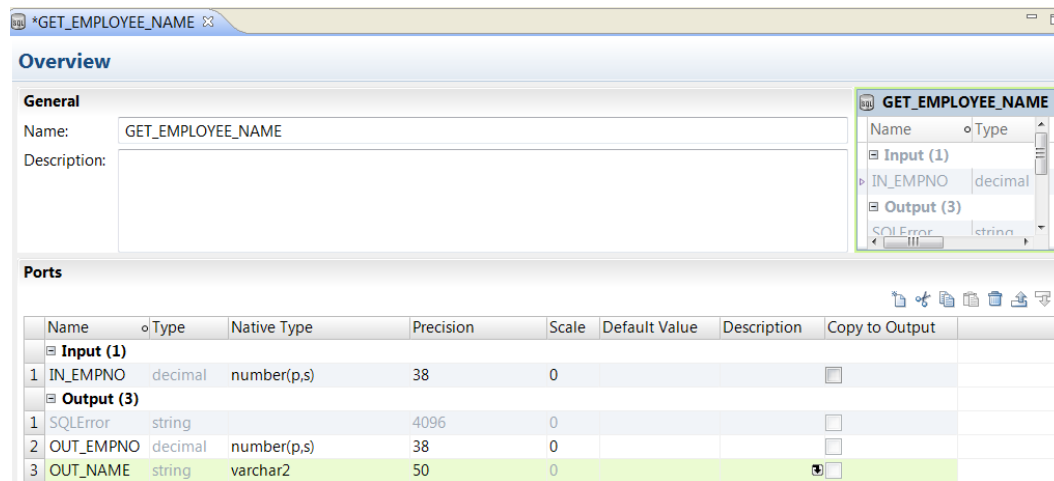
You can call a stored procedure that returns data to the SQL transformation.

The following stored procedure receives an employee number and returns one row with the employee number and the employee name:

```
CREATE OR REPLACE PROCEDURE SP_GETNAME
(IN_EMPNO IN NUMBER, OUT_EMPNO NUMBER, OUT_NAME OUT STRING)
AS
BEGIN
SELECT EMP_KEY,EMP_NAME into OUT_EMPNO , OUT_NAME from EMP_TABLE where EMP_KEY=IN_EMPNO;
END;/"
```

To create the SQL transformation, import the stored procedure. The Developer tool creates the input ports and the output ports. The port names are the same as the parameter names in the stored procedure.

The following figure shows the ports for the SQL transformation:



The Developer tool creates the following stored procedure call to retrieve the employee name:

```
call SP_GETNAME (?IN_EMPNO?,?OUT_EMPNO?,?OUT_NAME?)
```

You can view the stored procedure call in the SQL Editor. Any SQL errors appear in the SQLError port.

SQL Transformation Connection

Configure the SQL transformation connection in the transformation run-time properties. You might need to configure a run-time connection if you did not specify a connection when you created the transformation.

You can define a different connection name than the connection that you selected to create the SQL transformation. You must select a connection of the same database type as the database type in the SQL transformation **Advanced** properties.

You can configure a parameter for the SQL transformation connection name. You must define the parameter in the **Parameters** view of the mapping before you assign it to a run-time connection.

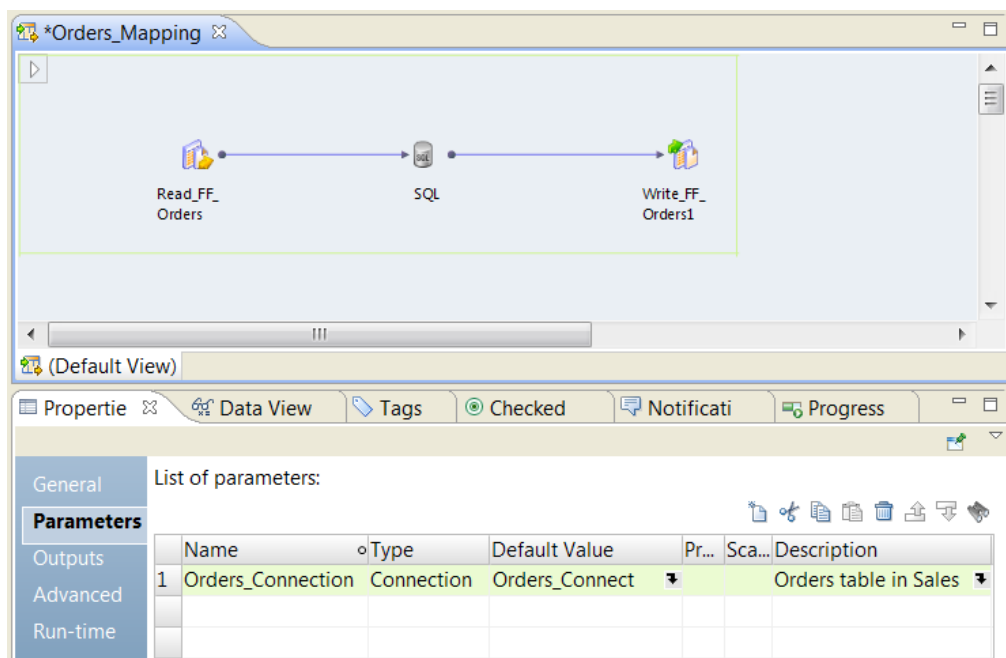
Creating a Connection Name Parameter

You can specify a user-defined parameter in the run-time connection name for an SQL transformation. The Developer tool creates a mapping parameter for the connection instead of a transformation parameter.

1. Create a mapping that includes the SQL transformation.
2. Click the SQL transformation **Run-time** tab.
3. In the **Connection Name**, click the selection arrow and choose **Assign Parameters**.
4. In the **Assign Parameter** dialog box, click **New**.
5. In the **Parameters** dialog box, enter a connection parameter name and a description for the parameter. The parameter type defaults to connection.
6. Click **OK** in the Parameters dialog box and in the **Assign Parameters** dialog box.

The Developer tool creates a mapping parameter and assigns it to the Connection Name. The parameter name appears in the Run-time properties.

7. To view a list of the mapping parameters, click inside the editor and then click the mapping **Parameters** tab.



Manually Creating an SQL Transformation

You can manually create an SQL transformation. Manually create the transformation when you configure a transformation that runs an SQL query. You might also manually create a transformation that calls a stored procedure when the procedure is not available to import. When you manually create the transformation, you configure the input and output ports and type the SQL statements in the SQL Editor.

1. Select a project or a folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
The **New** dialog box appears.
3. Select the SQL transformation.
4. Click **Next**.
5. Select **Create as Empty**.
6. Enter a name for the transformation and enter the repository location for the transformation.
7. Click **Finish**
8. Click the **Overview** view to add ports to the transformation.
9. To add an input port, click **Input** in the **Ports** panel to indicate where to add the port. Click the **New** button and enter the port name, the native type, and the precision.
The default database type is Oracle. The Developer tool shows native types for Oracle databases unless you change the database type on the **Advanced** view.
10. To add an output port, click **Output** in the **Ports** panel before you add the port. Click the **New** button and enter the port name, the native type, and the precision.
The **SQLException** port is the first output port by default.
11. In the **Advanced** view, select the database type that the SQL transformation connects to. Configure other advanced properties for error handling and other optional properties.
When you choose the database type, the Developer tool changes the native datatypes of the ports on the **Overview** view.
12. Type the SQL query or stored procedure call on the **SQL** view. Select ports for parameter binding or string substitution in the **SQL Editor**.
If the stored procedure returns a result set, you must enter the stored procedure call with a syntax similar to the following syntax: (?Field1?,?Field2?,?Field3?) = **call** Stored_Procedure_Name (?Input_Parm?).

RELATED TOPICS:

- [“Define the SQL Query” on page 620](#)

Creating an SQL Transformation from a Stored Procedure

You can configure an SQL transformation by importing a stored procedure from a database connection.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.

The **New** dialog box appears.

3. Select the SQL transformation.
4. Click **Next**.
5. Select **Create from an existing stored procedure**.
6. Browse for and select a database connection.
7. Browse for and select the stored procedure to import.
8. Enter a name and location for the transformation.
9. Click **Finish**.

The Developer tool creates the ports and the stored procedure call.

10. If the stored procedure returns a result set, you must manually add the output ports and then reconfigure the stored procedure call.
 - a. In the **Overview** view, click **Output** in the **Ports** panel. Click the **New** button and enter the output port name, the native type, and the precision.
 - b. In the **SQL** view, change the stored procedure call to use the following syntax: (?Field1?, ?Field2?, ?Field3?) = **call** Stored_Procedure_Name (?Input_Parm?)

You can select the input and the output parameters from the **Parameter Binding** list of ports in the SQL Editor.

CHAPTER 44

Standardizer Transformation

This chapter includes the following topics:

- [Standardizer Transformation Overview, 629](#)
- [Standardization Strategies, 629](#)
- [Standardization Properties, 630](#)
- [Configuring a Standardization Strategy, 631](#)
- [Standardizer Transformation Advanced Properties, 631](#)
- [Standardizer Transformation in a Non-native Environment, 632](#)

Standardizer Transformation Overview

The Standardizer transformation is a passive transformation that examines input strings and creates standardized versions of those strings.

The Standardizer transformation creates columns that contain standardized versions of input strings. The transformation can replace or remove strings in the input data when creating these columns.

For example, you can use the Standardizer transformation to examine a column of address data that contains the strings Street, St., and STR. You can replace all instances of these strings with the string St.

Within a Standardizer transformation, you can create multiple standardization strategies. Each strategy can contain multiple standardization operations. The Standardizer transformation provides a wizard that you use to create strategies.

Standardization Strategies

Use standardization strategies to create columns with standardized versions of input strings.

When you configure a standardization strategy, you add one or more operations. Each operation implements a specific standardization task.

You can add the following types of operations to a standardization strategy:

Replace Reference Table Matches With Valid Values

Replace strings that match reference table values with the "Valid" value from the reference table.

Replace Reference Table Matches With Custom Strings

Replace strings that match reference table values with a user-defined replacement string.

Remove Reference Table Matches

Remove strings that match reference table values.

Replace Custom Strings

Replace user-defined strings with a user-defined replacement string.

Remove Custom Strings

Remove user-defined strings.

Important: You can change the order of operations. The order of operations can change the output of a strategy because each operation reads the results of the preceding operation.

Standardization Properties

To configure properties for standardization strategies and operations, select the **Strategies** view in the Standardizer transformation.

Strategy Properties

Strategy properties apply to all the operations within a strategy. You can configure the following strategy properties:

Remove multiple spaces

Replaces multiple consecutive spaces with one space.

Remove trailing and leading spaces

Removes spaces from the beginning and end of data strings.

Delimiters

Determines the delimiters that define search tokens. For example, if you choose "Semicolon," the Standardizer transformation searches the string "oranges;apples;" and finds the strings "oranges" and "apples." The transformation uses the space delimiter by default.

Operation Properties

You can configure properties for the following types of standardization operations.

Reference Table Operations

Reference table operations include the following properties:

- **Reference table.** Determines the reference table you use to standardize data. Click **Browse** to select a reference table.
- **Case Sensitive.** Determines whether input strings must match the case of reference table entries.
- **Replace With.** Replaces input strings that match reference table entries with the text you provide. Applies to replacement operations only.
- **Scope.** Specifies the part of the input string that contains the reference table value.

Custom String Operations

Custom string operations include the following properties:

- **Match Tokens With.** Defines the search strings to find within input data.
- **Replace With.** Replaces input strings that match the search strings you specify. Applies to replacement operations only.
- **Scope.** Specifies the part of the input string to search.

Configuring a Standardization Strategy

To configure a standardization strategy, edit the settings in the **Strategies** view of the Standardizer transformation.

1. Select the **Strategies** view.
2. Click **New**.
The **New Strategy** wizard opens.
3. Click the **Inputs** field to select ports for the strategy.
4. Configure the strategy properties and click **Next**.
5. Choose an operation and click **Next**.
6. Configure the operation properties.
7. Optionally, click **Next** to add more operations to the strategy.
8. After you add all operations to the strategy, click **Finish**.
9. Optionally, add more strategies to the transformation.
10. Optionally, change the order that the transformation processes strategies or operations. Select an strategy or operation and click **Move Up** or **Move Down**.

Standardizer Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Standardizer transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Standardizer Transformation in a Non-native Environment

The Standardizer transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 45

Union Transformation

This chapter includes the following topics:

- [Union Transformation Overview, 633](#)
- [Groups and Ports, 634](#)
- [Union Transformation Advanced Properties, 634](#)
- [Union Transformation Processing, 635](#)
- [Creating a Union Transformation, 635](#)
- [Union Transformation in a Non-native Environment, 636](#)

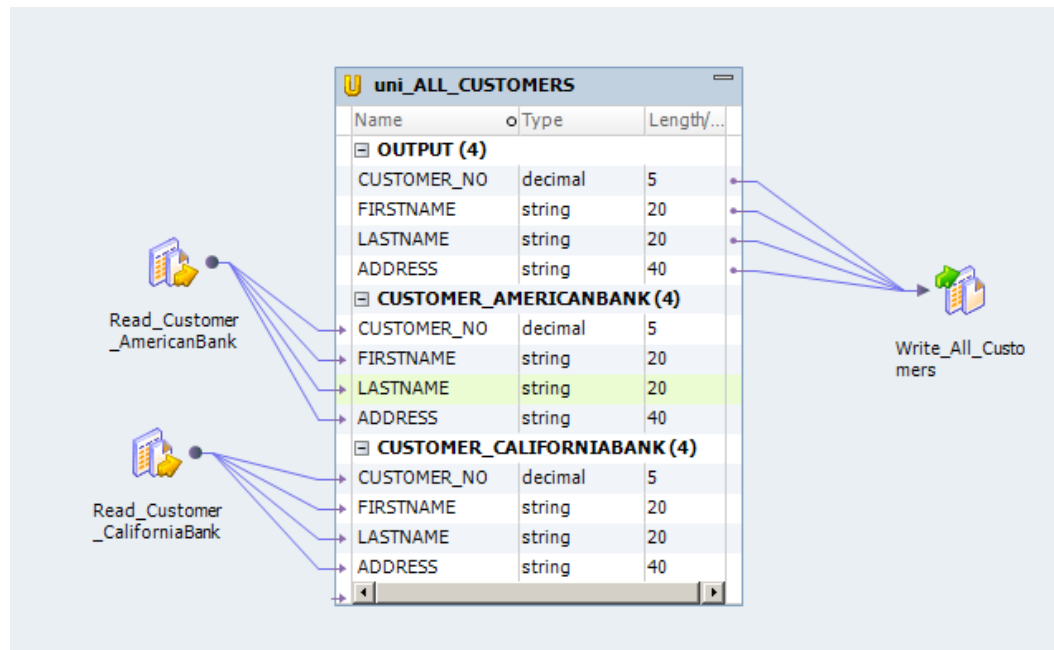
Union Transformation Overview

Use the Union transformation to merge data from multiple pipelines or pipeline branches into one pipeline branch.

The Union transformation is an active transformation with multiple input groups and one output group. It merges sources with matching ports, and it passes data through an output group that has the same port structure as the input groups. Use a Union transformation in the Developer tool to merge data from multiple sources without removing the duplicate rows.

For example, you want to combine customer account data from American Bank and California Bank. You can create a mapping that contains a Union transformation to merge data from the source objects and write to a target object.

The following figure shows a mapping with a Union transformation:



Groups and Ports

A Union transformation has multiple input groups and one output group. You can create one or more input groups. The Developer tool creates one output group. You cannot create, edit, or delete the output group. Each group must have matching ports.

To create ports, you can copy ports from a transformation, or you can manually create them. When you create ports, the Developer tool creates input ports in each input group and output ports in the output group. The Developer tool uses the output port names you specify for each input and output port. The Developer tool also uses the same metadata for each port, such as datatype, precision, and scale.

You can connect the input groups from different branches in a single pipeline or from 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 a port in one input group, but do not connect the same port in another input group, the Data Integration Service passes NULLs to the unconnected port.

Union Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service displays log details for the Union transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Union Transformation Processing

Use the Union transformation to merge data from multiple pipelines or pipeline branches into one pipeline branch. The Data Integration Service processes all input groups in parallel. It concurrently reads sources connected to the Union transformation and passes blocks of data to the input groups of the transformation. The Union transformation processes the blocks of data based on the order it receives the blocks from the Data Integration Service. The Union transformation does not block input data on the input groups.

Creating a Union Transformation

You can create a reusable or non-reusable Union transformation.

Creating a Reusable Union Transformation

Create a reusable Union transformation to use in multiple mappings or mapplets.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File > New > Transformation**.
The **New** dialog box appears.
3. Select the Union transformation.
4. Click **Next**.
5. Enter a name for the transformation.
6. Click **Finish**.
The transformation appears in the editor.
7. Click the **New** button to add a port to the transformation.
8. Edit the port to set the name, datatype, and precision.
9. Select the **Groups** view.
10. Click the **New** button to add an input group.
11. Click the **Advanced** view and edit the transformation properties.

Creating a Non-Reusable Union Transformation

Create a non-reusable Union transformation in a mapping or mapplet.

1. In a mapping or mapplet, drag a Union transformation from the Transformation palette to the editor.
The transformation appears in the editor.
2. On the **General** tab, edit the transformation name and the description.

3. Select all the ports from the upstream transformation and drag them to the Union transformation. The ports appear as ports in an input group and output group of the Union transformation.
4. Click **New** on the **Groups** tab in the **Properties** view to add an input group.
Another input group appears which has similar ports as the existing input group.
5. Select the ports in the output group of the Union transformation and drag them to the downstream transformation in the mapping.

Union Transformation in a Non-native Environment

The Union transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Supported with restrictions in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Union Transformation in a Streaming Mapping

Streaming mappings have mapping validation restrictions.

Mapping Validation

Mapping validation fails in the following situation:

- The transformation is configured to merge data from streaming and non-streaming pipelines.

Union Transformation on the Databricks Spark Engine

CHAPTER 46

Update Strategy Transformation

This chapter includes the following topics:

- [Update Strategy Transformation Overview, 637](#)
- [Update Strategy Transformations in Dynamic Mappings, 638](#)
- [Flagging Rows Within a Mapping, 638](#)
- [Specifying Update Options for Individual Targets, 640](#)
- [Update Strategy Transformation in a Non-native Environment, 640](#)

Update Strategy Transformation Overview

The Update Strategy transformation is an active transformation that flags a row for insert, update, delete, or reject. Use an Update Strategy transformation to control changes to existing rows in a target based on a condition that you apply.

As an active transformation, the Update Strategy transformation might change the number of rows passed through it. The Update Strategy transformation tests each row to see if it meets a particular condition, and then flags the row accordingly. The transformation passes rows that it flags for insert, update, or delete to the next transformation. You can configure the transformation to pass rows flagged for reject to the next transformation or to drop rows flagged for reject.

For example, you might use the Update Strategy transformation to flag all customer rows for update when the mailing address has changed. Or, you might flag all employee rows for reject for people who no longer work for the organization.

You can use an Update Strategy transformation to write results to a relational database target when the mapping runs on the Spark engine. The mapping uses a JDBC connection string.

Setting the Update Strategy

To define an update strategy, complete the following steps:

1. To control how rows are flagged for insert, update, delete, or reject within a mapping, add an Update Strategy transformation to the mapping. Use an Update Strategy transformation to flag rows destined for the same target for different database operations, or to reject rows.
2. Define insert, update, and delete options for individual targets when you configure the mapping. On a target-by-target basis, you can allow or disallow inserts and deletes for all rows flagged for insert or delete. You can choose different ways to handle updates for all rows flagged for update.

3. If the mapping runs on the Spark run-time engine, you can choose Use Hive Merge in Update Strategy transformation advanced properties. When a query uses a MERGE statement instead of INSERT, UPDATE or DELETE statements, processing is usually more efficient.

Update Strategy Transformations in Dynamic Mappings

You can use an Update Strategy transformation in a dynamic mapping. You can configure dynamic ports in the transformation and reference the generated ports.

You can reference a dynamic port or a generated port in the Update Strategy transformation. However, if the generated port does not exist at run time, the mapping fails.

If you use a dynamic port in the update strategy expression, the dynamic port can have no more than one generated port.

You can parameterize the update strategy expression. Use an expression type parameter and enter the entire expression as the default parameter value.

Flagging Rows Within a Mapping

Add an Update Strategy transformation to a mapping to flag individual rows for insert, update, delete, or reject.

Define an update strategy expression to test each row to see if it meets a particular condition. Then, assign each row a numeric code to flag the row for a particular database operation.

The following table lists the constants for each database operation and their numeric equivalent:

Operation	Constant	Numeric Value
Insert	DD_INSERT	0
Update	DD_UPDATE	1
Delete	DD_DELETE	2
Reject	DD_REJECT	3

The Data Integration Service treats any other value as an insert.

Update Strategy Expressions

Enter an update strategy expression in the Expression Editor.

The update strategy expression uses the IIF or DECODE function from the transformation language to test each row. For example, the following IIF statement flags a row for reject if the entry date is after the apply

date. Otherwise, the statement flags the row for update: `IIF((ENTRY_DATE > APPLY_DATE), DD_REJECT, DD_UPDATE)`

You can configure parameters in an update strategy expression. Create parameters or browse parameters in the Expression Editor.

If the transformation is in a dynamic mapping, the generated fields in the transformation might change. You can parameterize the complete update strategy expression. If you use a parameter to define the expression, the Developer tool cannot validate the expression. An expression parameter cannot contain another parameter.

Update Strategy Transformation Advanced Properties

Configure advanced properties to help determine how the Data Integration Service processes data for the Update Strategy transformation.

You can define the following advanced properties for the Update Strategy transformation on the Advanced tab:

Forward Rejected Rows

Determines whether the Update Strategy transformation passes rejected rows to the next transformation or drops rejected rows. By default, the Data Integration Service forwards rejected rows to the next transformation. The Data Integration Service flags the rows for reject and writes them to the reject file. If you do not select Forward Rejected Rows, the Data Integration Service drops rejected rows and writes them to the mapping log file.

Use Hive Merge

Determines whether the Update Strategy transformation uses Hive MERGE to perform updates on Hive targets when the mapping runs on the Spark engine. When a query uses a MERGE statement instead of INSERT, UPDATE or DELETE statements, processing is more efficient.

The mapping ignores the Hive MERGE option in the following scenarios:

- The mapping runs on the Databricks Spark engine.
- The mapping runs on the Blaze engine.
- MERGE is restricted by Hive implementation on particular Hadoop distributions.

Mappings that run on the Databricks Spark engine use the Databricks "MERGE INTO" SQL command instead of Hive MERGE. In mappings that run on the Blaze engine and scenarios where MERGE is restricted, the run-time engine uses INSERT, UPDATE and DELETE to perform the operation.

The mapping log contains results of the operation, including whether restrictions affected results.

When the update affects partitioning or bucketing columns, updates to the columns are omitted.

Note: The Developer tool and the Data Integration Service do not validate against this restriction. If the Update Strategy expression violates these restrictions, the mapping might produce unexpected results.

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Aggregator and Update Strategy Transformations

When you connect Aggregator and Update Strategy transformations as part of the same pipeline, put the Aggregator before the Update Strategy transformation. In this order, the Data Integration Service performs

the aggregate calculation, and then flags rows that contain the results of this calculation for insert, update, delete, or reject.

If you put the Update Strategy before the Aggregator transformation, you must consider how the Aggregator transformation handles rows flagged for different operations. In this order, the Data Integration Service flag rows for insert, update, delete, or reject before it performs the aggregate calculation. How you flag a row determines how the Aggregator transformation treats values in that row used in the calculation. For example, if you flag a row for delete and then use the row to calculate the sum, the Data Integration Service subtracts the value in this row. If you flag a row for reject and then use the row to calculate the sum, the Data Integration Service does not include the value in this row. If you flag a row for insert or update and then use the row to calculate the sum, the Data Integration Service adds the value in this row to the sum.

Specifying Update Options for Individual Targets

After you use an Update Strategy transformation to flag each row for a particular database operation, define insert, update, and delete options for each target in the mapping. You can disallow inserts or deletes for rows flagged for insert or delete. You can choose different ways to handle updates for all rows flagged for update.

Define the update strategy options in the Advanced properties of a target data object in a mapping. You can set the following update strategy options:

Insert

Inserts all rows flagged for insert into a target. Default is enabled.

Delete

Deletes all rows flagged for delete from a target. Default is enabled.

Update Strategy

Update strategy for existing rows. Select one of the following strategies:

- **Update as Update.** Updates all rows flagged for update. This is the default value.
- **Update as Insert.** Inserts all rows flagged for update.
- **Update else Insert.** Updates all rows flagged for update if they exist in the target and then inserts any remaining rows marked for insert.

Truncate Table

Truncates the target before loading data. Default is disabled.

Update Strategy Transformation in a Non-native Environment

The Update Strategy transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.

- Databricks Spark engine. Supported with restrictions.

Update Strategy Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

General Restrictions

You can use the Update Strategy transformation on the Hadoop distributions that support Hive ACID.

If the Update Strategy transformation receives multiple update rows for the same primary key value, the transformation selects one random row to update the target.

If multiple Update Strategy transformations write to different instances of the same target, the target data might be unpredictable.

The Blaze engine executes operations in the following order: deletes, updates, inserts. It does not process rows in the same order as the Update Strategy transformation receives them.

Hive targets always perform Update as Update operations. Hive targets do not support Update Else Insert or Update as Insert.

Mapping Validation and Compile Validation

Mapping validation fails in the following situations:

- The Update Strategy transformation is connected to more than one target.
- The Update Strategy transformation is not located immediately before the target.
- The Update Strategy target is not a Hive target.
- The Update Strategy transformation target is an external ACID table.
- The target does not contain a primary key.
- The Hive target property to truncate the target table at run time is enabled.
- The Hive target property to create or replace the target table at run time is enabled.

The mapping fails in the following situation:

- The target is not ORC bucketed.
- The Hive target is modified to have fewer rows than the actual table.

Compile validation errors occur and the mapping execution stops in the following situations:

- The target table is not enabled for transactions.
- The Hive version is earlier than 0.14.

Using Hive Target Tables

To use a Hive target table with an Update Strategy transformation, you must create the Hive target table with the following clause in the Hive Data Definition Language: `TBLPROPERTIES ("transactional"="true")`.

To use an Update Strategy transformation with a Hive target, verify that the following properties are configured in the `hive-site.xml` configuration set associated with the Hadoop connection:

```
hive.support.concurrency      true
hive.enforce.bucketing      true
hive.exec.dynamic.partition.mode nonstrict
hive.txn.manager              org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
hive.compactor.initiator.on   true
hive.compactor.worker.threads 1
```

Update Strategy Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

General Restrictions for Hive Targets

You can use the Update Strategy transformation on the Hadoop distributions that support Hive ACID.

You can also use the Update Strategy transformation to write mapping results to JDBC-compliant relational targets.

The Update Strategy transformation does not forward rejected rows to the next transformation when the target is a Hive table or JDBC-compliant table.

If the Update Strategy transformation receives multiple update rows for the same primary key value, the transformation selects one random row to update the target.

If multiple Update Strategy transformations write to different instances of the same target, the target data might be unpredictable.

If the mapping runs on the Spark engine, you can choose the Use Hive Merge option. The option has the following restrictions:

- A single row for delete or update cannot match multiple rows in the target. When the mapping violates this restriction, the mapping fails with a runtime error.
- If you configure the Update Strategy expression to update partitioning or bucketing columns, the mapping ignores the Hive MERGE option and does not update the columns.

Note: The Developer tool and the Data Integration Service do not validate against these restrictions. If the expression or the mapping violates these restrictions, the mapping might run, but the results will not be as expected.

Hive targets always perform Update as Update operations. Hive targets do not support Update Else Insert or Update as Insert.

Using Hierarchical Data in Update Strategy Expressions

You can configure the expression to use the primitive types in a complex data field.

For example, you have a nested array of struct data types defined by the following statement:

```
addresses array<struct<city:varchar(15), apartment:smallint, pincode:int>>
```

You can use the city column in the update strategy expression since it is a primitive type using the following expression:

```
IIF( (addresses[0].city = 'NEW YORK'), DD_INSERT, DD_UPDATE)
```

Mapping Validation

Update Strategy transformation output fields must exactly match the target input fields.

Mapping validation fails in the following situations.

- The Update Strategy transformation is connected to more than one target.
- The Update Strategy transformation is not located immediately before the target.
- The Update Strategy transformation target is an external ACID table.
- The target does not contain a connected primary key.
- The property to enable truncation of the target table at run time is selected.

- One of the following target strategies for the target table at run time is selected:
 - Create or replace the target table
 - ApplyNewColumns
 - ApplyNewSchema
 - Fail

The mapping fails in the following situations when the target is a Hive target:

- The target table is not enabled for transactions.
- The target is not ORC bucketed.

Using Hive Target Tables

To use a Hive target table with an Update Strategy transformation, you must create the Hive target table with the following clause in the Hive Data Definition Language: `TBLPROPERTIES ("transactional"="true")`.

To use an Update Strategy transformation with a Hive target, verify that the following properties are configured in the `hive-site.xml` configuration set associated with the Hadoop connection:

```
hive.support.concurrency      true
hive.enforce.bucketing      true
hive.exec.dynamic.partition.mode nonstrict
hive.txn.manager              org.apache.hadoop.hive.q1.lockmgr.DbTxnManager
hive.compactor.initiator.on   true
hive.compactor.worker.threads 1
```

Update Strategy Transformation on the Databricks Spark Engine

You can use the Update Strategy transformation to perform INSERT, UPDATE, and other operations. You can also perform INSERT and UPDATE operations in select Delta Lake columns.

General Restrictions

You can only use the Update Strategy transformation on the Databricks Spark Engine when you use a Delta Lake table target.

The Update Strategy transformation is supported only on Databricks distributions that support Delta ACID.

Mapping Validation

Mapping validation fails in the following situations.

- The Update Strategy transformation is connected to more than one target.
- The Update Strategy transformation is not located immediately before the target.
- The Update Strategy transformation target is an external ACID table.
- The target does not contain a connected primary key.
- The property to enable truncation of the target table at run time is selected.
- One of the following target strategies for the target table at run time is selected:
 - Create or replace the target table
 - ApplyNewColumns
 - ApplyNewSchema
 - Fail

CHAPTER 47

Web Service Consumer Transformation

This chapter includes the following topics:

- [Web Service Consumer Transformation Overview, 644](#)
- [WSDL Selection, 647](#)
- [Web Service Consumer Transformation Ports, 647](#)
- [Web Service Consumer Transformation Input Mapping, 649](#)
- [Web Service Consumer Transformation Output Mapping, 652](#)
- [Web Service Consumer Transformation Advanced Properties, 655](#)
- [Filter Optimizations, 659](#)
- [Creating a Web Service Consumer Transformation, 661](#)
- [Web Service Consumer Transformation Example, 663](#)

Web Service Consumer Transformation Overview

The Web Service Consumer transformation connects to a web service as a web service client to access or transform data. The Web Service Consumer transformation is a multiple-group transformation.

A web service uses open standards, such as SOAP, WSDL, and XML. SOAP is the communications protocol for web services. The web service client request and the web service response are SOAP messages. A WSDL is an XML schema that describes the protocols, formats, and signatures of the web service operations.

Web service operations include requests for information, requests to update data, or requests to perform tasks. For example, the Web Service Consumer transformation sends a SOAP request to run a web service operation called `getCustomerOrders`. The transformation passes a customer ID in the request. The web service retrieves the customer and order information. The web service returns the information to the transformation in a SOAP response.

The Web Service Consumer transformation connects to a web service using an endpoint URL defined in the WSDL, in a web services connection, or in an endpoint URL input port. You enable security for web services in a web services connection.

SOAP Messages

The Web Service Consumer transformation uses Simple Object Access Protocol (SOAP) to exchange information with the web services provider and to request web services. SOAP defines the format for web service request and response messages.

When you transform data with a Web Service Consumer transformation, the transformation generates a SOAP request and connects to the web service. The transformation connects to the web service using an endpoint URL defined in the WSDL object, in a web services connection, or in an endpoint URL input port. The SOAP request contains the information that the web service requires to run the requested operation. The web service operation returns data to the transformation in a SOAP response. The transformation maps data from the SOAP response and returns the data in output ports.

The Web Service Consumer transformation encodes the SOAP message headers in ISO-8859-1.

The transformation can process SOAP messages with document/literal encoding. The document/literal style requires an XML schema to describe the SOAP message. SOAP messages are formed from the XML. When a SOAP message contains multiple-occurring elements, the groups of elements form levels in the XML hierarchy. The groups are related when one level is nested within another.

A SOAP request message can contain hierarchical data. For example, the Web Service Consumer transformation sends a request to add customer orders to a sales database. The transformation passes two groups of data in a SOAP request message. One group contains a customer ID and name, and the other group contains order information. The order information occurs multiple times.

A SOAP response message can contain hierarchical data. For example, the Web Service Consumer transformation generates a SOAP request for customer orders. The web service returns an order header and multiple-occurring order detail elements in the SOAP response.

WSDL Files

A WSDL file contains a description of the data to be passed to the web service so that the sender and the receiver understand the data to exchange. You must import a WSDL file to the repository before you can create a Web Service Consumer transformation.

The WSDL describes the operations to perform on the data and a binding to a protocol or transport, so that the web service consumer can send the request message in the correct format. The WSDL describes the network address to connect to the web service.

The WSDL includes information about how to encode SOAP request and response messages. SOAP encoding determines the format of the SOAP message body. It describes the format for request and response messages that the web service uses to communicate to the web service consumer. Web service developers can use a variety of toolkits to create web services. Toolkits support different ways of encoding SOAP messages.

The Web Service Consumer transformation supports the document/literal SOAP encoding style. You can use WSDL 1.1 with the Web Service Consumer transformation. You cannot use WSDL attachments such as MIME, DIME, and MTOM messages.

Operations

A web service contains an operation for each action supported by the web service.

For example, a web service can have an operation named `getcustomerid` that receives a customer name and responds with the customer details. The operation input includes an element for the customer name. The operation output includes elements for customer details based on the customer name.

When you configure a Web Service Consumer transformation, you define how the transformation maps data to the operation input and how the transformation maps data from the operation output. You configure the following information in the transformation:

Input mapping

Define how to map the transformation input ports to the web service operation input nodes. The operation input defines the elements in the SOAP request for the operation.

Output mapping

Define how to map the web service operation output nodes to the transformation output ports. The operation output defines the elements in a SOAP response for the operation.

Web Service Security

You enable security for web services in a web services connection. You can configure the following types of security:

Web Service Security

The Data Integration Service can include a web service security header when it sends a SOAP request to the web service provider. The web service security header contains authentication information so the web service provider can authenticate the Data Integration Service.

The Web Service Consumer transformation supplies the user name token. The Data Integration Service creates a separate security SOAP header in the SOAP request and passes the request to the web service provider.

You can use the following types of web service security in a web services connection:

- **PasswordText.** The Data Integration Service does not change the password in the WS-Security SOAP header.
- **PasswordDigest.** The Data Integration Service combines the password with a nonce and a time stamp. The Data Integration Service applies a SHA hash on the password, encodes it in base64 encoding, and uses the encoded password in the SOAP header.

Transport layer security

Security implemented on top of the transport layer (TCP layer) of TCP/IP using Secure Sockets Layer (SSL). Web services use Hypertext Transfer Protocol over SSL (HTTPS) as a web address for secure message transport. Web Service Consumer transformations can use TLS 1.2, TLS 1.1, or TLS 1.0. You can use the following authentication with transport layer security: HTTP authentication, proxy server authentication, and SSL certificates.

SSL authentication

You can use SSL authentication when you connect through the HTTPS protocol.

You can use the following types of SSL authentication:

- One-way SSL authentication
- Two-way SSL authentication

HTTP authentication

You can use HTTP authentication when you connect through the HTTP protocol.

You can use the following HTTP authentication methods:

- Basic authentication
- Digest authentication

- NT LAN Manager (NTLM) authentication

WSDL Selection

Before you create a Web Service Consumer transformation, you must import a WSDL file into the model repository. You can only import WSDL files that are defined with a SOAP binding style of document/literal.

The WSDL defines the operation signature of the web service you want to run. When you import a WSDL, the Developer tool creates a physical data object that you can reuse for other transformations.

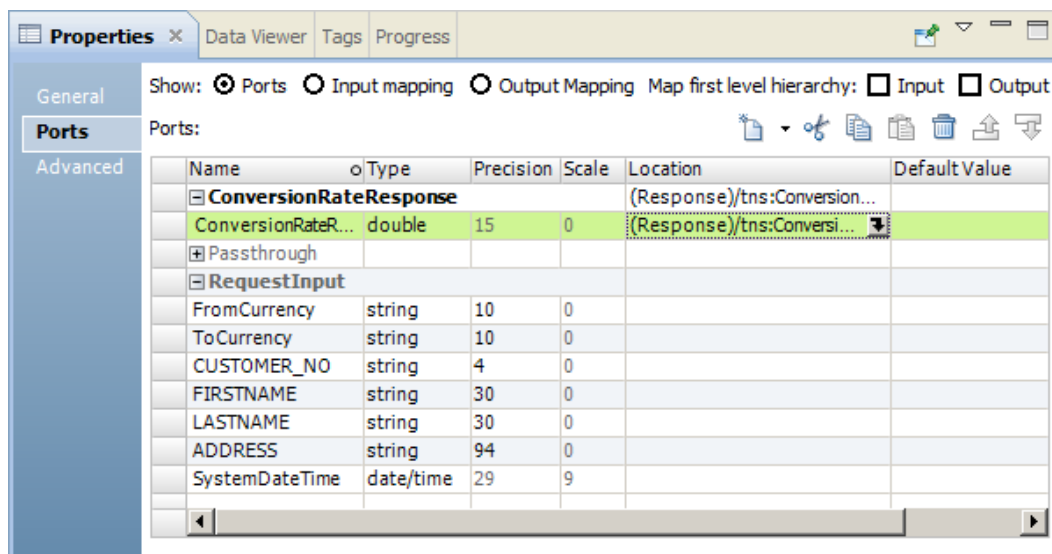
A WSDL can define multiple operations. When you create a Web Service Consumer transformation, select which operation you want to run. You can view the operation input and the operation output hierarchies in the Web Service Consumer transformation. The hierarchies define the structure of the SOAP request message and the SOAP response message.

You can also import a WSDL with one-way input operation. You must create dummy output ports when you import a WSDL with one-way input operation.

Web Service Consumer Transformation Ports

When you view the transformation ports, show the ports if you do not need to view the operation hierarchy. When you show the ports, you can define groups, define ports, and map nodes from the operation output to the output ports.

The following figure shows the ports for a non-reusable Web Service Consumer transformation:



A Web Service Consumer transformation can have multiple input groups and multiple output groups. When you create ports, create groups and add the ports to the groups. Define the ports in a group hierarchy based on the structure of the operation input or operation output hierarchy. Add a key to relate a child group to a parent group. All groups except the lowest group in the hierarchy must have primary keys. All groups in the hierarchy except the root group must have foreign keys.

The transformation has a root input group named RequestInput. You must add a primary key to the root input group. The key must be string, bigint, or integer.

You can add additional pass-through ports to the root input group. Pass-through ports pass data through the transformation without modifying the data. The pass-through port can occur one time in the input data. You can add the pass-through port to any output group. Associate the output port to the input port. The input value that you pass through a SOAP request repeats in the output rows from the SOAP response.

You can also add HTTP headers, cookie ports, a dynamic URL port, and ports for web service security authentication to the root input group. Data in the root group occurs one time.

To map an operation output node to an output port, click the field in the **Location** column and expand the hierarchy in the **Select Location** dialog box. Then, choose a node from the hierarchy.

HTTP Header Input Ports

A web service might require additional HTTP headers. You can create input ports in the root input group to pass the additional header information to the web service provider.

To add an HTTP header and an HTTP port, select the root input group and click the arrow next to the **New** button. Then, click **HTTP Header**. Enter a header name and port name.

You can create multiple HTTP headers.

Other Input Ports

You can add predefined input ports to the Web Service Consumer transformation.

You can add the following predefined input ports:

Cookie port

You can configure the Web Service Consumer transformation to use cookie authentication. The remote web service server tracks the web service consumer users based on the cookies. You can increase performance when a mapping calls a web service multiple times.

When you project the cookie port to a web service request message, the web service provider returns a cookie value in the response message. You can pass the cookie value to another transformation downstream in the mapping or you can save the cookie value in a file. When you save the cookie value in a file, you can configure the cookie as input to the Web Service Consumer transformation.

You can project the cookie output port to any of the Web Service Consumer transformation output groups.

Endpoint URL port

The Web Service Consumer transformation connects to a web service using an endpoint URL. You can define the endpoint URL in the WSDL file, in a web services connection, or in an endpoint URL input port. When the transformation receives the URL dynamically in a port, the Data Integration Service overrides the URL defined in the WSDL file or in the web services connection.

The Web Service Consumer transformation can have one URL port value for each web service request. Add an Endpoint URL port to the root input group.

WS-Security ports

You enable web service security in a web services connection. When you enable web service security, you must define the user name and password in a web services connection or in WS-Security input ports.

When you add WS-Security ports, you pass the user name and the password through input ports in the transformation. When the transformation receives the user name and password dynamically in ports, the Data Integration Service overrides the values defined in the web services connection.

Note: A web services connection has one user name and password for HTTP and WS-Security authentication.

To add predefined input ports, click the root input group in the **Ports** area. Click the arrow next to the **New** button and then click **Other Ports**. Choose the ports to add.

Web Service Consumer Transformation Input Mapping

When you view the transformation ports, show the input mapping to view the operation input hierarchy. When you show the input mapping, you can define input groups, define input ports, and map input ports to operation input nodes.

The input mapping includes the following areas:

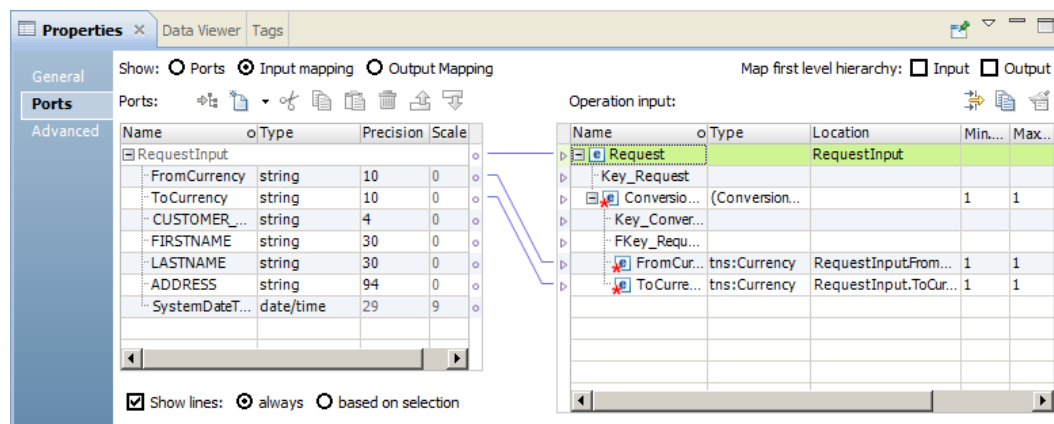
Ports

Create the transformation input groups and input ports in the **Ports** area.

Operation Input

The **Operation Input** area shows the nodes in the SOAP request message that the Web Service Consumer transformation sends to the web service. The WSDL data object that you use to create the transformation defines the operation input hierarchy.

The following figure shows the input mapping for a non-reusable Web Service Consumer transformation:



After you create input ports, map the input ports from the **Ports** area to the nodes in the **Operation Input** area. When you map an input port to a node in the operation input, the location of the port appears in the **Location** column in the **Operation Input** area.

The Developer tool maps nodes in the first level of the operation input to input ports when you choose to map the first level of the input hierarchy. The Developer tool also creates the ports to perform the mapping. If the first level of the hierarchy contains a multi-occurring parent node with one or more multi-occurring child nodes, the Developer tool does not map the first level of the hierarchy.

You can map XML data from one string or text input port to the entire SOAP request message. When you map XML data to the entire SOAP request, you cannot map ports to nodes in the operation input.

You can choose to view the lines that connect the input ports to the nodes in the operation input.

RELATED TOPICS:

- [“Generating Web Service SOAP Messages Overview” on page 674](#)

Rules and Guidelines to Map Input Ports to Nodes

Review the following rules when you map input ports to nodes in the operation input hierarchy:

- You can map an input port to one node in the hierarchy. You can map the same port to any number of keys in the hierarchy.
- The input port and the node must have compatible data types.
- You can map ports from one input group to multiple hierarchy levels in the operation input.
- You must map input ports to the keys in the operation input. Any port that you map to a key must be a string, integer, or bigint datatype. Map data to the keys in all levels in the operation input above the hierarchy level that you are including in the SOAP message. Include the foreign keys for all levels above and including the level you are mapping.

Note: You do not have to map input ports to keys if you are mapping only the lowest level of the operation input hierarchy.

- You can map multiple string, bigint, or integer input ports to a key in the **Operation Input** area to create a composite key. When you click the **Location** field for a composite key, you can reorder the input ports or remove one of the ports.

Customize View Options

You can change the operation input hierarchy to show keys in the **Operation Input** area. You can also show grouping constructs that define how to order nodes.

Click the **Customize View** button () in the **Operation Input** area. Enable any of the following options:

Sequence, Choice, and All

Show a line that indicates whether an element definition is sequence, choice, or all.

Nodes in an all group must all be included in the SOAP message.

Nodes in a sequence group must be in the order specified in the WSDL.

At least one node in a choice group must appear in the SOAP message.

Keys

View the keys in the **Operation Input** area. The **Operation Input** area includes keys for each group. You can add a key to an input port in the **Ports** area.

Mapping Input Ports to the Operation Input

When you show the transformation input mapping, you can define input groups, define input ports, and map input ports to operation input nodes.

1. Open a Web Service Consumer transformation.
2. To show the transformation input mapping, use one of the following methods:

- For a reusable transformation, click the **Overview** view. Choose to show the input mapping.
 - For a non-reusable transformation, click the **Ports** tab in the **Properties** view. Choose to show the input mapping.
3. Define a primary key for the root input group.
 4. To add an input group or port to the **Ports** area, use one of the following methods:

Option	Description
Drag a node	Drag a group node or a child node in the Operation Input area to an empty column in the Ports area. If the node is a group node, the Developer tool adds a group without ports.
Manually add a group or port	To add a group, click the arrow next to the New button and then click Group . To add a port, click the arrow next to the New button and then click Field .
Drag a port from another transformation	In the editor, drag a port from another transformation to the Web Service Consumer transformation.
Copy a port	Select ports from another transformation and copy them to the Ports area. To copy ports, you can use keyboard shortcuts or you can use the copy and paste buttons in the Developer tool.
Select Map first level of hierarchy	Select Map first level of hierarchy . The Developer tool maps nodes in the first level of the operation input to input ports and groups. The Developer tool also creates the input ports and groups to perform the mapping.

5. If you manually create a port or you copy a port from another transformation, click the **Location** column in the **Operation Input** area and choose a port from the list.
6. To map input ports as a composite key, use one of the following methods:

Option	Description
Drag input ports	Select two or more input ports and drag them to a key in the operation input hierarchy.
Select input ports from the Select Location dialog box	Click the Location column of a key in the operation input hierarchy and then select the input ports.

7. To clear the locations of nodes, use one of the following methods:

Option	Description
Click Clear	Select one or more nodes in the Operation Input area and click Clear .
Delete the lines that connect ports to nodes	Select one or more lines that connect the input ports to the nodes in the operation input and press Delete .

8. If the associated WSDL data object includes anyType elements, any elements, anyAttribute attributes, derived type elements, or substitution groups, choose objects in the **Operation Input** area. In the **Type** column for a node, click **Choose** and then choose one or more types, elements, or attributes from the list.
9. To map XML data from a string or text input port to the complete SOAP request, right-click the port and select **Map as XML**.

Web Service Consumer Transformation Output Mapping

When you view the transformation ports, show the output mapping to view the operation output hierarchy. When you show the output mapping, you can define output groups, define output ports, and map operation output nodes to output ports.

The output mapping includes the following areas:

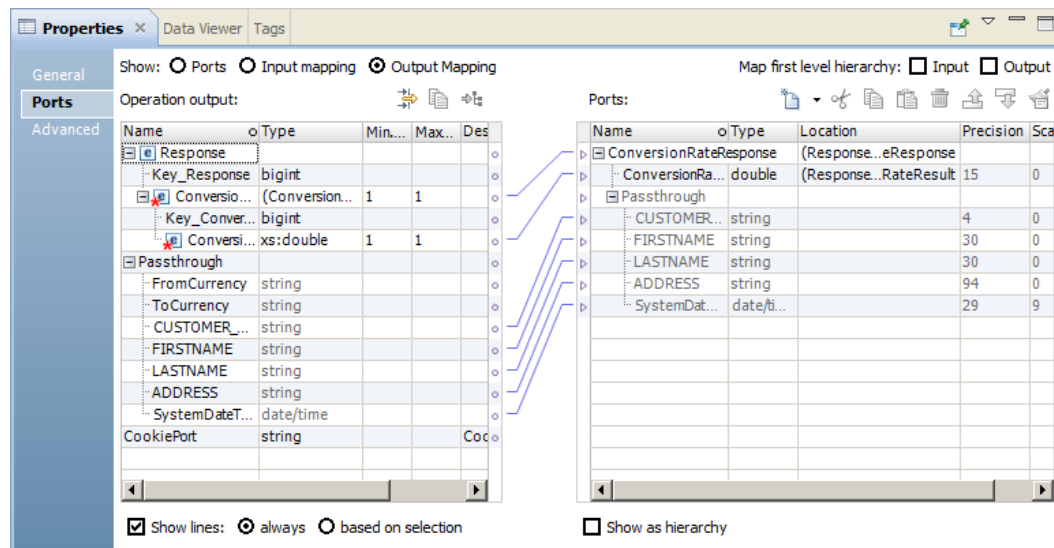
Operation Output

The **Operation Output** area shows the nodes in the SOAP response message that the web service returns to the Web Service Consumer transformation. The WSDL data object that you use to create the transformation defines the operation output hierarchy.

Ports

Create the transformation output groups and ports in the **Ports** area.

The following figure shows the output mapping for a non-reusable Web Service Consumer transformation:



After you create output ports, map the nodes from the **Operation Output** area to the ports in the **Ports** area. When you map a node from the operation output to an output port, the location of the node appears in the **Location** column in the **Ports** area.

The Developer tool maps nodes in the first level of the operation output to output ports when you choose to map the first level of the output hierarchy. The Developer tool also creates the ports to perform the mapping. If the first level of the hierarchy contains a multi-occurring parent node with one or more multi-occurring child nodes, the Developer tool does not map the first level of the hierarchy.

You can choose to display the output ports in a hierarchy. Each child group appears under the parent group. You can also choose to view the lines that connect the nodes in the operation output to the output ports.

If the associated WSDL data object is deleted from the repository, the Developer tool retains the location of the operation nodes in the output mapping. When you show the output mapping, the **Ports** area still displays the location of the operation nodes in the **Location** column for the output ports. If you associate another WSDL with the transformation, the Developer tool checks whether each location is valid. The Developer tool clears the location of operation nodes in the **Ports** area of the output mapping if the location is no longer valid.

RELATED TOPICS:

- [“Parsing Web Service SOAP Message Overview” on page 666](#)

Rules and Guidelines to Map Nodes to Output Ports

Review the following rules when you map nodes in the operation output hierarchy to output ports:

- The operation output node and the output port must have compatible datatypes.
- You cannot map a node to more than one output port in a group.
- Each output port must have a valid location, unless the port is a pass-through port.
- If you drag a multiple-occurring child node to an empty output port, you must relate the group to other output groups. When you select a group, the Developer tool creates keys to relate the groups.
- When you drag a multiple-occurring element into a group that contains the parent element, you can configure the number of child element occurrences to include. Or, you can replace the parent group with the multiple-occurring child group in the transformation output.

Mapping the SOAP Message as XML

You can map the complete SOAP message as XML instead of returning the data to separate output ports.

When you map the SOAP message as XML, the Data Integration Service returns the complete SOAP message in one port. Do not create output ports.

To map the complete message, right-click the root group in the **Operation Output** area. Select **Map as XML**.

The Developer tool creates a string output port. The precision is 65535 bytes.

Customize View Options

You can change the operation output hierarchy to show the cookie ports, pass-through ports, and keys in the **Operation Output** area. You can also show grouping constructs that define how to order nodes.

Click the **Customize View** button in the **Operation Output** area. Enable any of the following options:

Sequence, Choice, and All

Show a line that indicates whether an element definition is sequence, choice, or all.

Nodes in an all group must all be included in the SOAP message.

Nodes in a sequence group must be in the order specified in the WSDL.

At least one node in a choice group must appear in the SOAP message.

Keys

View the keys in the **Operation Output** area. The **Operation Output** area includes keys for each group. You can add a key to an output port in the **Ports** area.

Pass-through Ports

The **Operation Output** area shows the pass-through ports. Pass-through ports are ports that pass data through the transformation without changing the data. You can project pass-through ports from the operation output to any of the Web Service Consumer transformation output groups. A pass-through port receives data one time, so the port is at the root level in the SOAP messages.

Cookie Ports

Shows the cookie port. When you configure cookie authentication, the remote web service server tracks the web service consumer users based on the cookies. When you project a web service cookie in the request message, the web service returns a cookie in the response message. You can project the cookie from the operation output to any of the Web Service Consumer transformation output groups.

Mapping the Operation Output to Output Ports

When you show the transformation output mapping, you can define output groups, define output ports, and map operation output nodes to output ports.

1. Open a Web Service Consumer transformation.
2. To show the transformation output mapping, use one of the following methods:
 - For a reusable transformation, click the **Overview** view. Choose to show the output mapping.
 - For a non-reusable transformation, click the **Ports** tab in the **Properties** view. Choose to show the output mapping.
3. To add an output group or port to the **Ports** area, use one of the following methods:

Option	Description
Drag a node	Drag a group node or a child node in the Operation Output area to an empty column in the Ports area. If the node is a group node, the Developer tool adds a group without ports.
Manually add a group or port	To add a group, click the arrow next to the New button and then click Group . To add a port, click the arrow next to the New button and then click Field .
Drag a port from another transformation	In the editor, drag a port from another transformation to the Web Service Consumer transformation.
Copy a port	Select ports from another transformation and copy them to the Ports area. To copy ports, you can use keyboard shortcuts or you can use the copy and paste buttons in the Developer tool.
Select Map first level of hierarchy	Select Map first level of hierarchy . The Developer tool maps nodes in the first level of the operation output to output ports and groups. The Developer tool also creates the output ports and groups to perform the mapping.

4. If you manually create a port or you copy a port from another transformation, click the **Location** column in the **Ports** area and choose a node from the list.
5. To clear the locations of ports, use one of the following methods:

Option	Description
Click Clear	Select one or more ports in the Ports area and click Clear .
Delete the lines that connect nodes to ports	Select one or more lines that connect the nodes in the operation output to the output ports and press Delete .

6. If the associated WSDL data object includes anyType elements, any elements, anyAttribute attributes, derived type elements, or substitution groups, choose objects in the **Operation Output** area. In the **Type** column for a node, click **Choose** and then choose one or more types, elements, or attributes from the list.

- To map the complete SOAP response message as XML, right-click the root group in the **Operation Output** area and select **Map as XML**.

Web Service Consumer Transformation Advanced Properties

The Web Service Consumer transformation advanced properties include the tracing level, generic fault ports, web services connection, and concurrent web service request messages.

You can define the following advanced properties for the Web Service Consumer transformation on the Advanced tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

SOAP Action

Overrides the SOAP action value defined in the WSDL with a constant value for the Web Service Consumer transformation.

Enable Generic SOAP Fault Handling

Returns fault messages that are not defined in the WSDL. Creates output ports in a GenericFault output group to handle fault codes and messages.

The following table describes the fault output ports for SOAP 1.1 and SOAP 1.2:

Fault Output Port for SOAP 1.1	Fault Output Port for SOAP 1.2	Description
Fault Code	Code*	Returns a fault identification code.
Fault String	Reason*	Returns an explanation of the error in a fault message.
Fault Detail	Detail	Returns custom information that the web service provider passes to the Web Service Consumer transformation in a generic fault message.
Fault Actor	Role	Returns information about the object that caused the fault to happen.
-	Node	Returns the URI of the SOAP node that generated the fault.
* The Code and Reason output ports are hierarchical.		

Note: You can expand the Code fault output port to extract the SubCode fault output port up to one level.

Enable HTTP Error Handling

Returns any HTTP error from the web service. Creates an HTTP error output port in the GenericFault output group.

Treat Fault as Error

Adds fault messages to the mapping log. When a fault occurs, the Data Integration Service increments the error count for the mapping. Disable this property to allow early selection and push-into optimization. Default is enabled.

Connection

Identifies the web services connection object to connect to the web service. Create the web services connection in the Developer tool. Edit the web services connection in the Developer tool or the Administrator tool. When you configure a web services connection, configure the endpoint URL, the type of security that the web service requires, and a connection timeout period.

The Web Service Consumer transformation connects to a web service using an endpoint URL. You can define the endpoint URL in the WSDL file, in a web services connection, or in an endpoint URL input port.

Use the following guidelines to determine when to configure a web services connection:

- Configure a connection if you want to use an endpoint URL that differs from the URL in the WSDL file and if you are not using an endpoint URL input port.
- Configure a connection if the web service you connect to requires web service security, HTTP authentication, or SSL certificates.
- Configure a connection if you want to change the default connection timeout period.

Note: You can associate a WSDL data object in the repository with a web services connection. The associated connection becomes the default connection for each Web Service Consumer transformation that you create from that WSDL.

Enable Compression

Enables coding of SOAP requests with the GZIP compression method and enables decoding of SOAP responses with GZIP or deflate.

XML Schema Validation

Validates the SOAP response message at run time. Select **Error on Invalid XML** or **No Validation**.

Sorted Input

Enables the Data Integration Service to generate output without processing all of the input data. Enable sorted input when the input data is sorted by the keys in the operation input hierarchy.

Push-into Optimization

Enables push-into optimization. Click the **Open** button in the **Push-Into Optimization** property to select filter ports that receive filter values. For each filter port, choose the output port that contains the filtered column in the web service response.

Has Side Effects

Checkbox that indicates that the web service performs any function besides returning rows. The Web Service Consumer transformation has a side effect if the web service, in addition to returning a rows, modifies an object or interacts with other objects or functions. The web service might modify a database, add to a total, raise an exception, write an email, or call other web services with side effects. Disable the **Has Side Effects** property in order to allow push-into optimization or early selection optimization. Default is enabled.

Enable Concurrency

Enables the Web Service Consumer transformation to create multiple concurrent connections to a web service so that it can send multiple web service requests in parallel. When you enable the Web Service Consumer transformation to create multiple concurrent connections to the web service, you can set the total memory consumption limit and the number of concurrent connection limits.

The following table describes the options:

Options	Description
Enable concurrency	Creates multiple concurrent connections to a web service.
Concurrency Connection Limit	The number of concurrent web service connections. Default is 20.
Total Concurrency Memory Limit (in MB)	The total memory allocation limit for all the concurrent connections. Default is 100 MB.

Web Service Error Handling

You can configure the Web Service Consumer transformation to pass SOAP faults and HTTP errors downstream in a mapping. You can increment the error count when a fault occurs. Configure web service error handling in the transformation advanced properties.

A web service returns either a response message or it returns a fault. A fault is an error. The web service can generate different faults based on the errors that occur.

The Web Service Consumer transformation can return the following types of faults:

SOAP faults

SOAP errors that the WSDL defines. Configure output error ports that return the faults in the web service response message. For a SOAP 1.1 binding, the Data Integration Service returns the fault message, fault code, fault string, and fault actor elements for the fault. For a SOAP 1.2 binding, the Data Integration Service returns the fault message, code, reason, node, and role elements for the fault.

Generic SOAP faults

The web service generates generic SOAP faults at run time. The fault elements are different for a SOAP 1.1 binding and a SOAP 1.2 binding. The WSDL does not define generic SOAP faults. Generic SOAP faults include authentication failures and SOAP request errors.

HTTP errors

The Developer tool adds the HTTP fault output port when you enable HTTP error handling in the transformation. The Data Integration Service returns HTTP errors from the web service in a single string port. An HTTP error includes an error code and a message.

If the SOAP response from the web service has XML data that is not valid, the Web Service Consumer transformation returns an error.

You can configure whether to treat SOAP faults as errors. When you enable Treat Fault as Error and a SOAP fault occurs, the Data Integration Service increments the error count for the mapping. The fault appears in the message log.

Message Compression

When you enable SOAP message compression, the Web Service Consumer transformation compresses web service request messages and receives compressed web service response messages.

The Web Service Consumer transformation encodes the SOAP request with GZip compression. The transformation accepts a response message encoded with the GZip or the deflate compression.

When the Data Integration Service receives the response from the web service, it checks the Content-Encoding HTTP header in the SOAP message and it decodes the message.

The default is no compression encoding. The web service does not compress the SOAP response.

The following table shows the headers in the request and response messages when compression is on or off:

Compression	Header
On	Content-Encoding header: GZip Accept-Encoding header: GZip, deflate
Off	Empty Content-Encoding header Empty Accept-Encoding header

Sometimes a web service encodes a response message with a default compression. The Web Service Consumer transformation decodes the message if it is GZip or deflate encoded. The Web Service Consumer transformation logs a message to the mapping log if the web service encodes the response message unexpectedly.

Enable compression in the transformation advanced properties.

Concurrency

You can enable the Web Service Consumer transformation to create multiple concurrent connections to a web service so that it can send multiple web service requests in parallel.

For example, while querying bank information, you can configure the Web Service Consumer transformation for concurrency so that multiple rows are sent in parallel. If there are 20 input rows, you can send 20 requests concurrently for faster processing.

When you enable concurrency in the Web Service Consumer transformation, you can configure the total memory consumption limit.

When you enable concurrency in the Web Service Consumer transformation, you can configure the number of concurrent web service connections.

Rules and Guidelines for Concurrency

Use the following rules and guidelines while using concurrency:

- Concurrency supports sorted input rows as multiple concurrent connections to a web service. Ordered output rows are not supported.
- Use concurrency if the data set is more than 100 rows.
- It is advisable not to increase the number of concurrent web service connections. The number of concurrent web service connections is linked to the number of sockets used by the operating system. Increasing the number of sockets is expensive.
- Use systems that have multi-core processors with a minimum of 100 MB of RAM for optimal performance while using the concurrency feature.
- Concurrency memory limit represents the memory consumed by concurrent work flows while invoking web services.
- When you enable concurrency in the Web Service Consumer transformation, you can configure the memory consumption limit. Ensure that the memory consumption is not more than the physical RAM on the server.

Best Practices for Concurrency

For optimal performance while using concurrency, adhere to the following best practices:

- Avoid changing the default values of the total concurrency memory limit and concurrency connection limit.
- Avoid using concurrency for data sets of less than 100 rows.
- Avoid pass-through ports in the mapping while using concurrency.

Filter Optimizations

Filter optimization increases performance by reducing the number of rows that pass through the mapping. The Data Integration Service can apply the early selection optimization or push-into optimization.

When the Data Integration Service applies a filter optimization method, it moves a filter as close to the source as possible in a mapping. If the Data Integration Service cannot move a filter before a transformation in a mapping, it might be able to push the filter logic into a transformation.

Enabling Early Selection Optimization with the Web Service Consumer Transformation

Enable early selection optimization for the Web Service Consumer transformation if the transformation does not have side effects and it does not treat faults as errors.

1. Open the Web Service Consumer transformation **Advanced Properties** view.
2. Clear **Treat Fault as Error**.
3. Clear **Has Side Effects**.

Push-Into Optimization with the Web Service Consumer Transformation

You can configure push-into optimization with the Web Service Consumer transformation when the transformation is in a virtual table in an SQL data service.

The mapping calls the web service to retrieve a set of data or a subset of the data based on the statements in the end-user SQL query. The end-user SQL query contains an optional filter condition.

With push-into optimization, the Web Service Consumer transformation receives the filter value in a filter port. The filter port is an unconnected input port that you identify as a filter port when you configure push-into optimization. The filter port has a default value that ensures that the web service returns all rows if the end-user query contains no filter. The filter port is not a pass-through port.

Note: The filter field must be part of the root group in the web service request.

When you configure a filter port, you identify an output port in the Web Service Consumer transformation that receives the column data from the web service response. For example, if the filter port is an input port named EmployeeID, the output port from the response might be a port named EmployeeNum. The Developer tool needs to associate the input filter port and an output port in order to push the filter logic from the virtual table read to the web service consumer request. The input ports for a web service request are usually different than the output ports from the web service response.

The filter field cannot be a pass-through port. When you configure a filter port, the default value of the port changes to the value of the filter condition, so the pass-through output port value changes. A filter based on the output pass-through port returns unexpected results.

You can push multiple filter expressions to the Web Service Consumer transformation. Each filter condition must be the following format:

```
<Field> = <Constant>
```

The filter conditions must be joined by AND. You cannot join the conditions with an OR.

Push-Into Optimization with Web Service Consumer Transformation Example

An SQL data service returns orders for all customers or it returns orders for a specific customer based on the SQL query it receives from the user.

The data service contains a logical data object with the following components:

Customer table

An Oracle database table that contains customer information.

Web Service Consumer transformation

A transformation that calls a web service to retrieve the latest orders for customers. The Web Service Consumer transformation has input ports for customerID and orderNum. The transformation has pass-through ports that contain customer data that it receives from the Customer table. The orderNum port is the filter port and is not connected. orderNum has the default value "*". When the web service receives this value in the web service request, it returns all orders.

Orders virtual table

A virtual table that receives the customer and order data from the web service. The end-user queries this table. Orders contains a customer column, orderID column, and customer and order data.

The end-user passes the following SQL query to the SQL data service:

```
SELECT * from OrdersID where customer = 23 and orderID = 56
```

The Data Integration Service splits the query to optimize the mapping. The Data Integration Service uses early selection optimization and moves the filter logic, `customer = 23`, to the Customer table read. The Data Integration Service uses push-into optimization and pushes the filter logic, `orderID = 56`, into the Web Service Consumer transformation filter port. The Web Service Consumer transformation retrieves ordersID 56 for customer 23.

Enabling Push-Into Optimization with the Web Service Consumer Transformation

Enable push-into optimization for the Web Service Consumer transformation if the transformation does not have side effects and it does not treat faults as errors.

1. Open the Web Service Consumer transformation **Advanced Properties** view.
2. Clear **Treat Fault as Error**.
3. Clear **Has Side Effects**.
4. Click the **Open** button in the **Push-Into Optimization** property.
5. Choose the filter port name in the Optimized Input dialog box. You can choose multiple filter ports.
6. Click the **Output** column.

7. For each filter port, choose the output port that contains the filtered column in the web service response.
8. Enter a default value for each filter port.

Note: You cannot configure a default value for a Web Service Consumer port unless it is a filter port.

Creating a Web Service Consumer Transformation

You can create a reusable or non-reusable Web Service Consumer transformation. Reusable transformations can exist in multiple mappings. Non-reusable transformations exist within a single mapping.

You can create Web Service Consumer transformations for a SOAP 1.1 binding and a SOAP 1.2 binding from a single WSDL object.

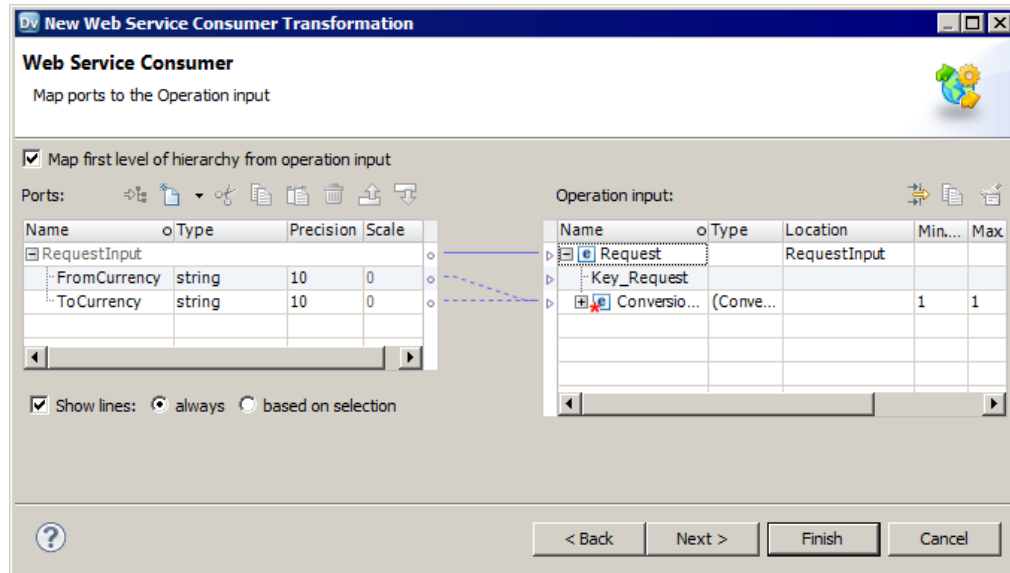
1. To create a transformation, use one of the following methods:

Option	Description
Reusable	Select a project or folder in the Object Explorer view. Click File > New > Transformation . Select the Web Service Consumer transformation and click Next .
Non-reusable	In a mapping or maplet, drag a Web Service Consumer transformation from the Transformation palette to the editor.

The **New Web Service Consumer Transformation** dialog box appears.

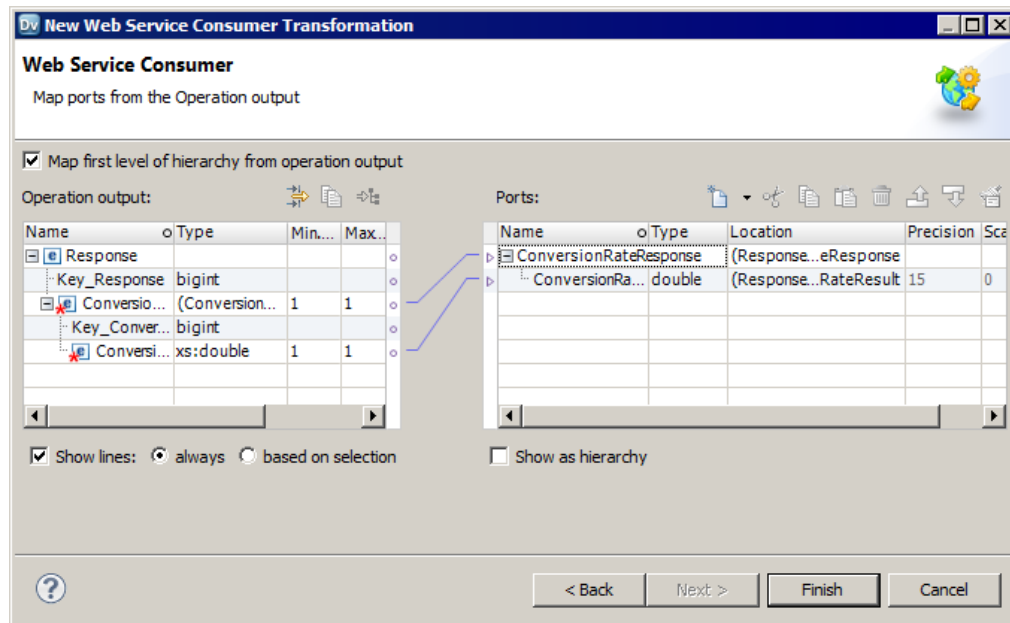
2. Browse and select a WSDL data object to define the web service request and response messages.
If the WSDL is not in the repository, you can import the WSDL from the New Web Service Consumer Transformation dialog box.
3. Browse and select an operation from the WSDL.
You can choose an operation that has a SOAP 1.1 binding or SOAP 1.2 binding.
4. Click **Next**.

The **Map Ports to the Operation Input** screen appears. The **Ports** area shows the transformation input groups and input ports. The **Operation Input** area shows the request message hierarchy.



5. Define the input groups and input ports and map the input ports to operation input nodes.
6. Click **Next**.

The **Map Ports from the Operation Output** screen appears. The **Operation Output** area shows the response message hierarchy. The **Ports** area shows the transformation output groups and output ports.



7. Define the output groups and output ports, and map the operation output nodes to the output ports.
8. Click **Finish**.
9. Click the **Advanced** view to configure the transformation properties and the web services connection.

RELATED TOPICS:

- [“Mapping Input Ports to the Operation Input” on page 650](#)
- [“Mapping the Operation Output to Output Ports” on page 654](#)
- [“Web Service Consumer Transformation Advanced Properties” on page 655](#)

Web Service Consumer Transformation Example

Your organization needs to expose order information for the RT100 product line to the sales organization. The sales team needs to query the order summary and order details on a daily basis.

Create a logical data object that exposes the daily order information in virtual tables. The read mapping contains a Web Service Consumer transformation that returns the latest RT100 orders. The Web Service Consumer transformation consumes a web service that returns the daily order summary and order detail information for the RT100 product line.

Input File

The input file is a flat file that contains the product line number.

Create a physical data object to define the input file. The file has one field, Product_Line. The field value is RT100. Define the location of the physical data object in the **Runtime Properties** view.

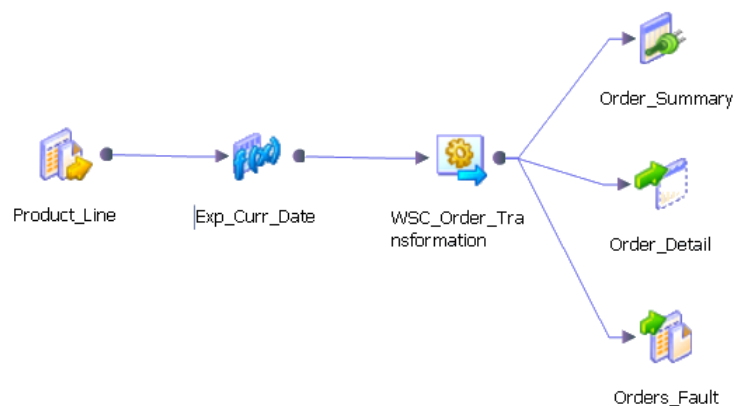
Logical Data Object Model

A business analyst in your organization creates a logical data model that describes the order summary and order detail table structures. The logical data model contains the Order_Summary and Order_Detail logical data objects.

The analyst creates a schema in a modeling tool that defines the logical data model. You import the logical data model from the schema and create the Order_Summary and Order_Detail logical data objects.

Logical Data Object Mapping

The logical data object mapping describes how to access data through the logical data object.



The read mapping contains the following objects:

Product_Line

Input flat file that contains the product line number.

Exp_Curr_Date transformation

Expression transformation that returns the current date and a primary key for the Web Service Consumer transformation root level input group.

WSC_Order transformation

Web Service Consumer transformation that consumes a web service to retrieve order information. The transformation passes the product line and current date to the web service in the request message. The transformation receives order information from the web service in the response message.

Order_Summary table

A logical data object that contains order information such as Order_No, Customer_Id, Qty, and Order_Date.

Order_Detail table

A logical data object that contains order detail information such as Order_No, Product_Id, Qty, and Status.

Orders_Fault

Output flat file that receives generic fault messages.

Web Service Consumer Transformation

The Web Service Consumer transformation receives a product line, date, and a sequence number as input. The transformation consumes the Get_Order_Info web service operation to retrieve the order information.

When you create the Web Service Consumer transformation, choose a WSDL data object that describes the request and response web service messages. A web service message contains hierarchical groups of XML elements. An element can contain other elements. Some elements occur multiple times. Create the transformation from the Order_Info WSDL object in the repository.

Configure the transformation input ports and map the ports to the operation input hierarchy. Map nodes from the operation output hierarchy to the output ports. Define the web services connection and run-time properties.

Transformation Input Mapping

When you show the input mapping on the **Ports** view, you can define input ports and map them to nodes in the operation input.

The transformation **Ports** area has a root group and an Order group. The root group is the Request input group. Add one port to the Request input group to represent the primary key.

The Order group has the **Select_Date** and **Select_Product_Line** input ports.

Map the input ports to the **Order_Date** and **Product_Line** nodes in the **Operation Input** area.

The **Operation Input** area defines the request message that the Web Service Consumer transformation passes to the web service. The nodes appear in the **Operation Input** area by default.

Transformation Output Mapping

When you show the output mapping on the **Ports** view, you can define the output ports by mapping nodes from the operation output to the transformation output groups.

The web service returns the following hierarchy in a web service response message:

```
Response
  Orders
    Order
      Key_Order
      Order_ID
      Order_Date
      Customer_ID
      Total_Qty
      Order_Details
        Order_Detail
          Product_ID
          Description
          Qty
          Status
```

The web service returns multiple orders. Order is a multiple-occurring node in the Orders level. For each order, the web service can return multiple order details. Order_Detail is a multiple-occurring node in the Order_Details level.

Note: The Developer tool adds the Key_Order node in the user interface. You can map the key to output groups to define relationships between groups. For this example, the Order_ID is the primary key in Order, and it is the foreign key in Order_Details.

Create the following output groups in the **Ports** area:

```
Order
  Order_ID
  Order_Date
  Customer_ID
  Total_Qty

Order_Detail
  Order_ID
  Product_ID
  Description
  Qty
  Status
```

The Data Integration Service writes a row from the Order group whenever the value of Order_ID changes.

The Data Integration Service writes a row from the Order_Detail group whenever the values of Order_ID and Product_ID change.

Transformation Advanced Properties

Configure the following advanced properties for the Web Service Consumer transformation:

Enable Generic SOAP Fault Handling

Adds output ports that receive SOAP fault messages.

Connection

Choose a web services connection to access the web service.

Enable Compression

The Web Service Consumer transformation compresses the web messages with GZIP.

CHAPTER 48

Parsing Web Service SOAP Messages

This chapter includes the following topics:

- [Parsing Web Service SOAP Message Overview, 666](#)
- [Transformation User Interface, 667](#)
- [Multiple-Occurring Output Configuration, 668](#)
- [Parsing anyType Elements, 670](#)
- [Parsing Derived Types, 671](#)
- [Parsing QName Elements, 672](#)
- [Parsing Substitution Groups, 672](#)
- [Parsing XML Constructs in SOAP Messages, 672](#)

Parsing Web Service SOAP Message Overview

The Data Integration Service generates row data when it parses a SOAP message in a web service transformation.

The web service Input transformation and the Web Service Consumer transformation are web service transformations that parse SOAP messages.

To configure a transformation to parse a SOAP message, create output ports in a structure similar to the SOAP message hierarchy. Map the nodes in the SOAP message hierarchy to the ports.

You can configure normalized groups of output ports, denormalized groups, and pivoted groups of ports. When the SOAP message contains derived types, anyType elements, or substitution groups, you can configure different output groups based on what types might occur in the SOAP message instance.

RELATED TOPICS:

- [“Web Service Consumer Transformation Output Mapping” on page 652](#)

Transformation User Interface

The Web Service Consumer transformation and the web services Input transformation provide a user interface that you can use to map data from the SOAP message to the transformation output ports.

The following figure shows a mapping between SOAP 1.1 message nodes and output ports in a Web Service Consumer transformation:

The screenshot displays the Transformation User Interface with two main panes. The left pane, titled 'Ports', shows a tree view of the SOAP message hierarchy. The right pane, titled 'Ports', shows a table of output ports. Blue arrows indicate the mapping between the two panes.

Left Pane (Ports):

Name	Type	Min...	Max...	Description
Response				
Key_Response	bigint			
Response	(Response)	1	1	
Key_Company	bigint			
Company	(Company)	1	1	
mySessionHeader	(mySessionH...	1	1	
Name	xs:string	1	1	
Departments	(Departments)	1	1	
Key_Departments	bigint			
Department	(Department)	0	Unb...	
Key_Department	bigint			
DeptName	xs:string	1	1	
Employees	(Employees)	1	1	
SessionHeader	tns:SessionH...	0	1	
Fault (fault1)				
Passthrough				

Right Pane (Ports):

Name	Type	Location	Precision	Scale
Company		(Response)tn.../tns:Company		
Name	string	(Response)tn...any/tns:Name 10	10	0
Fault1		(Fault)		
faultcode	string	(Fault)/faultcode	10	0
faultstring	string	(Fault)/faultstring	10	0
faultactor	string	(Fault)/faultactor	10	0

Operation Area

The Operation area contains the SOAP message hierarchy. Complex nodes or multiple-occurring nodes define hierarchy levels in the structure. The Developer tool adds keys to the levels that define the parent-child relationships between them.

In the preceding figure, the SOAP message hierarchy has the following levels:

Response or Request

Level that represents the root of the response or request message.

Company

Top level of the request data.

Departments

Multiple-occurring departments within the company.

Employees

Employee is a complex element within a department.

Fault Group

Fault message group that receives error messages.

Ports Area

You can map data from levels of the SOAP message to output ports. Each group of output ports can be related to other output groups with primary foreign-key relationships.

In the preceding figure, the transformation has groups of output ports that correspond to the groups of nodes in the SOAP message.

Multiple-Occurring Output Configuration

When an Input transformation or Web Service Consumer transformation returns multiple-occurring data, you can configure the output ports in different configurations.

You can configure normalized output data, pivoted output data, or denormalized output data.

For example, a SOAP message contains Departments and Employees complex elements. Each department contains multiple employees. Departments is the parent of Employees.

The SOAP message contains the following element hierarchy:

```
Departments
  Department_ID
  Department_Name
  Employees
    Employee_ID
    Employee_Name
```

Normalized Relational Output

When you create normalized output data, the data values do not repeat in an output group. You create a one-to-one relationship between the hierarchy levels in the SOAP message and the output groups of ports.

When the SOAP message contains a Departments parent hierarchy level and an Employees child hierarchy level, you might create the following groups of ports:

```
Departments
  Department_Key
  Department_ID
  Department_Name

Employees
  Department_Key
  Employee_ID
  Employee_Name
```

The Department_Key is a generated key that relates the Employees output group to a Department group.

Generated Keys

When you add an output group, the Developer tool relates the output group to another output group with a generated key. The Developer tool adds a bigint key to the parent group and to the child group. At run time, the Data Integration Service creates the key values for the generated keys.

Example

The SOAP hierarchy has the following nodes:

```
Departments
  Dept_Key
  Dept_Num
  Dept_Name

  Employees
```

```
Dept_FK
Employee_Num
Employee_Name
```

When you create an output group of ports for Departments, you map the Departments node to an empty field in the Ports area. The Developer tool creates the following output group:

```
Departments
  Dept_Num
  Dept_Name
```

When you map the Employees node to an empty field in the Ports area, the Developer tool prompts you to relate the Employees group to the Departments group. You can relate the Employees group to more than one group. The Developer tool adds a key to the each group.

The Developer tool creates the following groups and generated keys:

```
Departments
  Key_Departments
  Dept_Num
  Dept_Name

Employees
  Key_Departments
  Employee_Num
  Employee_Name
```

Note: You do not have to map nodes to the generated keys. The Data Integration Service creates the key values at run time.

The Developer tool can create generated keys to multiple levels in one output group. The Employees group might contain the following ports:

```
Employees
  Key_Employees
  Key_Departments
  Key_Managers
  Employee_Num
  Employee_Name
```

Key_Departments and Key_Managers are the generated keys that point to parent groups. Key_Employees is a generated key for the Employees group. Key_Employees appears when you relate a child group to the Employees group.

Denormalized Relational Output

You can denormalize relational output. When you denormalize the output data, the element values from the parent group repeat for each child element.

To denormalize output data, map nodes from the parent hierarchy level to the child group of output ports.

The following example shows the Department_ID and the Department_Name in the Employees output group:

```
Employees
  Department_ID
  Department_Name
  Employee_ID
  Employee_Name
```

Department_ID and Department_Name repeat for each employee in the department:

Department_ID	Department_Name	Employee_ID	Employee_Name
100	Accounting	56500	Kathy Jones
100	Accounting	56501	Tom Lyons
100	Accounting	56509	Bob Smith

Pivoted Relational Output

You can include a specific number of multiple-occurring elements in an output group.

To pivot multiple-occurring elements, map the multiple-occurring child element to the parent group of output ports. The Developer tool prompts you to define the number of child elements to include in the parent.

The following example shows two instances of Employee_ID in the Departments parent group:

```
Departments
  Department_ID
  Department_Name
  Employee_ID1
  Employee_ID2
```

Parsing anyType Elements

The anyType element represents a choice of all global types in a WSDL or schema. When you map nodes to ports in the Developer tool, you choose which types to appear in the SOAP message for the anyType element. You must replace an anyType element in the SOAP message with a complex type or xs:string. Create groups of ports for each type that you choose.

You must choose a type to map data to output ports. If the WSDL or schema does not contain a global type, the Developer tool replaces the anyType element with xs:string.

To choose an element type in the Operation area, click **Choose** in the **Type** column for the anyType element. A list of available complex types and xs:string appears.

When you replace an anyType element with derived types, the Data Integration Service populates elements for one type at a time. The SOAP message does not contain data for the base type and the derived type at the same time.

Derived Types Example

The WSDL contains an anyType element. You replace the element with AddressType and a derived type called USAddressType. The SOAP message hierarchy has the following groups:

```
Address:AddressType (base type)
  Address: AddressType
    Street
    City

Address:USAddressType (derived type)
  Street
  City
  State
  ZipCode
```

The SOAP message contains the following data:

```
<address xsi:type="AddressType">
  <street>1002 Mission St.</street>
  <city>san jose</city>
</address>

<address xsi:type="USAddressType">
  <street>234 Fremont Blvd</street>
  <city>Fremont</city>
  <zip>94556</zip>
  <state>CA</state>
```

```
</address>
```

The Data Integration Service returns one row for xsi: AddressType:

Street	City
1002 Mission St.	San Jose

The Data Integration Service returns one row for the derived type xsi: USAddressType:

Street	City	State	Zip
234 Fremont Blvd.	Sunnyvale	CA	94556

The Data Integration Service does not populate the AddressType if the type is xsi: USAddressType.

Parsing Derived Types

You can parse SOAP messages that contain derived types. When you define the ports that receive the data from the SOAP message, choose which types might appear in a SOAP message. The elements in the types you choose determine the ports that you need to create.

For example, the WSDL might contain an AddressType and a derived type called USAddressType. You can create the following groups in the Developer tool Operation area:

```
Address
  Address: AddressType
    Street
    City

  Address:USAddressType
    Street
    City
    State
    ZipCode
```

The SOAP message might contain the following data:

```
<address>
<street>1002 Mission St.</street>
<city>san jose</city>
</address>

<address xsi:type="USAddressType">
<street>234 Fremont Blvd</street>
<city>Fremont</city>
<zip>94556</zip>
<state>CA</state>
</address>

<address xsi:type="USAddressType">
<street>100 Cardinal Way</street>
<city>Redwood City</city>
<zip>94536</zip>
<state>CA</state>
</address>

<address>
<street>100 El Camino Real</street>
<city>Sunnyvale</city>
</address>
```

The Data Integration Service returns the following rows for the base type, Address:

Street	City
1002 Mission St.	San Jose
234 Fremont Blvd	Sunnyvale
100 Cardinal Way	Redwood City
100 El Camino Real	Sunnyvale

The Data Integration Service returns the following rows for the derived type, USAddress:

Street	City	State	Zip
234 Fremont Blvd.	Sunnyvale	CA	94556
100 Cardinal Way	Redwood City	CA	94536

The Data Integration Service returns all addresses in the base type. The Data Integration Service returns US Addresses in the derived type. The derived type includes the Street and City elements that the USAddressType inherits from the base type.

Parsing QName Elements

When Data Integration Service parses QName elements in the SOAP message, it updates QName values that belong to the namespace of the schema to use the namespace prefix defined in the schema. Otherwise, Data Integration Service does not update the value of the element.

For example, the schema has the namespace prefix `tns` defined for namespace `"http://user/test"`. The SOAP message has namespace prefix `mytns` defined for the same namespace. When the Data Integration Service parses QName value `mytns:myelement` it changes the value to `tns:myElement`.

When Data Integration Service generates QName elements in the SOAP message, it does not update the value of the element.

Parsing Substitution Groups

A substitution group replaces one element with another element from the same group. Substitution groups are similar to derived types, except that each element definition includes a substitution group name.

You can configure an output group of ports that receives elements from a specific type in a substitution group. You can create a different output group of ports that receives an element from another type in the substitution group.

Parsing XML Constructs in SOAP Messages

A SOAP message might contain XML constructs such as choice, list, and union elements.

Web service transformations can parse SOAP messages with these constructs with some limitations.

Choice Element

A choice element restricts a child element to one of the elements in the <choice> declaration.

The following text shows a person element that is an employee or a contractor:

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="contractor" type="contractor"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

You can map choice elements using the following methods:

- Create output ports for each choice element in an output group. Some elements will have null values in the output row.
- Create an output group for each choice. For the above example, create an employee group and a contractor group. The Data Integration Service generates a row based on which element appears in the SOAP message.

List Element

A list is an XML element that can contain multiple simple type values, such as "Monday Tuesday Wednesday".

The Data Integration Service can return a list as a string value. When the SOAP message contains a list, you cannot map items from the list to separate output rows. You can configure an Expression transformation to separate the elements in the list if you need them separated in a mapping.

Union Element

The union element is a simple type that is a union of more than one type.

The following text shows an element Size that is a union of two simple types, size_no and size_string:

```
<xs:element name="Size">
  <xs:simpleType>
    <xs:union memberTypes="size_no size_string" />
  </xs:simpleType>
</xs:element>
```

To map the size to an output port, create one port for the size. Configure the output port as a string. You can configure another transformation in the mapping to convert the data to another type.

CHAPTER 49

Generating Web Service SOAP Messages

This chapter includes the following topics:

- [Generating Web Service SOAP Messages Overview, 674](#)
- [Transformation User Interface, 675](#)
- [Port and Hierarchy Level Relationships , 676](#)
- [Keys, 677](#)
- [Map Ports, 678](#)
- [Pivoting Multiple-Occurring Ports , 680](#)
- [Map Denormalized Data, 682](#)
- [Derived Types and Element Substitution, 683](#)
- [Generating XML Constructs in SOAP Messages, 684](#)

Generating Web Service SOAP Messages Overview

The Data Integration Service generates XML data from groups of input data when it generates a SOAP message. When you create a Web Service Consumer transformation, a web service Output transformation, or a Fault transformation, you configure which input ports to map to the SOAP message hierarchy.

To configure a transformation to generate a SOAP message, create groups of input ports and map each group to a group in the SOAP message hierarchy. A WSDL or schema defines the structure of the SOAP message.

You can configure groups of data in the SOAP message from denormalized input data. You can also pivot multiple-occurring input data to multiple-occurring nodes in the SOAP message.

You can map data to derived types, anyType elements, or substitution groups in a SOAP message. You must choose which types can occur in the SOAP message when you define a transformation. The types that you choose determine the input ports that you need to create.

When you view the SOAP message hierarchy in the Developer tool, the hierarchy contains keys. The keys do not appear in the SOAP message. The Data Integration Service uses keys to define parent-child relationships between groups in the SOAP message. To configure key values, map input data to keys in the SOAP message.

RELATED TOPICS:

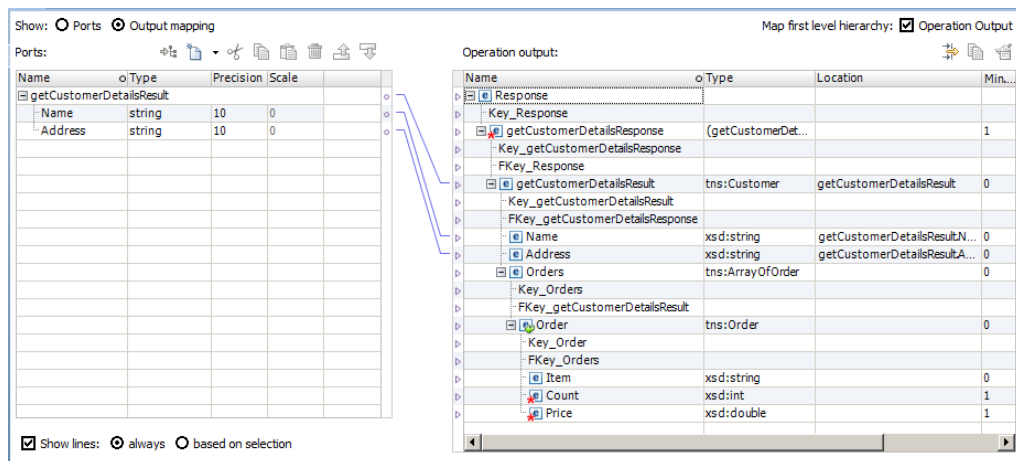
- [“Web Service Consumer Transformation Input Mapping” on page 649](#)

Transformation User Interface

The web services Output transformation, the Fault transformation, and the Web Service Consumer transformation contain a user interface that you use to configure the SOAP message.

To configure a transformation to generate a SOAP message, create input ports in a structure similar to the SOAP message hierarchy. The WSDL or schema determines the structure of the hierarchy. Map each input port to a node in the SOAP message.

The following figure shows a mapping between the input ports and the SOAP message nodes in a web service Output transformation:



Input Ports Area

Create groups of input ports in the **Input Ports** area. Include input ports for each level in the SOAP message hierarchy that you need to map.

You must create a Response or Request input group and the child groups that receive the data.

When you create input port groups, define a primary key in each parent group. Define a foreign key in each child group. The foreign key relates the group to a parent group.

You do not have to define keys for the Response level or the WSDL root level unless you are passing data at the WSDL root level. For example, the root level might contain HTTP headers.

You might create groups of ports similar to the following groups for customers and orders:

```
Response
  Response_Key

Customer_Details_Root
  Key_Cust_Det
  FK_Response_Key

Customer
  Customer_ID
```

```

    FK_Cust_Det
    Name
    Address

    Orders
    Order_Num
    FK_Cust_ID

    Order_Items
    Order_Num
    Item
    Count
    Price

```

Operation Area

The **Operation** area shows the elements in the SOAP message hierarchy as defined by the WSDL or schema. The SOAP message does not have to contain all the elements from the WSDL or schema. The message contains the data that you map from the input ports.

Multiple-occurring nodes and complex nodes define hierarchy levels in the SOAP message structure. The Developer tool adds keys to the levels to create parent-child relationships between them. All levels in the hierarchy, except for the leaf levels, have a primary key. Each child level has a foreign key to a parent level. The keys that appear in the SOAP message hierarchy do not appear in a SOAP message instance. The Data Integration Service requires values in the keys to relate levels of the data when it generates the SOAP message.

The **Location** column contains the group name and input port that contains the data for an element in the SOAP message. The **Location** column is empty until you map an input port to the node.

In the preceding figure, the SOAP message contains a single instance of customer details and orders. The Orders group contains a multiple-occurring element called Order. The SOAP message hierarchy has the following levels related by key:

```

Response
  GetCustomerDetailsResponse
    GetCustomerDetailsResult
      Orders
        Order

```

The Response level represents the root of the response message. The Data Integration Service requires this level to attach headers to the SOAP message.

The GetCustomerDetailsResponse level is the root of the message.

Port and Hierarchy Level Relationships

When you map input ports to the SOAP message hierarchy, you maintain a relationship between an input group and a SOAP message hierarchy level. For example, you might have two input groups, Department and Employee.

The Department input group receives the following rows:

Dept_num	Name	Location
101	HR	New York
102	Product	California

The Employee input group receives the following rows:

Dept_num	Employee
101	Alice
101	Bob
102	Carol
102	Dave

Map the department number in the Employee group as a foreign key that establishes the relationship between the Department and the Employee group. The department number occurs in the department hierarchy level, but not the employee level.

The SOAP message contains the following XML structure:

```
<department>
  <dept_num>101</dept_num>
  <name>HR</name>
  <location>New York</location>

  <employee>
    <name>Alice</name>
  </employee>

  <employee>
    <name>Bob</name>
  </employee>
</department>

<department>
  <dept_num>102</dept_num>
  <name>Product</name>
  <location>California</location>

  <employee>
    <name>Carol</name>
  </employee>

  <employee>
    <name>Dave</name>
  </employee>
</department>
```

Keys

A SOAP message hierarchy includes keys. The Data Integration Service requires key values to construct the XML hierarchy in the SOAP message.

You must map input port data to the keys in the SOAP message hierarchy. Map data to the keys in each level that you are providing data. When you have a multiple-occurring node, you need to relate the node to a parent.

The keys appear in the SOAP message without types. Any port that you map to a key must be a string, integer, or bigint datatype. The primary key in the parent group and the foreign key in each child group must have the same datatype, precision, and scale. You can map generated keys to the SOAP message keys.

You can map a port to a node and to a key in the same hierarchy level. For example, you map Employee_ID to a node in the SOAP message and you map it to a key in the Employee level.

If two group nodes in the hierarchy have a parent-child relationship, complete the following tasks:

- Map a port to the primary key in the parent node group.

- Map a port to the foreign key in the child node group.

You can also map primary keys to input ports to remove rows with a primary key of null or with duplicate primary keys.

You can create a composite key in a SOAP message by mapping multiple ports to the same key. Use composite keys when you need to denormalize data and maintain unique keys for some multiple-occurring combinations of values. You can create composite keys that contain strings, bigint, or integer values.

Note: You can include an Expression transformation in the operation mapping to generate key values.

Composite Key Example

Configure a unique division-department key from the following groups of ports:

```
Company
  Company_Num
  Company_Name

Division
  Company_Num
  Division_Num
  Division_Name

Department
  Division_Num
  Dept_Num
  Dept_Name
  Location
```

The Dept_Num is unique within a division, but Dept_Num is not unique for all divisions in the Company.

You might configure a Department group that contains the division and the department information. Configure the division number and the department number as part of the composite key:

```
Department
  Division_Num + Dept_Num (key)
  Dept_Name
  Location
```

The order that you map the ports determines the key value.

Map Ports

After you create input ports, map each input port to the SOAP message hierarchy. The location of the port appears next to the node in the **Operation** area.

You can map ports to the following types of nodes:

Atomic node

A simple element or an attribute that has no children and is not divisible.

Multiple-occurring atomic node

A simple element or an attribute that occurs multiple times at the same location in the hierarchy.

Complex node

An element that contains other elements.

If the parent node does not have a location, the parent node receives the input group name as the location. When the parent node has a location, each node in the hierarchy level must have an output location from the same location.

You can map an input group name to a parent node in a hierarchy level. The Developer tool updates the location field for the parent node in the hierarchy. The Developer tool does not update the child nodes that belong to the group in the hierarchy. When you map input ports to the child nodes, each input port location must be the same location as the location parent node.

After you map an input group to a hierarchy level, you can change it later. You can click **Clear** or you can delete the lines between the Ports and the Operation area. To delete the lines, drag the pointer of the lines to select them. Click **Delete**.

Map a Port

When you map a port to a node in the SOAP message, the Developer tool provides different results based on the type of node to which you map the port.

The following table describes the results when you map a single port to different target nodes in the **Operation** area:

Target Node	Results
Atomic node	When you map a single port to a node and the parent node does not have a location, the node receives the location of the port. The parent node location receives the location of the input group for the single port. When you map a single port to a node and the parent node already has a location, you can change the location for the parent node and clear the location for the other child nodes in the same level. The hierarchy level location changes to the group name of the port.
Multiple-occurring atomic node or the primary key of the multiple-occurring atomic node	When you map a single port to the multiple-occurring atomic node, the Developer tool sets the location for the atomic node to the group of the selected port.
Complex node	When you map a single port to a complex node, the Developer tool sets the location of the complex node to the location of the group that contains the port. The Developer tool prompts you for the single occurring atomic node to assign the port to. If all single-occurring atomic nodes have a location, you cannot map the complex node.

Map a Group

When you map an input group to a node in the SOAP message, the Developer tool provides different results based on the type of node that you map the port to.

The following table describes the results when you map a group to a node in the **Operation** area:

Target Node	Results
Atomic node	You cannot map a group to an atomic node.
Multiple-occurring atomic node	You are prompted to choose a port in the input group to update the location for the node and primary key.
Multiple-occurring complex node	The Developer tool sets the location for the complex node to the location of the group.

Map Multiple Ports

When you map multiple ports to a node in the SOAP message, the Developer tool provides different results based on the type of node you map the ports to. You can map multiple ports at the same time if you map them from the same group.

The following table describes the results for the node when you map multiple ports to nodes:

Target Node	Results
Single atomic node	When you map multiple ports to a single node, you update the location for more than one single atomic node in the Operation area. If the hierarchy does not have enough nodes in the level to update, the Developer tool maps ports just for the available nodes.
Multiple-occurring atomic node	When you map multiple ports to multiple-occurring atomic node, you pivot the ports into multiple occurrences of the node. The Developer tool creates instances of the node based on how many ports you map. A message appears that describes the number of ports that you projected.
Multiple-occurring complex node	When you map multiple ports to a complex node, you must select which single-occurring nodes atomic nodes to update. You pivot the ports into multiple occurrences of the node. The Developer tool creates instances of the node based on how many ports you map.

Pivoting Multiple-Occurring Ports

You can map multiple input ports to a multiple-occurring node in the SOAP message. The Developer tool pivots the input data into multiple nodes in the SOAP message.

To change the number of elements to pivot, choose **Override existing pivoting** in the **Map Options** dialog box.

If you remove one of the pivoted port instances from the **Ports** area, the Developer tool removes all instances from the **Operation** area.

Pivoting Example

An input group might have the following rows:

Num	Name	Location	emp_name1	emp_name2	emp_name3
101	HR	New York	Alice	Tom	Bob
102	Product	California	Carol	Tim	Dave

Each row contains a department number and three employees names.

Employee is a multiple-occurring node in the SOAP message hierarchy. You can map all instances of Employee from the input row to the SOAP message hierarchy. Select all occurrences of Employee. Click **Map**. The **Map Options** dialog box prompts you to choose a node from the list.

The Developer tool changes the Employee node to include the multiple name nodes in the SOAP message hierarchy:

```
Department
  num
  name
  location
  Employee (unbounded)
    emp_name1
    emp_name2
    emp_name3
```

The SOAP message returns the following hierarchy:

```
<department>
  <num>101</num>
  <name>HR</name>
  <location>New York</location>
  <employee>
    <emp_name>Alice</name>
  </employee>
  <employee>
    <emp_name>Tom</name>
  </employee>
  <employee>
    <emp_name>Bob</name>
  </employee>
</department>

<department>
  <num>102</num>
  <name>Product</name>
  <location>California</location>
  <employee>
    <emp_name>Carol</name>
  </employee>
  <employee>
    <emp_name>Tim</name>
  </employee>
  <employee>
    <emp_name>Dave</name>
  </employee>
</department>
```

Map Denormalized Data

You can map denormalized data and pass it to normalized nodes in a SOAP message.

When you map denormalized data, you pass data from one input group to multiple nodes in the SOAP message hierarchy. You can create group relationships in the SOAP message similar to the following types of relationships:

Linear Node Relationship

Node A is parent to Node B. Node B is a parent to Node C. Node C is the parent of Node D.

Hierarchical Node Relationship

Node A is a parent to Node B. Node A is also a parent to Node C. Node B and Node C are not related.

The following table shows input rows that contain denormalized division and department data:

Division	Dept_Num	Dept_Name	Phone	Employee_Num	Employee_Name
01	100	Accounting	3580	2110	Amir
01	100	Accounting	3580	2113	Robert
01	101	Engineering	3582	2114	Stan
01	101	Engineering	3582	2115	Jim
02	102	Facilities	3583	2116	Jose

The input data contains unique employee numbers and names. The division and department data repeat for each employee in the same department and division.

Linear Group Relationship

When you configure ports, you can configure a separate group for Division, Department, and Employee. Division is a parent of Department, and Department is the parent of Employee. You can configure groups in the following linear structure:

```
Division
  Division_Key
  Division_Num
  Division_Name

  Department
    Department_Key
    Division_FKey
    Dept_Num
    Dept_Name
    Phone

    Employee
      Department_Fkey
      Employee_Num
      Employee_Name
```

The SOAP message contains unique instances of Division and Department although Division_Num and Dept_Num repeat in the input data. Define the Division_Num as the primary key in the Division group. Define Dept_Num as the primary key in the Department group.

Hierarchical Group Relationship

You can create a group hierarchy that contains the Division parent group and Department and Employee child groups. Department and Employee do not have a primary key-foreign key relationship. Department and Employee are children of Division. You can configure the groups in the following structure:

```
Division
  Division_Key
  Division_Num
```

```

Division_Name

Department
  Division_FKey
  Dept_Num
  Dept_Name

Employee
  Division_FKey
  Employee_Num
  Employee_Name

```

Derived Types and Element Substitution

You can map input ports to derived complex types, anyType elements, and substitution groups in a SOAP message. The SOAP message can include elements for the base type and the derived types.

In a type relationship, the base type is the type from which you derive another type. A derived type inherits elements from the base type. An extended complex type is a derived type that inherits elements from a base type and includes additional elements. A restricted complex type is a derived type that restricts some elements from the base type.

Generating Derived Types

When a WSDL or schema includes derived types, you must choose the types that you want to include in the SOAP message.

For example, the WSDL defines a base type AddressType. The WSDL also contains a USAddressType and UKAddressType that are the derived AddressTypes.

Each type contains the following elements:

- AddressType: street, city
- USAddressType (extends AddressType): state, zipCode
- UKAddressType (extends AddressType): postalCode, country

When you choose a USAddressType in the Operation area, the Developer tool creates a group for the USAddressType element in the SOAP message. The group includes the street and city from the base address and the state and zipCode for the USAddress. Derived types that extend base types always include the elements from the base type.

If you choose all of the available derived types for the SOAP message, the Developer tool creates groups similar to the following in the SOAP hierarchy:

```

Address
  Address: Address
    Street
    City

Address:USAddressType
  Street
  City
  State
  ZipCode

Address: UKAddressType
  Street
  City

```

PostalCode
Country

You need to define input port groups for Address, USAddress, and UKAddress.

Generating anyType Elements and Attributes

Some schema elements and attributes allow any type of data in a SOAP message.

The anytype element represents a choice of all globally known types. Before you map a port to an anyType element in a SOAP message, choose an available complex type or xs:string. If the WSDL or schema does not contain a complex type, the Developer tool replaces the anyType element type with xs:string.

To choose an element type in the Operation area, click **Choose** in the **Type** column for the anyType element. A list of available complex types and xs:string appears.

The following element and attributes allow any type of data:

anyType element

Allows an element to be any datatype in the associated XML file.

anySimpleType element

Allows an element to be any simpleType in the associated XML file.

ANY content element

Allows an element to be any global element defined in the schema.

anyAttribute attribute

Allows an element to be any attribute already defined in the schema.

Generating Substitution Groups

Use substitution groups to replace one element with another in a SOAP message. Substitution groups work similarly to derived types, except that the element definitions include a substitution group name.

For example, you might have a base type Address and the derived types USAddress and UKAddress:

```
xs:element name="Address" type="xs:string"/>
<xs:element name="USAddress" substitutionGroup="Address"/>
<xs:element name="UKAddress" substitutionGroup="Address"/>
```

When you configure the SOAP message hierarchy, you can choose which element to substitute for Address in the SOAP message.

Generating XML Constructs in SOAP Messages

A WSDL or schema can contain choice, list, or union elements. Web service transformations can generate SOAP messages that contain these elements.

Choice Element

A choice element restricts a child element to one of the elements in the <choice> declaration.

To map ports to a SOAP message that contains choice elements, create one input group that includes all the elements in the choice construct. For example, an item description might be a dimension or a weight:

```
item: description, choice {dimension, weight}
```

When the description is a dimension, the description is a complex type that contains, length, width, and height.

When the description is a weight, the description is a simple string type.

The input data has the following columns and rows:

description	length	width	height	weight
box	20cm	18cm	15cm	NULL
coffee	NULL	NULL	NULL	500g

The SOAP message contains an Item group that contains dimensions or weight descriptions:

```
Item
  Description
    Dimension
      Length
      Width
      Height
    Weight
```

The NULL values in the input data become missing elements in the XML output.

The SOAP message contains the following data:

```
<item>
  <desc>box</desc>
  <dimension>
    <length>20cm</length>
    <width>18cm</width>
    <height>15cm</height>
  </dimension>
</item>

<item>
  <desc>coffee</desc>
  <weight>500g</weight>
</item>
```

List Element

A list is an XML element that can contain multiple simple type values in the same element or attribute. The Data Integration Service can process a list in the input data if the list is represented as consolidated string of data.

If each item in the list is a separate element, such as ClassDates1, ClassDates2, and ClassDates3, the Data Integration Service cannot process the items as a list. You can use an Expression transformation to combine them into a string if you need to return a list in a SOAP message.

The following input rows contain a list element called ClassDates that contains days of the week:

CourseID	Name	ClassDates
Math 1	Beginning Algebra	Mon Wed Fri
History 1	World History	Tue Thu

The Data Integration Service can return a SOAP message with the following XML structure:

```
<class>
  <courseId>Math 1</courseId>
  <name>Beginning Algebra</name>
  <classDates>Mon Wed Fri</classDates>
</class>
<class>
  <courseId>History 1</courseId>
  <name>World History</name>
  <classDates>Tue Thu</classDates>
</class>
```

Union Element

The union element is a simple type that is a union of more than one type. When a SOAP message contains a union element, you must map a single input port that contains the data in a string.

For example, the SOAP message contains an element called size. Size is a union of integer and string:

```
<xs:element name="size">
  <xs:simpleType>
    <xs:union memberTypes="size_no size_string" />
  </xs:simpleType>
</xs:element>
```

The input rows contain items with a description and size. An item can have a numeric size, such as 42. Or, an item can have a size that is a string value, such as large, medium, or small.

The following table shows input rows with a numeric size and a string size:

Desc	Size
shoes	42
shirt	large

Create one port for the item size. Map the port as a string. The SOAP message contains the following elements:

```
<item>
  <desc>shoes</desc>
  <size>42</size>
</item>

<item>
  <desc>shirt</desc>
  <size>large</size>
</item>
```

CHAPTER 50

Weighted Average Transformation

This chapter includes the following topics:

- [Weighted Average Transformation Overview, 687](#)
- [Configuring a Weighted Average Transformation, 687](#)
- [Weighted Match Scores Example, 688](#)
- [Weighted Average Transformation Advanced Properties, 688](#)
- [Weighted Average Transformation in a Non-native Environment, 689](#)

Weighted Average Transformation Overview

The Weighted Average transformation is a passive transformation that reads match scores from multiple matching operations and produces single match score.

You can apply a numerical weight to each score that enters the Weighted Average Transformation. A weight is a value between zero and 1. You can edit the weight applied to each input score to increase or decrease its contribution to the output score. Apply weights that reflect the relative importance of each data column in the duplicate analysis.

Use the Weighted Average transformation when you add Comparison transformations to a mapping or mapplet.

Note: You also can assign weights in a Match transformation. Use the Match transformation to configure matching strategies and assign weights in a single transformation. You can embed a matching mapplet in a Match transformation.

Configuring a Weighted Average Transformation

You use the Weighted Average transformation to adjust the overall match score that a mapping generates for a series of match analysis operations. You edit the relative weights of each input port to reflect the priorities

of the data comparisons that you defined for the source data set. Each input port on the Weighted Average transformation represents a match score output from a Comparison transformation strategy.

The following steps describe the process to configure a non-reusable Weighted Average transformation in a mapplet or mapping that uses Comparison transformations.

1. Open a match analysis mapplet or mapping, and add a Weighted Average transformation downstream of the Comparison transformations.
2. Connect the score output from a Comparison transformation to a Weighted Average input port. Repeat this step for other Comparison transformations in the mapplet or mapping.
3. Select the **Ports** tab on the Weighted Average transformation.
4. Double-click the **Weight** field for each input, and enter a weight value between 0.001 and 1. The weight value should reflect the relative importance of the input score when compared to other inputs on the transformation.
5. Save the mapplet or mapping.

Weighted Match Scores Example

You create a match analysis mapping to determine the number of duplicate customer names in a customer database. You add two Comparison transformations to generate match scores for the `ZIP code` and `Surname` columns in the data set.

Many records have matching ZIP codes, but a much smaller number of records have matching surnames. When averaging these match scores, you need to emphasize the importance of the more unique matches.

To emphasize the importance of the surname match scores, you apply a higher weight to the `Surname` match score.

For example, you set the **Weight** value of the `Surname` score input to 0.8, and you set the value of the **Weight** value of the `ZIP Code` score input to 0.4.

Weighted Average Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Weighted Average transformation.

You can configure tracing levels for logs.

Configure the following property on the **Advanced** tab:

Tracing Level

Amount of detail that appears in the log for this transformation. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

Weighted Average Transformation in a Non-native Environment

The Weighted Average transformation processing in a non-native environment depends on the engine that runs the transformation.

Consider the support for the following non-native run-time engines:

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 51

Window Transformation

Use the Window transformation when you want to accumulate streamed data into data groups and then process the data sets. The Window transformation is a passive transformation.

When you read from unbounded sources, you might want to accumulate the data into bounded data groups for further processing. To introduce bounded intervals to unbounded data, use a Window transformation.

When you configure a Window transformation, define the type of window and the data boundaries by time. To specify data boundaries, configure the window size and window slide interval. The window size defines the time interval for which data is accumulated as a data group. The slide interval defines the time interval after which the accumulated data group is processed further. The watermark delay defines the threshold time for a delayed event to be accumulated into a data group.

You can only run the Window transformation for streaming mappings on the Spark engine.

For more about the Window transformation, see the *Data Engineering Streaming User Guide*.

CHAPTER 52

Write Transformation

This chapter includes the following topics:

- [Write Transformation Overview, 691](#)
- [Write Transformation Properties , 691](#)
- [Create a Write Transformation, 697](#)

Write Transformation Overview

A Write transformation is a passive transformation. The mapping uses the Write transformation to write data to a target. The Write transformation is nonreusable.

You can create a Write transformation from a physical data object, a logical data object, or a parameter. If you want to create a Write transformation from a physical data object that you imported from a PowerExchange adapter source, the mapping editor might prompt you to specify a write operation before you can create a Write transformation from the data object.

The properties that you can configure for a Write transformation depend on the type of data object that you used to create the transformation.

Write transformations can represent dynamic targets. You can configure a Write transformation to dynamically update its ports, metadata, and other properties. For information about configuring dynamic targets, see the *Informatica Developer Mapping Guide*.

Write Transformation Properties

After you create a Write transformation, you can configure properties for the transformation.

Configure Write transformation properties on tabs in the **Properties** view of the transformation. The tabs that you can configure depend on the type of target that the Write transformation represents.

The following table describes each property tab and identifies the target type that you use the tab for:

Property Tab	Description	Target Type
General	Specify transformation properties and behavior. For relational and customized data object sources, synchronize transformation input ports with source.	All
Data Object	Specify the transformation data source. For relational and customized data object sources, get data object columns from data source at run time.	Flat file Relational Customized data object
Format	Input settings for a flat file data source	Flat file
Ports	Set port definition by the associated data object or by mapping flow.	All
Run-time	Properties that the Data Integration Service uses when writing data to the target at run time, such as where to send reject files. For a flat file target, reject file names and directories.	Flat file Relational
Data Object Parameters	View the data object parameters. Configure parameter values for the mapping or bind the parameters to a mapping parameters.	Flat file Customized data object Logical data objects
Run-time Linking	Create new run-time links and view link properties.	All
Advanced	Set tracing level and row order. For a relational or Hive target, set the target schema strategy.	Flat file Relational Customized data objects Logical data objects

General Properties

You can configure the name and description of the Write transformation. You can also configure the following properties:

When Column Metadata Changes

Available for relational and customized targets. Select one of the following options:

- Synchronize input ports. The Developer tool updates Write transformation input ports with metadata changes that the Model repository stores for the data object.
- Do not synchronize. The Developer tool does not show metadata changes in the data object.

Default is the Synchronize input ports option.

Physical Data Object

Available for flat file and customized targets. The object used to create the transformation.

You can select the data object name and configure its properties.

Data Object Properties

On the Data Object tab, you can specify or change the Write transformation target, and make relational and customized data object targets dynamic.

You can configure the following properties:

Specify By

To specify target columns and metadata for the Write transformation, select one of the following options:

- Value. The Write transformation uses the associated data object to specify target columns and metadata.
- Parameter. The Write transformation uses a parameter to specify target columns and metadata.

Default is the Value option.

Data Object

If you created the Write transformation from an existing data object, the field displays the name of the object. Click **Browse** to change the data object to associate with the Write transformation.

Parameter

Choose or create a parameter to associate with the Write transformation.

At run time, get data object columns from the data source

When you enable this option, the Data Integration Service fetches metadata and data definition changes from target tables to the Write transformation.

To preview how the Data Integration Service fetches metadata and data definition changes, view the mapping with resolved parameters.

Ports Properties

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

Columns defined by

Select one of the following options to define Write transformation columns:

- Associated data object. Use column names, metadata, and other properties from the data object in the Data Object tab.
- Mapping flow. The mapping fetches column names, metadata, and other properties from the upstream objects in the mapping.

Default is the Associated data object option.

Column resource properties

Available for flat file and customized targets. The resource for each column is the data object from which the column fetches its name, metadata, and other properties. Select the resource name to change resource properties.

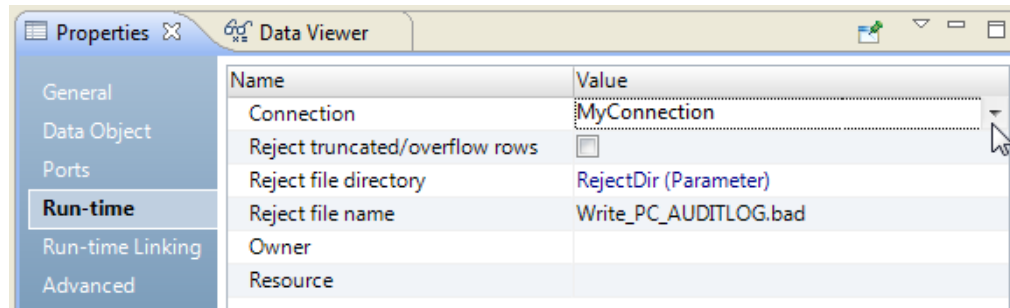
Run-Time Properties

You can configure the following Write transaction properties on the Run-time tab:

Connection

Available for relational targets. Connection used by the transformation. Click in the right side of the field to change the connection.

The following image shows the location of the dropdown button to click:



Reject truncated/overflow rows

Available for relational and customized targets.

The Developer tool lets you convert data by passing it from port to port. Sometimes a conversion causes an overflow of numeric data or truncation of strings in columns that contain characters. For example, passing data from a Decimal (28, 2) to a Decimal (19, 2) port causes a numeric overflow. Likewise, if you pass data from a String(28) port to a String(10) port, the Data Integration Service truncates the strings to 10 characters.

When a conversion causes an overflow, the Data Integration Service, by default, skips the row. The Data Integration Service does not write the data to the reject file. For strings, the Data Integration Service truncates the string and passes it to the next transformation.

Select this option to include all truncated and overflow data between the last transformation and target in the session reject file. The Data Integration Service sends all truncated rows and any overflow rows to the session reject file or to the row error logs, depending on how you configure the session.

Reject file directory

Directory where the reject file exists. Default is the RejectDir system parameter.

Reject file name

File name of the reject file. Default is <output_file_name>.bad.

If multiple partitions write to the flat file target, each partition writes to a separate reject file named <output_file_name><partition_number>.bad.

Run-time Linking Properties

Create and configure a run-time link on the **Run-time Linking** tab. A run-time link is a group-to-group link between transformations that uses a parameter, a link policy, or both to determine which ports to link at run time. Run-time links appear as thick lines in the mapping editor.

Create and configure run-time links to a Write transformation in the following cases:

- The target data object in the Write transformation uses a parameter.
- Ports from the upstream transformation can change at run time.

Note: Do not create a run-time link to a Write transformation when you define the target columns based on the mapping flow.

You can perform the following tasks on the **Run-time Linking** tab:

Create a run-time link.

In the **Links** area, click the **New** button and select the transformation from which you want to link the ports to the Write transformation at run time in the New Link dialog box.

Configure the run-time link properties.

In the **Link Properties** area, configure the following run-time link properties:

Parameter

Select this option when the port names can change between mapping runs and you know the port name values. Use a parameter of type Input Link Set to connect ports by name values between mapping runs. The syntax for the Input Link Set mapping parameter consists of comma-separated pairs of ports: `Afield1->Bfield2, Afield3->Bfield4` .

Link Policy

Select this option when you want to automatically link ports that have matching names. For example, when both of the mapping objects contain a port called SALARY, the Data Integration Service links them. You can ignore prefixes and suffixes in the port names.

At run time, the Data Integration Service establishes and resolves links between the ports in the following order:

- Links that you manually create in the mapping editor.
- Links based on the parameter that you configured for a run-time link.
- Links based on the link policy that you configured for a run-time link.

For more information on run-time links, see the *Informatica Developer Mapping Guide*.

Advanced Properties

Configure advanced properties to determine how the Data Integration Service processes data for the Write transformation.

Configure the following properties on the **Advanced** tab:

Tracing level

Control the amount of detail in the mapping log file.

Target load type

Type of target loading. Select Normal or Bulk. You can set the target load type for relational resources or customized data objects.

If you select Normal, the Data Integration Service loads targets normally. You can choose Bulk when you load to DB2, Sybase, Oracle, or Microsoft SQL Server. If you specify Bulk for other database types, the Data Integration Service reverts to a normal load. Bulk loading can increase mapping performance, but it limits the ability to recover because no database logging occurs. When you write to an Oracle target with bulk loading, you can optimize performance by disabling constraints in the Oracle database.

Choose Normal mode if the mapping contains an Update Strategy transformation. If you choose Normal and the Microsoft SQL Server target name includes spaces, configure the following environment SQL in the connection object:

```
SET QUOTED_IDENTIFIER ON
```

Update override

Overrides the default UPDATE statement for the target.

Delete

Deletes all rows flagged for delete.

Default is enabled.

Note: The Databricks Spark engine ignores this property when it writes to a Delta Lake target. To delete rows, use an Update Strategy transformation.

Insert

Inserts all rows flagged for insert.

Default is enabled.

Note: The Databricks Spark engine ignores this property when it writes to a Delta Lake target.

Target Schema Strategy

Type of target schema strategy for the relational or Hive target table.

You can select one of the following target schema strategies:

- **RETAIN - Retain existing target schema.** The Data Integration Service retains the existing target schema.
- **CREATE - Create or replace table at run time.** The Data Integration Service drops the target table at run time and replaces it with a table based on a target table that you identify.
- **Assign Parameter.** You can assign a parameter to represent the value for the target schema strategy and then change the parameter at run time.

DDL Query to create or replace

Creates or replaces the target table at run time based on a DDL query that you define. Applicable when you select **CREATE - Create or replace table at run time** target schema strategy option.

Truncate target table

Truncates the target before it loads data.

Default is enabled.

Truncate target partition

Truncates an internal or external partitioned Hive target before it loads data. You must choose **Truncate target table** before you choose this option.

Default is disabled.

Update strategy

Update strategy for existing rows. You can select one of the following strategies:

- Update as update. The Data Integration Services updates all rows flagged for update.
- Update as insert. The Data Integration Service inserts all rows flagged for update. You must also select the **Insert** target option.
Note: If the target is a Databricks Delta Lake, you do not need to select **Insert**.
- Update else insert. The Data Integration Service updates rows flagged for update if they exist in the target and then inserts any remaining rows marked for insert. You must also select the **Insert** target option.

Note: If the target is a Databricks Delta Lake, you do not need to select **Insert**.

PreSQL

SQL command the Data Integration Service runs against the target database before it reads the source.

The Developer tool does not validate the SQL.

PostSQL

SQL command that the Data Integration Service runs against the target database after it writes to the target.

The Developer tool does not validate the SQL.

Maintain row order

Maintain the row order of the input data to the target. Select this option if the Data Integration Service should not perform any optimization that can change the row order.

When the Data Integration Service performs optimizations, it might lose the row order that was established earlier in the mapping. You can establish row order in a mapping with a sorted flat file source, a sorted relational source, or a Sorter transformation. When you configure a target to maintain row order, the Data Integration Service does not perform optimizations for the target.

Constraints

SQL statements for table-level referential integrity constraints. Applies to relational targets only.

Create a Write Transformation

When you create a Write transformation, you choose one of the following methods based on the resource you create the transformation from:

Create the transformation from a data object in the Model repository.

Use this method if you want to base the Write transformation metadata on a data object.

To create a Write transformation in the mapping editor, use the **New Write Transformation** wizard, or drag a data object from the **Object Explorer** view and select Write as the transformation type.

Create the transformation from the flow of mapping objects.

Use this method to base the ports in the Write transformation on the flow of metadata from upstream mapping transformations.

Create the transformation using a parameter.

Use this method if you want the Write transformation to inherit ports from the object that the parameter represents.

Create the transformation using another transformation in the mapping.

Use this method if you want the Write transformation to have the same ports as the source transformation.

Creating a Write Transformation from a Data Object

You can create and configure a Write transformation in the mapping editor.

You might want to use this method if you want to configure specific properties for the transformation. For example, you can create the transformation, and then configure it to get column metadata from the data source at run time.

1. Right-click in the mapping editor and select **Add Transformation**.
The **Add Transformation** dialog box opens.
2. Select **Write** and click **OK**.
The **New Write Transformation** wizard opens.
3. Select **Physical Data Object** or **Logical Data Object**, and then click **Browse**.
The **Select Data Object** dialog box opens.

4. Select a data source, and then click **OK**.
5. To define Write transformation columns by the associated data object, select **Associated data object**.
The Write transformation ports are the same as the columns of the associated data object.
6. To dynamically update target object columns at run time with changes from the target file, select **At run-time, get data object columns from data source**.
The Data Integration Service refreshes the column metadata for the Write transformation when the mapping runs.
7. Click **Finish**.

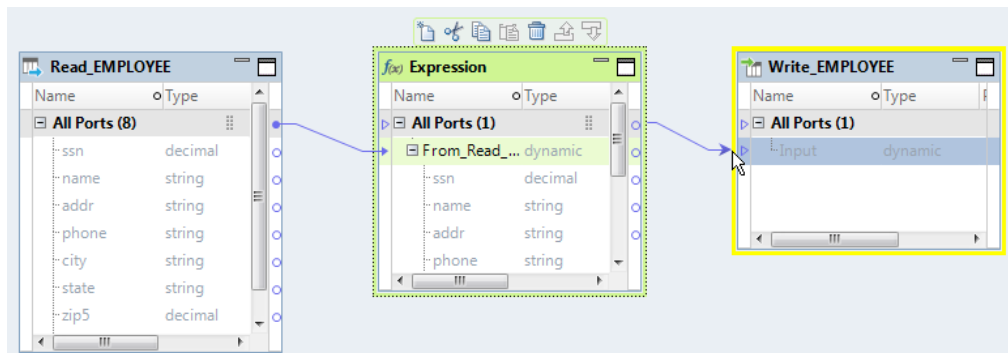
Creating a Write Transformation from Mapping Flow

You can create and configure a Write transformation in the mapping editor.

You might want to use this method if you want to configure specific properties for the transformation. For example, you can create the transformation, and then configure it to define columns based on mapping flow and to get column metadata from the data source at run time.

1. Right-click in the mapping editor and select **Add Transformation**.
The **Add Transformation** dialog box opens.
2. Select **Write** and click **OK**.
The **New Write Transformation** wizard opens.
3. Select **Physical Data Object** or **Logical Data Object**, and then click **Browse**.
The **Select Data Object** dialog box opens.
4. Select a data source, click **OK**.
5. Select **Mapping flow**, and then click **Finish**.
6. Drag ports from the All Ports port of the upstream to the **Input** port of the Write transformation.

The following image shows how to connect upstream ports to the **Input** port of the Write transformation:



The Write transformation receives column definitions from upstream mapping objects.

7. To dynamically update target object columns at run time with changes from the target file, select **At run-time, get data object columns from data source**.

The Data Integration Service refreshes the column metadata for the Write transformation when the mapping runs.

Creating a Write Transformation from a Parameter

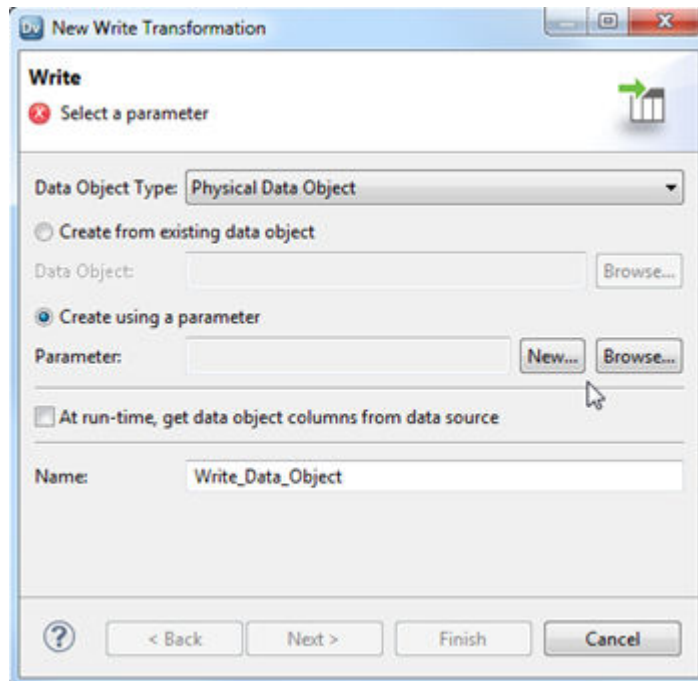
A parameter is a constant value that you can change between mapping runs.

In a dynamic mapping, you can use parameters to change sources and targets. You can also use parameters for input rules, selection rules, run-time links, and transformation properties. When you change the value of the parameter, the Data Integration Service creates or recreates the target according to the value specified in the parameter.

1. Right-click in the editor and select **Add Transformation**.
2. Select Write from the list of transformations, and then click **OK**.

Tip: Type the first letter of the transformation to filter the list.

The New Write Transformation dialog box opens.



3. Select **Create using a parameter**.
4. Browse to select the parameter, or click **New** to create a new parameter, and then browse to select a data object to create the parameter from.
5. Optionally, select the option **At run time, get data object columns from data source** to refresh the transformation columns at run time.
6. Optionally, click **Next** to choose how to define transformation columns.
7. Click **Finish**.

Creating a Write Transformation from an Existing Transformation

You can create a Write transformation with the same ports in a transformation that exists in the mapping.

You can create a target for complex files, flat files, or relational resources. If one of the ports in the existing transformation contains a complex port, you must create a complex file target and link ports by name or link

ports at run time based on the link policy. The Developer tool can create an Avro, Parquet, ORC, or JSON complex file target.

1. Open a mapping in the editor.
2. Right-click a transformation in the mapping editor and select **Create Target**.
The **Create Target** window opens.
3. Choose the data object type as Complex File, Flat File, Relational, or Other.
4. If you choose **Other**, select the required target object type from the list.
5. Choose a link type.

You can choose from the following link types:

Link ports by name

Ports in the Write transformation correspond to those in the source and have the same names.

Link dynamic port based on the mapping flow

The Write transformation contains dynamic ports based on upstream objects in the mapping flow.

Link ports at run time based on link policy

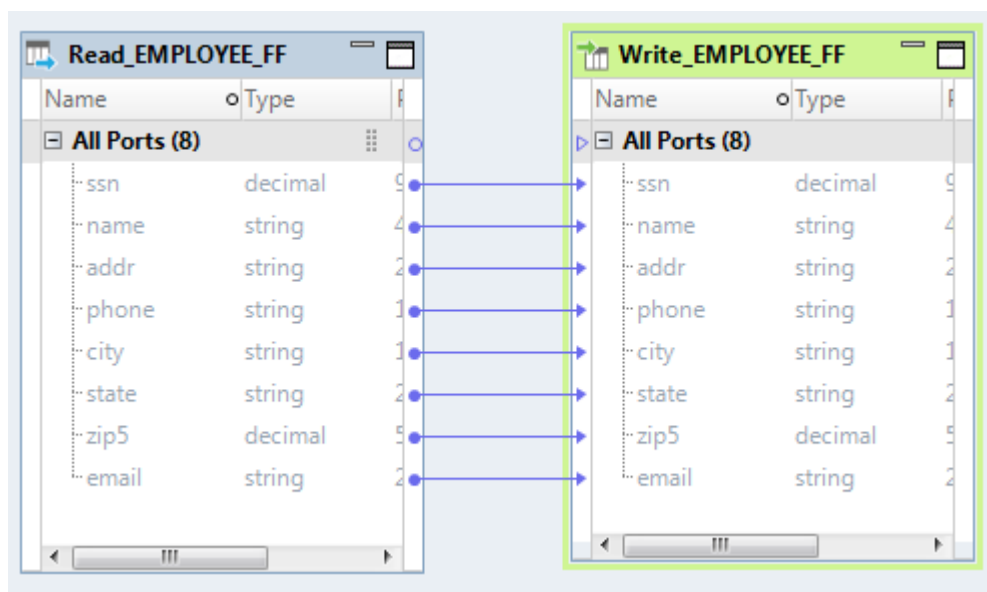
Ports are created in the target at run time based on the link policy that you configure on the Run Time Linking tab of the Write transformation.

For more information about dynamic ports and run-time link configuration, see the *Informatica Developer Mapping Guide*.

6. Name the new data object.
7. Optionally, click **Browse** to select a location for the data object.
8. If you chose to create a complex file target, select the complex file format as Avro or Parquet in the **Resource Format** drop-down box.
9. Click **Finish**.

The Developer tool performs the following tasks:

- Adds a Write transformation to the mapping.
The following image shows a Write transformation created from a Read transformation:



- Links ports.

- Creates a physical data object.
You can configure the physical data object properties. For example, you must specify an HDFS connection for the complex file data object.

APPENDIX A

Transformation Delimiters

This appendix includes the following topic:

- [Transformation Delimiters Overview, 702](#)

Transformation Delimiters Overview

Transformation delimiters specify divisions between data strings.

The following table lists the delimiters that transformations use to parse and write data strings:

Delimiter Name	Delimiter Symbol
at symbol	@
comma	,
dash	-
double quote	"
forward slash	/
full stop	.
hash	#
pipe	
semi-colon	;
single quote	'
space	[Spacebar]
tab	[Tab key]
underscore	_

INDEX

A

- ABORT function
 - using [51](#)
- active transformations
 - description [32](#)
 - Java [296](#), [297](#)
 - Rank [533](#)
- Address Validator transformation
 - non-native environment [127](#)
- advanced condition type
 - Joiner transformation [352](#)
- advanced interface
 - EDataType class [338](#)
 - example [340](#)
 - invoking Java expressions [337](#)
 - Java expressions [337](#)
 - JExpression class [340](#), [341](#)
 - JExprParaMetadata class [338](#)
- advanced properties
 - Association transformation [144](#)
 - Consolidation transformation [179](#)
 - Duplicate Record Exception transformation [259](#)
 - Java transformations [301](#)
 - Key Generator transformation [370](#)
 - REST Web Service Consumer transformation [570](#)
 - Web Service Consumer transformation [655](#)
- Advanced Properties
 - Bad Record Exception transformation [154](#)
 - SQL transformation [608](#)
- Aggregator transformation
 - advanced properties [137](#)
 - aggregate expressions [130](#)
 - aggregate functions [131](#)
 - Blaze engine [139](#)
 - cache size [69](#)
 - caches [135](#)
 - creating non-reusable [138](#)
 - creating reusable [137](#)
 - Databricks Spark engine [141](#)
 - developing [129](#)
 - dynamic mappings [129](#)
 - group by ports [132](#)
 - nested aggregate functions [132](#)
 - non-aggregate expressions [134](#)
 - non-native environment [139](#)
 - overview [128](#)
 - ports [129](#)
 - sorted input [135](#)
 - sorting data [136](#)
 - Spark engine [140](#)
 - tips [138](#)
 - troubleshooting [139](#)
 - Update Strategy combination [639](#)
 - using variables [46](#)

- all group
 - viewing in REST Web Service Consumer transformation [569](#)
 - viewing in Web Service Consumer transformation [650](#), [653](#)
- any elements
 - Web Service Consumer transformation [649](#), [652](#)
- anyAttribute attributes
 - Web Service Consumer transformation [649](#), [652](#)
- anyType
 - map ports [684](#)
- anyType elements
 - parsing [670](#)
 - Web Service Consumer transformation [649](#), [652](#)
- API methods
 - Java transformation [322](#)
 - Java transformations [322](#)
- assigning ports
 - quality issues [153](#)
- Association transformation
 - advanced properties [144](#)
 - overview [142](#)
 - tracing levels [144](#)
- At End tab
 - Java transformations [310](#)
- auto cache size
 - description [71](#)
- automatic consolidation
 - Duplicate Record Exception transformation [255](#)

B

- Bad Record Exception transformation
 - advanced properties [154](#)
 - blank quality issue ports [146](#)
 - Configuration view [151](#)
 - configuring [154](#)
 - example [155](#)
 - example bad records output [158](#)
 - example Issues output [158](#)
 - example mapplet [155](#)
 - example output [158](#)
 - good records example output [159](#)
 - input ports [150](#)
 - mapping [147](#)
 - output groups [146](#)
 - output ports [150](#)
 - overview [145](#)
 - port groups [149](#)
 - process flow [147](#)
 - quality issues [149](#)
- binding
 - WSDL file element [645](#)
- blocking transformations
 - description [41](#)
- blurring
 - date values [210](#)

- blurring (*continued*)
 - numeric values [209](#)
- buffer input port
 - Data Processor transformation [225](#)
- buffer output port
 - Data Processor transformation [226](#)

C

- cache
 - Lookup transformation [394](#)
- cache directory
 - Data Masking transformation [217](#)
- cache file directory
 - Association transformation [144](#)
 - Consolidation transformation [179](#)
 - Duplicate Record Exception transformation [259](#)
 - Key Generator transformation [370](#)
- cache file size
 - Association transformation [144](#)
 - Consolidation transformation [179](#)
 - Duplicate Record Exception transformation [259](#)
 - Key Generator transformation [370](#)
- cache files
 - directory [71](#)
 - overview [70](#)
- cache size
 - auto [71](#)
 - Data Masking transformation [217](#)
- caches
 - cache files [69](#)
 - data [70](#)
 - directory for files [71](#)
 - dynamic lookup cache [424](#)
 - index [70](#)
 - Lookup transformation [417](#)
 - overview [69](#)
 - partitioning [73](#)
 - size [71](#)
 - size increased by Data Integration Service [73](#)
 - size optimization [74](#)
 - static lookup cache [419](#)
 - transformations [69](#)
 - types [70](#)
- calculations
 - using variables with [47](#)
- Case Converter transformation
 - non-native environment [162](#)
 - overview [160](#)
- choice elements
 - description [685](#)
 - parsing SOAP messages [673](#)
 - viewing in REST Web Service Consumer transformation [569](#)
 - viewing in Web Service Consumer transformation [650](#), [653](#)
- Classifier transformation
 - classifier algorithm [164](#)
 - classifier model [164](#)
 - non-native environment [171](#)
 - overview [163](#)
- classpath
 - mapping property [304](#)
- Cluster Data
 - output group [257](#)
- clusters
 - Duplicate Record Exception transformation [254](#)
- code snippets
 - creating for Java transformations [306](#)

- Comparison transformation
 - non-native environment [176](#)
 - overview [172](#)
- compilation errors
 - identifying the source of in Java transformations [315](#)
- compiling
 - Java transformations [314](#)
- compiling Java code
 - Full Code tab [310](#)
- Component view
 - Data Processor transformation [224](#)
- composite keys
 - creating with Sequence Generator transformation [582](#)
 - REST Web Service Consumer transformation [566](#)
 - Web Service Consumer transformation [649](#)
- concurrent web service request messages
 - enabling in Web Service Consumer transformation [655](#)
- condition type
 - advanced for Joiner transformation [352](#)
 - simple for Joiner transformation [352](#)
- conditions
 - Joiner transformation [351](#)
 - Router transformation [576](#)
- Configuration view
 - Bad Record Exception transformation [151](#)
 - Bad Record Exception transformation example [157](#)
 - Duplicate Record Exception transformation [255](#)
- connected lookups
 - description [385](#)
 - overview [386](#)
- connected transformations
 - Java [296](#)
 - Rank [533](#)
 - Sequence Generator [581](#)
- connection
 - REST web services [570](#)
 - web services [655](#)
- considerations
 - Java transformations [300](#)
- Consolidation transformation
 - advanced properties [179](#)
 - non-native environment [193](#)
 - overview [177](#)
 - sorting records [179](#)
 - tracing levels [179](#)
- constants
 - replacing null values with [50](#)
- constraints
 - source optimization [548](#)
- Continue on SQL Error
 - SQL transformation [612](#), [615](#)
- cookie authentication
 - REST Web Service Consumer transformation [565](#)
 - Web Service Consumer transformation [648](#)
- creating
 - Java transformations [313](#), [314](#)
 - Sequence Data Object [586](#)
 - Sequence Generator transformation [589](#)
- Current Value (property)
 - Sequence Generator transformation [584](#)
- cycle
 - Sequence Generator transformation property [584](#)
- Cycle (property)
 - Sequence Generator transformation property [584](#)

D

- data
 - storing temporary [46](#)
- data caches
 - transformations [70](#)
- Data Integration Service)
 - restart mode [329](#)
- data masking transformation
 - masking IP addresses [212](#)
- Data Masking transformation
 - blurring [209](#)
 - cache directory [217](#)
 - cache size [217](#)
 - default value file [214](#)
 - dependent data masking [203](#)
 - description [194](#)
 - dictionary for substitution masking [200](#)
 - dictionary name expression masking [197](#)
 - expression masking [197](#)
 - expression masking guidelines [198](#)
 - mask format [207](#)
 - masking credit cards [211](#)
 - masking date values [209](#)
 - masking email addresses [211](#)
 - masking phone numbers [212](#)
 - masking Social Insurance numbers [214](#)
 - masking social security numbers [213](#), [215](#)
 - masking techniques [194](#)
 - masking URLs [213](#)
 - non-native environment [220](#)
 - random masking [195](#)
 - range [209](#)
 - repeatable expression masking [197](#)
 - repeatable SIN numbers [214](#)
 - repeatable SSN [213](#)
 - runtime properties [217](#)
 - shared storage table [217](#)
 - source string characters [208](#)
 - special mask formats [210](#)
 - storage commit interval [217](#)
 - storage table [197](#)
 - storage tables [201](#)
 - substitution masking [200](#)
 - substitution masking properties [201](#), [202](#)
 - unique output [217](#)
- data object
 - duplicate parameterized [398](#)
 - parameterizing [545](#)
- Data Object tab
 - description [545](#)
 - field descriptions [396](#)
 - parameterize using a new data object [398](#)
- Data Processing transformation
 - startup component [227](#)
- Data Processor Events view
 - Data Processor transformation [235](#)
 - viewing event log [237](#)
- Data Processor Hex Source view
 - description [224](#)
- Data Processor transformation
 - encoding settings [228](#)
 - creating [238](#)
 - description [223](#), [224](#)
 - exporting as a service [242](#)
 - input ports [225](#)
 - non-native environment [245](#)
 - output control settings [231](#)
- Data Processor transformation (*continued*)
 - output ports [226](#)
 - ports [225](#)
 - processing settings [232](#)
 - service parameter ports [226](#)
 - settings [228](#)
 - testing in the Data Viewer [242](#)
 - user logs [237](#)
 - views [224](#)
 - XMap settings [233](#)
 - XML settings [233](#)
- Data Transformation service
 - import multiple services [243](#)
 - importing to the Model repository [243](#)
- data types
 - Java transformations [297](#)
- date values
 - random data masking [196](#)
- datetime values
 - data masking [200](#)
- Decision transformation
 - advanced properties [251](#)
 - non-native environment [252](#)
 - overview [246](#)
 - quality issue sample strategy [149](#)
- default groups
 - Router transformation [575](#)
- default values
 - data masking [214](#)
 - entering [53](#), [55](#)
 - input ports [49](#), [50](#)
 - output ports [49](#), [50](#)
 - overview [49](#)
 - pass-through ports [49](#), [50](#)
 - rules for [55](#)
 - user-defined [50](#)
 - validating [53](#), [55](#)
- defineJExpression
 - Java expression API method [339](#)
- defineJExpression method
 - Java transformations [323](#)
- denormalized input
 - web service ports [682](#)
- denormalized output
 - SOAP message parsing [669](#)
- dependencies
 - implicit [63](#)
 - link path [63](#)
- dependent column
 - data masking [203](#)
- dependent masking
 - description [203](#)
- derived type elements
 - Web Service Consumer transformation [649](#), [652](#)
- derived types
 - parsing SOAP messages [671](#)
 - web services [683](#)
- design-time event log
 - description and location [236](#)
- designing
 - Java transformations [300](#)
- dictionary
 - repeatable expression masking [197](#)
 - substitution data masking [200](#)
- dictionary information
 - Data Masking transformation [202](#)
- distinct output
 - Sorter transformation [594](#)

- driver scores
 - Match transformation [448](#)
- dual-source match analysis [443](#)
- Duplicate Record Exception transformation
 - advanced properties [259](#)
 - automatic consolidation [255](#)
 - cluster output example [263](#)
 - clusters [254](#)
 - Configuration view [255](#)
 - configuring [264](#)
 - example [259](#)
 - example configuration settings [261](#)
 - generating duplicate records table [256](#)
 - mapping example [260](#)
 - output groups [257](#)
 - overview [253](#)
 - ports [256](#)
 - sample output [262](#)
 - sorting records [259](#)
 - Standard Output group [257](#)
 - tracing levels [259](#)
- dynamic expression
 - creating [279](#)
- dynamic expressions
 - example [276](#)
 - output port settings [277](#)
 - overview [276](#)
- dynamic lookup cache
 - description [424](#)
 - lookup SQL override [430](#)
 - use cases [425](#)
 - using flat file sources [424](#)
- dynamic mappings
 - port selector [270](#), [402](#)
 - Aggregator transformation [129](#)
 - Filter transformation [284](#)
 - Joiner transformations [348](#)
 - Lookup condition [400](#)
 - Lookup transformations [395](#)
 - Macro transformation [438](#)
 - Rank transformation [534](#)
 - Router transformation [575](#)
 - selection rules [270](#), [402](#)
 - Sorter transformation [593](#)
 - Update Strategy transformation [638](#)
- dynamic ports
 - join condition [354](#)
 - lookup condition [395](#)
- dynamic relational data objects
 - creating read transformations [549](#)
- dynamic URL
 - Web Service Consumer transformation [648](#)

E

- early selection optimization
 - SQL transformation [616](#)
 - Web Service Consumer transformation [659](#)
- EDataType class
 - Java expressions [338](#)
- Edit Output Group dialog box
 - Normalizer transformation [511](#)
- elements
 - union [686](#)
- encoding guidelines
 - Data Processor transformation [231](#)

- encoding settings
 - Data Processor transformation [228](#)
- encrypt storage
 - Data Masking transformation [217](#)
- encryption masking
 - description [205](#)
- End Value (property)
 - Sequence Generator transformation [584](#)
- endpoint URL
 - REST Web Service Consumer transformation [564](#)
 - Web Service Consumer transformation [648](#)
- error counts
 - incrementing for Java transformations [326](#)
- ERROR function
 - using [51](#)
- errors
 - handling [53](#)
 - increasing threshold in Java transformations [326](#)
- event log
 - viewing [237](#)
- event log, design-time
 - description and location [236](#)
- event types
 - Data Processor transformation [235](#)
- events
 - Data Processor transformation [235](#)
- Events view
 - Data Processor transformation [235](#)
- Events view, Data Processor
 - viewing event log [237](#)
- example
 - dynamic expression [276](#)
 - partition and order keys [274](#)
- example input file
 - Data Processor transformation [225](#)
- Example Source
 - creating in Data Processor transformation [241](#)
- Excel
 - configuring transformation ports [67](#)
 - copying ports [66](#)
 - copying to the Developer tool [67](#)
 - editing a transformation [68](#)
 - copying rules and guidelines [68](#)
- Exception transformation
 - Issue Assignment view [153](#)
- Expression Editor
 - description [43](#)
 - testing expressions [268](#)
 - validating expressions [44](#)
- expression masking
 - description [197](#)
 - repeatable masking [197](#)
 - repeatable masking example [198](#)
 - rules and guidelines [198](#)
- expression parameter
 - for a join condition [354](#)
 - lookup condition [403](#)
- Expression transformation
 - advanced properties [281](#)
 - Blaze engine [282](#)
 - Databricks Spark engine [282](#)
 - dynamic expression [276](#)
 - non-native environment [281](#)
 - output port settings [277](#)
 - overview [266](#)
 - port types [267](#)
 - Spark engine [282](#)
 - testing [268](#)

Expression transformation (*continued*)

using variables [46](#)

expressions

adding comments [44](#)

adding to a port [44](#)

entering [43](#)

in transformations [42](#)

Java transformations [332](#)

simplifying [46](#)

validating [44](#)

F

failing mappings

Java transformations [324](#)

failSession method

Java transformations [324](#)

failure event

Data Processor transformation [235](#)

fatal error event

Data Processor transformation [235](#)

field match analysis

field analysis defined [443](#)

process flow [465](#)

file input port

Data Processor transformation [225](#)

file output port

Data Processor transformation [226](#)

filter port

Web Service Consumer transformation [659](#)

Filter transformation

advanced properties [287](#)

Blaze engine [288](#)

dynamic mappings [284](#)

filter condition [285](#)

filter condition parameter [285](#)

non-native environment [287](#)

overview [283](#)

performance tips [287](#)

rows with null values [287](#)

filtering source rows

Lookup transformation [390](#)

finding

errors in Java code [315](#)

foreign keys

creating with Sequence Generator transformation [582](#)

format preserving encryption [205](#)

Forwarding Rejected Rows

configuring [639](#)

option [639](#)

Full Code tab

Java compilation errors [315](#)

Java transformations [310](#)

Functions tab

Java transformations [310](#)

G

GCID

description [498](#)

Normalizer transformation example [509](#)

generate bad records table

Bad Record Exception transformation [152](#)

generate Java code

Java expressions [334](#)

generate output row

Java transformations [324](#)

generated keys

web service output groups [668](#)

generateRow method

Java transformations [324](#)

generic fault output

enabling in Web Service Consumer transformation [655](#)

generic SOAP faults

Web Service Consumer transformation [657](#)

getBytes method

Java transformations [341](#)

getDouble method

Java transformations [341](#)

getInRowType method

Java transformations [325](#)

getInt method

Java transformations [342](#)

getLong method

Java transformations [342](#)

getMetadata method

Java transformations [326](#)

getResultDataType method

Java transformations [342](#)

getResultMetadata method

Java transformations [342](#)

getStringBuffer method

Java transformations [343](#)

Group By tab

description [133](#)

port list parameters [134](#), [537](#)

group filter condition

Router transformation [576](#)

groups

adding to Router transformation [579](#)

Router transformation [575](#)

user-defined [575](#)

groups in match analysis [444](#)

GZip

compressing SOAP messages [657](#)

H

Helpers tab

Java transformations [307](#), [309](#)

Hex Source view, Data Processor

description [224](#)

Hierarchical to Relational transformation

configure ports [293](#)

creating [294](#)

creating ports [294](#)

description [289](#)

development [293](#)

example [289](#)

ports [292](#)

testing in the Data Viewer [295](#)

high precision processing

Java transformations [301](#)

HTTP connection

REST web services [570](#)

HTTP error output

enabling in Web Service Consumer transformation [655](#)

HTTP header

adding to REST Web Service Consumer transformation [565](#)

adding to Web Service Consumer transformation [648](#)

HTTP response code

REST Web Service Consumer transformation [566](#)

Human tasks

bad record exceptions [149](#)

I

identity match analysis

identity analysis defined [443](#)

master data sets [450](#)

output properties for index data analysis [486](#)

persistence status codes [459](#)

persistence status descriptions [459](#)

persistent store of index data [450](#)

process flow [478](#)

rules and guidelines for persistent index data [450](#)

IIF function

replacing missing keys with Sequence Generator transformation [583](#)

Imports tab

Java transformations [307](#), [308](#)

Increment By (property)

Sequence Generator transformation property [584](#)

incrementErrorCount method

Java transformations [326](#)

incrementing

setting sequence interval [585](#)

index caches

transformations [70](#)

initializing

variables [49](#)

input hierarchy

Normalizer transformation [500](#)

input mapping

REST Web Service Consumer transformation [566](#)

Web Service Consumer transformation [649](#)

input ports

default values [49](#)

Java transformations [300](#), [301](#)

Input Ports area

generating SOAP messages [675](#)

input rows

getting the row type for [325](#)

Insert Else Update (property)

description [431](#)

instance variables

Java transformations [307](#)

invoke

Java expression API method [343](#)

invokeJExpression

API method [335](#)

invokeJExpression method

Java transformations [327](#)

isNull method

Java transformations [328](#)

isResultNull method

Java transformations [343](#)

Issue Assignment view

Bad Record Exception transformation [153](#)

Issues table

generating [152](#)

J

Java code

finding errors [315](#)

in Java transformations [304](#)

Java code snippets

creating for Java transformations [306](#)

Java expression API method

getResultDataType [342](#)

Java expression API methods

defineJExpression [339](#)

getBytes [341](#)

getDouble [341](#)

getInt [342](#)

getLong [342](#)

getResultMetadata [342](#)

getStringBuffer [343](#)

invoke [343](#)

isResultNull [343](#)

Java expressions

advanced interface [337](#)

advanced interface example [340](#)

configuring [333](#)

configuring functions [334](#)

creating [334](#)

creating in the Define Function dialog box [334](#)

EDataType class [338](#)

generate Java code [334](#)

generating [333](#)

invokeJExpression API method [335](#)

invoking [327](#)

invoking with advanced interface [337](#)

invoking with simple interface [335](#)

Java transformations [332](#)

JExpression class [340](#), [341](#)

JExprParaMetadata class [338](#)

rules and guidelines [335](#), [337](#)

rules and guidelines for invoking [327](#)

simple interface [335](#)

simple interface example [336](#)

Java packages

importing [307](#)

Java primitive data types

Java transformations [297](#)

Java transformation

API methods [322](#)

Blaze engine [319](#)

Non-native environment [319](#)

overview [296](#)

Spark engine [320](#)

Java transformations

active [297](#)

advanced properties [301](#)

API methods [322](#)

At End tab [310](#)

checking null values in [328](#)

compilation errors [315](#)

compiling [314](#)

creating [313](#), [314](#)

creating Java code snippets [306](#)

creating ports [300](#)

data type conversion [297](#)

default port values [300](#)

defineJExpression method [323](#)

designing [300](#)

failing mappings in [324](#)

failSession method [324](#)

Full Code tab [310](#)

Function tab [310](#)

generateRow method [324](#)

getInRowType method [325](#)

getMetadata method [326](#)

getting the input row type [325](#)

Java transformations *(continued)*

- Helpers tab [307](#), [309](#)
- high precision processing [301](#)
- identifying the source of compilation errors [315](#)
- Imports tab [307](#), [308](#)
- incrementErrorCount method [326](#)
- input ports [301](#)
- invokeJExpression method [327](#)
- isNull method [328](#)
- Java code [304](#)
- Java primitive data types [297](#)
- logError method [328](#)
- logInfo method [329](#)
- logs [328](#), [329](#)
- mapping-level classpath [304](#)
- nanosecond processing [301](#)
- non-user code errors [316](#)
- On Inputs tab [309](#)
- output ports [301](#)
- passive [297](#)
- resetNotification method [329](#)
- resetting variables in [329](#)
- retrieving metadata [326](#)
- setNull method [330](#)
- setting null values in [330](#)
- Stateless [301](#)
- storeMetadata method [331](#)
- storing metadata [331](#)
- Transformation Scope [301](#)
- troubleshooting [315](#)
- user code errors [315](#)

JDK

- Java transformation [296](#)

JExpression class

- Java expressions [340](#), [341](#)

JExprParaMetadata class

- Java expressions [338](#)

join condition

- overview [351](#)

Joiner transformation

- advanced condition type [352](#)
- advanced properties [345](#)
- Blaze engine [364](#)
- blocking source pipelines [362](#)
- cache size [69](#)
- caches [346](#)
- caching master rows [363](#)
- condition type [352](#)
- conditions [351](#)
- configuring sort order [357](#)
- Databricks Spark engine [365](#)
- detail outer join [356](#)
- dynamic mappings [348](#)
- dynamic ports [354](#)
- expression parameter [354](#)
- full outer join [356](#)
- join types [354](#)
- joining data from the same source [360](#)
- master outer join [356](#)
- non-native environment [364](#)
- normal join [355](#)
- overview [344](#)
- performance [363](#)
- port selectors [348](#)
- ports [347](#)
- rules and guidelines [364](#)
- selection rules [349](#)
- simple condition type [352](#)

Joiner transformation *(continued)*

- sorted [362](#)
 - sorted input [357](#)
 - Spark engine [364](#)
 - unsorted [362](#)
 - using port selectors [353](#)
- ## JRE
- Java transformation [296](#)

K

Key Generator transformation

- advanced properties [370](#)
- Non-native environment [370](#)
- overview [366](#)
- tracing levels [370](#)

key masking

- description [199](#)
- masking datetime values [200](#)
- masking numeric values [199](#)
- masking string values [199](#)
- numeric values [199](#)

keys

- creating with Sequence Generator transformation [582](#)
- SOAP message hierarchy [677](#)

L

Labeler transformation

- Non-native environment [383](#)
- overview [372](#)

Library

- creating in Data Processor transformation [240](#)

link

- one to many [61](#)
- one to one [61](#)

link scores

- Match transformation [448](#)

list elements

- description [685](#)
- parsing SOAP messages [673](#)

local variables

- overview [46](#)

Location column

- web service transformation [676](#)

log

- definition [236](#)

log, design-time event

- description and location [236](#)

logError method

- Java transformations [328](#)

logInfo method

- Java transformations [329](#)

logs

- Java transformations [328](#), [329](#)

lookup

- uncached [419](#)

lookup cache

- overview [394](#)

lookup caches

- definition [417](#)
- dynamic [424](#)
- overview [417](#)
- persistent [419](#)
- static [419](#)

- lookup condition
 - Data Masking transformation [202](#)
 - description [392](#)
 - specify by parameter [403](#)
 - using generated ports [400](#)
- lookup query
 - default query [388](#)
 - description [388](#)
 - ORDER BY [388](#)
 - override guidelines [389](#)
 - overriding [388](#), [390](#)
 - reserved words [389](#)
- lookup source
 - parameterizing [398](#)
- lookup source filter
 - limiting lookups [390](#)
- lookup SQL override
 - dynamic caches [430](#)
- Lookup transformation
 - advanced properties [407](#)
 - Blaze engine [413](#)
 - cache size [69](#)
 - cache size, reducing [388](#)
 - caches [417](#)
 - caching [394](#)
 - connected [385](#), [386](#)
 - creating non-reusable [409](#)
 - creating reusable [408](#)
 - creating unconnected lookups [410](#)
 - Databricks Spark engine [415](#)
 - default query [388](#)
 - developing [387](#)
 - dynamic cache [424](#)
 - dynamic mappings [395](#)
 - dynamic ports [395](#)
 - lookup condition [391](#)
 - lookup condition rules and guidelines [393](#)
 - lookup query override [388](#)
 - mapping parameters and variables [388](#)
 - Non-native environment [413](#)
 - overriding lookup [390](#)
 - overview [384](#)
 - persistent cache [419](#)
 - port name conflicts [397](#)
 - query properties [394](#)
 - relational parameterized data object [398](#)
 - run-time properties [406](#)
 - Spark engine [414](#)
 - uncached lookup [419](#)
 - unconnected [385](#), [387](#)
- lower threshold
 - configuring [151](#), [255](#)

M

- macro instruction
 - description [437](#)
- Macro transformation
 - description [437](#)
 - dynamic mappings [438](#)
 - dynamic ports [438](#)
 - macro instruction [437](#)
 - non-native environment [440](#)
 - parameters [438](#)
- maintain row order
 - Sequence Generator transformation [586](#)

- manual consolidation
 - Duplicate Record Exception transformation [255](#)
- Mapper
 - description [223](#)
- mapping parameters
 - in lookup SQL override [388](#)
- mapping variables
 - in lookup SQL override [388](#)
- mappings
 - flagging rows for update [638](#)
 - using Router transformations [580](#)
- Mapplet
 - reference [228](#)
- mask format
 - masking string values [207](#)
 - special mask formats [210](#)
- masking rules
 - blurring [209](#)
 - mask format [207](#)
 - range [209](#)
 - result string replacement characters [208](#)
 - source string characters [208](#)
 - special mask formats [210](#)
- masking techniques
 - data masking [194](#)
 - encryption [205](#)
- match strategy
 - Duplicate Record Exception transformation [260](#)
- Match transformation
 - Blaze engine [464](#)
 - cluster score options [446](#)
 - concepts in match analysis [442](#)
 - configuring a match analysis operation [463](#)
 - creating a mapplet for match analysis [462](#)
 - driver scores [448](#)
 - dual-source analysis [443](#)
 - field analysis defined [443](#)
 - field analysis process flow [465](#)
 - grouped data [444](#)
 - identity analysis defined [443](#)
 - identity analysis process flow [478](#)
 - identity index tables [450](#)
 - link scores [448](#)
 - Match Output view options [469](#)
 - Match Output view properties [470](#)
 - Match properties on the Match Output view [486](#)
 - Non-native environment [464](#)
 - null match scores [446](#)
 - output formats [484](#)
 - Output properties on the Match Output view [486](#)
 - performance factors [451](#)
 - persistence status codes [459](#)
 - persistence status descriptions [459](#)
 - predefined input ports [457](#)
 - predefined output ports [458](#)
 - sample match strategy [260](#)
 - single-source analysis [443](#)
 - Spark engine [464](#)
 - use cases [441](#)
 - viewing cluster analysis data [452](#)
 - viewing performance data [452](#)
- Max Output Row Count
 - SQL transformation [612](#)
- Maximum Output Row Count
 - SQL transformation [614](#)
- Merge transformation
 - Non-native environment [520](#)
 - overview [519](#)

- methods
 - Java transformation API [322](#)
- missing values
 - replacing with Sequence Generator [583](#)
- multi-group
 - transformations [41](#)
- multiple matches
 - Lookup transformation [413](#)
- multiple-occurring fields
 - Normalizer transformation [498](#)

N

- nanosecond processing
 - Java transformations [301](#)
- NEXTVAL port
 - Sequence Generator [582](#)
- non-reusable transformations
 - description [57](#)
- non-user code errors
 - in Java transformations [316](#)
- Normalizer transformation
 - advanced properties [512](#)
 - creating [513](#)
 - definition example [515](#)
 - dragging ports from other objects [514](#)
 - editing an output group [511](#)
 - example use case [514](#)
 - GCID [498](#)
 - input and output groups example [516](#)
 - input hierarchy [500](#)
 - input ports [501](#)
 - key generation [512](#)
 - mapping example [514](#)
 - mapping output example [517](#)
 - merging fields [502](#), [503](#)
 - multiple-occurring fields [498](#)
 - multiple-occurring records [499](#)
 - Non-native environment [518](#)
 - output groups [508](#), [509](#)
 - overview [497](#)
 - subrecords [499](#)
- Normalizer view
 - description [500](#)
- notification event
 - Data Processor transformation [235](#)
- null match scores
 - Match transformation [446](#)
- null values
 - checking in Java transformations [328](#)
 - replacing with a constant [50](#)
 - setting for Java transformation [330](#)
 - skipping [52](#)
- number of cached values
 - Sequence Generator property value [584](#)
- Number of Rows Affected
 - SQL transformation [608](#)
- numeric values
 - key masking [199](#)
 - random masking [196](#)
- NumRowsAffected
 - rows output [614](#)

O

- On Inputs tab
 - Java transformations [309](#)
- operation
 - WSDL file element [645](#)
- operation area
 - web service transformations [676](#)
- Operation Input area
 - customizing Web Service Consumer transformation [650](#)
- Operation Output area
 - customizing Web Service Consumer transformation [653](#)
- optional failure event
 - Data Processor transformation [235](#)
- ORDER BY
 - lookup query [388](#)
 - override [388](#)
- output control settings
 - Data Processor transformation [231](#)
- output groups
 - Bad Record Exception transformation [146](#)
 - Normalizer transformation [509](#)
- output mapping
 - REST Web Service Consumer transformation [568](#)
 - Web Service Consumer transformation [652](#)
- output port settings
 - description [277](#)
- output ports
 - default values [49](#)
 - error handling [49](#)
 - Java transformation [300](#)
 - Java transformations [301](#)
- override lookup
 - guidelines [389](#)
- override SOAP action
 - Web Service Consumer transformation [655](#)

P

- parameter binding
 - SQL transformation [611](#)
- Parameter option
 - Joiner transformation [354](#)
- parameters
 - filter condition [285](#)
 - Macro transformation [438](#)
 - Router transformation [578](#)
- Parser
 - description [223](#)
- Parser transformation
 - Non-native environment [531](#)
 - overview [521](#)
- pass-through ports
 - Expression transformation [267](#)
 - SQL transformation [606](#)
- pass-throught ports
 - default values [49](#)
- passive transformations
 - Java [296](#), [297](#)
 - overview [33](#)
 - Sequence Generator [581](#)
- performance
 - using variables to improve [46](#)
- persistent lookup cache
 - overview [419](#)
- persistent storage of identity index data [450](#)

- physical data objects
 - synchronization [545](#)
- pivoted data
 - SOAP messages [680](#)
- pivoted output
 - SOAP message parsing [670](#)
- port attributes
 - propogating [63](#)
- port list parameters
 - Group By tab [134](#), [537](#)
- port selector
 - selection rules [270](#), [401](#), [402](#)
 - creating [271](#), [350](#), [404](#)
 - in dynamic expressions [276](#)
 - Joiner transformation [348](#), [349](#)
 - selection rules [270](#), [401](#), [402](#)
- port selectors
 - in Joiner transformations [353](#)
- port values
 - Java transformations [300](#)
- ports
 - Bad Record Exception transformation [149](#)
 - configure [60](#)
 - copying from Excel [66](#)
 - creating [59](#)
 - default values overview [49](#)
 - denormalized web service input [682](#)
 - Duplicate Record Exception transformation [256](#)
 - evaluation order [48](#)
 - Java transformations [300](#)
 - linking [60](#)
 - linking automatically [61](#)
 - linking by name [61](#)
 - linking by position [62](#)
 - linking manually [61](#)
 - linking rules and guidelines [62](#)
 - map to SOAP messages [678](#)
 - propagated attributes by transformation [64](#)
 - Router transformation [579](#)
 - Sequence Generator transformation [581](#)
 - variables ports [46](#)
- Ports view
 - SQL transformation output [605](#)
- primary keys
 - creating with Sequence Generator transformation [582](#)
- properties
 - Read transformation
 - properties [541](#)
 - Write transformation
 - properties [691](#)
- push-into optimization
 - enabling in SQL transformation [617](#)
 - SQL transformation [616](#)
 - Web Service Consumer transformation [659](#)

Q

- Qname elements
 - parsing SOAP messages [672](#)
- quality issue ports
 - blank versus NULL [146](#)
- quality issues
 - assigning ports to [153](#)
- quality issues ports
 - description [150](#)
- queries
 - Lookup transformation [388](#)

- queries (*continued*)
 - overriding lookup [388](#)

R

- random masking
 - masking date values [196](#)
 - masking string values [195](#)
 - numeric values [195](#)
- range
 - masking numeric values [209](#)
- rank port
 - Rank transformation [536](#)
- Rank transformation
 - advanced properties [538](#)
 - Blaze engine [539](#)
 - cache size [69](#)
 - caches [537](#)
 - Databricks Spark engine [540](#)
 - defining groups for [536](#)
 - dynamic mappings [534](#)
 - Non-native environment [539](#)
 - options [534](#)
 - overview [533](#)
 - ports [534](#)
 - rank port [536](#)
 - RANKINDEX port [535](#)
 - Spark engine [539](#)
 - using variables [46](#)
- ranking
 - groups of data [536](#)
 - string values [534](#)
- Read data object
 - parameterizing [546](#)
- Read transformation
 - overview [541](#)
- read transformations
 - creating from relational data objects [549](#)
- records
 - Normalizer transformation [499](#)
- References view
 - Data Processor transformation [228](#)
 - Hierarchical to Relational transformation [293](#)
 - Relational to Hierarchical transformation [555](#)
- reject file
 - update strategies [639](#)
- rejected records
 - Bad Record Exception transformation [150](#)
- relational data objects
 - creating a dynamic read transformation [549](#)
- Relational to Hierarchical transformation
 - creating [555](#)
 - creating ports [555](#)
 - description [551](#)
 - development [555](#)
 - example [552](#)
 - ports [554](#)
- Repository view
 - Data Processor transformation [224](#)
- reserved words
 - lookup query [389](#)
- reset
 - Sequence Generator transformation [585](#)
- Reset (property)
 - Sequence Generator transformation [585](#)
- resetNotification method
 - Java transformations [329](#)

- response code
 - REST Web Service Consumer transformation [566](#)
- REST Web Service Consumer transformation
 - advanced properties [570](#)
 - argument ports [564](#)
 - configuration [559](#)
 - connection property [570](#)
 - cookie ports [565](#)
 - creating [571](#)
 - customizing output mapping view [569](#)
 - Delete method [563](#)
 - Get method [561](#)
 - HTTP header ports [565](#)
 - HTTP methods [560](#)
 - input mapping [566](#)
 - input mapping rules [566](#)
 - input ports [564](#)
 - internet media type [570](#)
 - mapping elements to ports [563](#)
 - mapping input [557](#)
 - mapping input ports [567](#)
 - mapping output [557](#)
 - mapping output ports [569](#)
 - message configuration [559](#)
 - non-reusable [571](#)
 - output mapping [568](#)
 - output mapping rules [569](#)
 - output ports [564](#)
 - output XML ports [565](#)
 - overview [557](#)
 - pass-through ports [564](#)
 - ports [563](#)
 - Post method [561](#)
 - process [558](#)
 - proxy server support [557](#)
 - Put method [562](#)
 - RequestInput port [563](#)
 - resource identification [559](#)
 - response code ports [566](#)
 - reusable [571](#)
 - security [646](#)
 - setting base URL [570](#)
 - sorted input [570](#)
 - tracing level [570](#)
 - transport layer security [646](#)
 - URL ports [564](#)
 - XML schema validation [570](#)
- result sets
 - output parameter placement [625](#)
 - sample stored procedures [623](#)
 - SQL transformation [623](#)
- result string replacement characters
 - Data Masking transformation [208](#)
- Reusable
 - Sequence Generator transformation [586](#)
- reusable transformations
 - description [56](#)
 - editing [57](#)
- Router transformation
 - advanced properties [580](#)
 - connecting in mappings [580](#)
 - dynamic mappings [575](#)
 - dynamic ports [578](#)
 - example [576](#)
 - group filter condition [576](#)
 - groups [575](#)
 - Non-native environment [580](#)
 - overview [574](#)

- Router transformation (*continued*)
 - parameters [578](#)
 - ports [579](#)
- rows
 - flagging for update [638](#)
- rules
 - default values [55](#)
- rules and guidelines
 - windowing properties [276](#)
- run-time event log
 - Data Processor transformation [237](#)
- Runtime view
 - SQL transformation [620](#)

S

- sample source file
 - defining in Data Processor transformation [239](#)
- schemas
 - Data Processor transformation [228](#)
 - Hierarchical to Relational transformation [293](#)
 - Relational to Hierarchical transformation [555](#)
- scope
 - master and detail [349](#)
 - port selector [270, 401](#)
- Script
 - creating in Data Processor transformation [239](#)
- Script Help view
 - Data Processor transformation [224](#)
- selection criteria
 - port selector [270, 401](#)
- selection rules
 - dynamic mappings [270, 402](#)
 - Joiner transformation [349](#)
 - port selectors [270, 401](#)
- Sequence Data Object
 - creating [586](#)
 - properties [586](#)
- Sequence Generator transformation
 - advanced properties [585](#)
 - Blaze engine [591](#)
 - creating [589](#)
 - creating composite key [582](#)
 - cycle [584](#)
 - Cycle [585](#)
 - Increment By property [585](#)
 - maintain row order [586](#)
 - NEXTVAL port [582](#)
 - Non-native environment [591](#)
 - overview [581](#)
 - ports [581](#)
 - properties [584–586](#)
 - range of values [585](#)
 - reset [585](#)
 - Spark engine [591](#)
 - start value [584](#)
 - using IIF function to replace missing keys [583](#)
- sequence group
 - viewing in REST Web Service Consumer transformation [569](#)
- service
 - WSDL file element [645](#)
- service parameter port
 - Data Processor transformation [225](#)
- service parameter ports
 - Data Processor transformation [226](#)
- setNull method
 - Java transformations [330](#)

- Settings view
 - Data Processor transformation [228](#)
- shared lookup cache
 - guidelines [421](#)
 - partitioning, guidelines [421](#)
- shared storage table
 - Data Masking transformation [217](#)
- side effects
 - SQL transformation [616](#)
 - Web Service Consumer transformation [659](#)
- simple condition type
 - Joiner transformation [352](#)
- simple interface
 - example [336](#)
 - Java expressions [335](#)
 - Java transformation API methods [335](#)
- SIN numbers
 - masking Social Insurance numbers [214](#)
 - repeatable data masking [214](#)
- single-source match analysis [443](#)
- SOAP action
 - overriding in Web Service Consumer transformation [655](#)
- SOAP compression
 - Web Service Consumer transformation [657](#)
- SOAP hierarchy
 - relationship to input ports [676](#)
- SOAP message
 - keys [677](#)
- SOAP message parsing
 - denormalized output [669](#)
 - derived types [671](#)
 - description [666](#)
 - normalized output [668](#)
 - pivoted output [670](#)
 - Qname elements [672](#)
 - union element [673](#)
- SOAP messages
 - map choice elements [685](#)
 - map list elements [685](#)
 - map multiple input ports [680](#)
 - map multiple-occurring nodes [668](#)
 - map ports [678](#)
 - map ports to union elements [686](#)
 - overview [645](#)
 - parsing anyType elements [670](#)
 - parsing choice elements [673](#)
 - parsing list elements [673](#)
 - parsing substitution groups [672](#)
 - pivoting data [680](#)
- Social Security numbers
 - area code masking [213](#)
 - repeatable data masking [213](#)
- Sort keys
 - configuring [594](#)
- sort property
 - Consolidation transformation [179](#)
 - Duplicate Record Exception transformation [259](#)
- Sort tab
 - Sorter transformation [594](#)
- Sorter
 - variable length [597](#)
- Sorter transformation
 - Blaze engine [600](#)
 - cache size [69](#)
 - caches [597](#)
 - Databricks Spark engine [601](#)
 - dynamic mappings [593](#)
 - Non-native environment [600](#)
- Sorter transformation (*continued*)
 - overview [592](#)
 - Sort tab [594](#)
 - Spark engine [600](#)
- source optimization
 - constraints [548](#)
- source string characters
 - Data Masking transformation [208](#)
- special format masking
 - credit card numbers [211](#)
 - email addresses [211](#)
 - IP addresses [212](#)
 - phone numbers [212](#)
 - repeatable SIN numbers [214](#)
 - Social Insurance numbers [214](#)
 - Social Security numbers [213](#)
 - URLs [213](#)
- SQL input ports
 - SQL transformation [604](#)
- SQL query
 - SQL transformation [611](#)
- SQL transformation
 - INOUT parameters [622](#)
 - Advanced Properties view [608](#)
 - calling a stored procedure [622](#)
 - continuing on SQL error [615](#)
 - create as empty [627](#)
 - creating from a stored procedure [627](#)
 - defining output ports [605](#)
 - defining the database connection [620](#)
 - defining the query [611](#)
 - early selection optimization [616](#)
 - example [617](#)
 - input ports description [604](#)
 - input row to output row cardinality [612](#)
 - number of rows affected [608](#)
 - number of rows output [614](#)
 - overview [603](#)
 - parameter binding [611](#)
 - pass-through ports [606](#)
 - ports [604](#)
 - push-into optimization [616](#)
 - push-into optimization properties [617](#)
 - query statement [620](#)
 - query string substitution [612](#)
 - restricting output rows [614](#)
 - result sets [623](#)
 - return value port [622](#)
 - run-time connection [626](#)
 - SQLException port [607](#)
 - stored procedure parameters [622](#)
 - stored procedures [621](#)
- SQLException port
 - SQL transformation [607](#)
- Standard Output group
 - Duplicate Record Exception transformation [257](#)
- Standard Output port
 - description [150](#)
- Standardizer transformation
 - Non-native environment [632](#)
 - overview [629](#)
- start digit
 - Social Insurance numbers [214](#)
- start value
 - Sequence Generator transformation [584](#)
- Start Value (property)
 - Sequence Generator transformation [584](#)

- startup component
 - Data Processor transformation [227](#)
- Stateless
 - Java transformations [301](#)
- static code
 - Java transformations [307](#)
- static lookup cache
 - overview [419](#)
- static variables
 - Java transformations [307](#)
- storage commit interval
 - Data Masking transformation [217](#)
- storage encryption key
 - Data Masking transformation [217](#)
- storage table
 - expression masking [197](#)
 - substitution data masking [201](#)
- stored procedures
 - example [625](#)
 - input and output ports [622](#)
 - parameters [622](#)
 - result set examples [623](#)
 - return value [622](#)
 - return value port [622](#)
 - SQL transformation [621](#)
 - writing to variables [48](#)
- storeMetadata method
 - Java transformations [331](#)
- Streamer
 - description [223](#)
- string substitution
 - SQL transformation [612](#)
- string values
 - custom data masking [195](#)
 - key data masking [199](#)
- strings
 - ranking [534](#)
- subrecords
 - Normalizer transformation [499](#)
- substitution groups
 - parsing SOAP messages [672](#)
 - Web Service Consumer transformation [649](#), [652](#)
 - web services [684](#)
- substitution masking
 - description [200](#)
 - masking properties [201](#)
- synchronization
 - customized data objects [545](#)
 - physical data objects [545](#)
- synchronize with editor
 - Data Processor transformation [242](#)

T

- targets
 - relational [637](#)
- tracing levels
 - Association transformation [144](#)
 - Consolidation transformation [179](#)
 - Duplicate Record Exception transformation [259](#)
 - Key Generator transformation [370](#)
 - Sequence Generator transformation property [585](#)
- transformation ports
 - overview [59](#)
- Transformation Scope
 - Java transformations [301](#)

- transformations
 - configuring ports in Excel [67](#)
 - copying metadata [67](#)
 - active [32](#)
 - cache files [70](#)
 - cache partitioning [73](#)
 - cache size [71](#), [73](#)
 - cache size optimization [74](#)
 - caches [69](#)
 - connected [33](#)
 - creating [58](#)
 - developing [41](#)
 - editing in Excel [68](#)
 - editing reusable [57](#)
 - expression validation [44](#)
 - expressions [42](#)
 - Hadoop environment [36](#)
 - handling errors [53](#)
 - Java [296](#)
 - multi-group [41](#)
 - non-reusable [57](#)
 - overview [32](#)
 - reusable [56](#), [57](#)
 - Sequence Generator [581](#)
 - unconnected [33](#)
- Transformer
 - description [223](#)
- transport layer security
 - REST Web Service Consumer transformation [646](#)
 - Web Service Consumer transformation [646](#)
- treat fault as error
 - enabling in Web Service Consumer transformation [655](#)
- troubleshooting
 - Java transformations [315](#)

U

- unconnected lookups
 - description [385](#)
 - overview [387](#)
- unconnected transformations
 - Lookup transformation [387](#)
- union element
 - parsing SOAP messages [673](#)
- union elements
 - description [686](#)
- Union transformation
 - Databricks Spark engine [636](#)
 - Non-native environment [636](#)
 - overview [633](#)
- unique output
 - Data Masking transformation [202](#)
- unique records table
 - creating [264](#)
- Update Else Insert (property)
 - description [431](#)
- Update Strategy transformation
 - advanced properties [639](#)
 - Aggregator combination [639](#)
 - Blaze engine [641](#)
 - creating [638](#)
 - dynamic mappings [638](#)
 - expressions [638](#)
 - forwarding rejected rows [639](#)
 - Non-native environment [640](#)
 - overview [637](#)
 - Spark engine [642](#)

Update Strategy transformation (*continued*)

- steps to configure [637](#)
- upper threshold
 - configuring [151](#), [255](#)
- Use Hive Merge
 - option [639](#)
- user code errors
 - Java transformations [315](#)
- user log
 - Data Processor transformation [237](#)
- user-defined group
 - Router transformation [575](#)
- user-defined methods
 - Java transformations [307](#)

V

- validating
 - default values [53](#), [55](#)
- Validation Rule object
 - creating in Data Processor transformation [241](#)
- variable length
 - in the Sorter transformation [597](#)
- variable ports
 - Expression transformation [267](#)
 - overview [46](#)
- variables
 - initializations [49](#)
 - Java transformations [307](#)
 - overview [46](#)
 - port evaluation order [48](#)
 - stored procedure results, capturing [48](#)

W

- warning event
 - Data Processor transformation [235](#)
- web service
 - derived types [683](#)
 - map ports to anyTypes [684](#)
 - substitution groups [684](#)
- Web Service Consumer transformation
 - adding HTTP headers [648](#)
 - advanced properties [655](#)
 - concurrent web service request messages [655](#)
 - cookie authentication [648](#)
 - creating [661](#)
 - dynamic web service URL [648](#)
 - dynamic WS-Security name [648](#)

Web Service Consumer transformation (*continued*)

- early selection optimization [659](#)
- enabling generic fault output [655](#)
- enabling HTTP error output [655](#)
- enabling push-into optimization [660](#)
- endpoint URL [648](#)
- error handling [657](#)
- filter optimization [659](#)
- generic SOAP faults [657](#)
- input mapping [649](#)
- mapping input ports [649](#)
- mapping output nodes [652](#)
- operations [645](#)
- output mapping [652](#)
- overview [644](#)
- push-into optimization [659](#)
- security [646](#)
- SOAP compression [657](#)
- SOAP messages [645](#)
- transport layer security [646](#)
- viewing keys [650](#), [653](#)
- web service transformations
 - Location column [676](#)
- web services connections
 - overview [655](#)
- Weighted Average transformation
 - Non-native environment [689](#)
- windowing
 - properties [273](#)
- windowing properties
 - frame [273](#)
 - order [273](#), [274](#)
 - partition [273](#), [274](#)
 - rules and guidelines [276](#)
- Write transformation
 - overview [691](#)
- WS-Security user name
 - dynamic port [648](#)
- WSDL file
 - binding element [645](#)
 - operation element [645](#)
 - port element [645](#)
 - service element [645](#)

X

- XMap object
 - creating in Data Processor transformation [240](#)