DISCLAIMER: Informatica Corporation provides this documentation "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of non-infringement, merchantability, or use for a particular purpose. Informatica Corporation does not warrant that this software or documentation is error free. The information provided in this software or documentation may include technical inaccuracies or typographical errors. The information in this software and documentation is subject to change at any time without notice.

NOTICES

This Informatica product (the “Software”) includes certain drivers (the “DataDirect Drivers”) from DataDirect Technologies, an operating company of Progress Software Corporation (“DataDirect”) which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.

Part Number: IN-TG-901HF2-0001
# Table of Contents

**Preface** ......................................................................................... x
Informatica Resources. ................................................................. x
  Informatica Customer Portal. .................................................... x
  Informatica Documentation. .................................................... x
  Informatica Web Site. ............................................................. x
  Informatica How-To Library. .................................................... x
  Informatica Knowledge Base. .................................................. xi
  Informatica Multimedia Knowledge Base. ............................... xi
  Informatica Global Customer Support. ...................................... xi

**Chapter 1: Working with Transformations.** .............................. 1
Working with Transformations Overview ........................................ 1
  Active Transformations. .......................................................... 1
  Passive Transformations. ......................................................... 1
  Multi-Strategy Transformations. .............................................. 2
  Transformation Descriptions. .................................................. 2
Developing a Transformation. ..................................................... 3
Reusable Transformations. .......................................................... 3
  Reusable Transformation Instances and Inherited Changes .......... 4
  Editing a Reusable Transformation. ......................................... 4
Expressions in Transformations. ................................................ 4
  The Expression Editor. ........................................................... 5
  Port Names in an Expression. .................................................. 5
  Adding an Expression to a Port. .............................................. 6
  Comments in an Expression. .................................................. 6
  Expression Validation. ............................................................ 6
Creating a Transformation. .......................................................... 6

**Chapter 2: Address Validator Transformation.** .......................... 7
Address Validator Transformation Overview ..................................... 7
Address Validator Transformation Input Port Groups. .................... 8
Address Validator Transformation Output Port Groups. .................. 9
CASS and SERP Certification Reports. ......................................... 10
Formatted Addresses in the Address Validator Transformation .......... 11
Address Validator Status Ports. .................................................. 11
  ElementInputStatus Output Port Values. ................................ 12
  ElementRelevance Output Port Values. ................................... 13
  ElementResultStatus Output Port Values. ............................... 13
Chapter 3: Aggregator Transformation

Aggregator Transformation Overview.

Developing an Aggregator Transformation.

Aggregator Transformation Ports.

Advanced Properties for Aggregator Transformations.

Aggregate Caches.

Aggregate Expressions.

Aggregate Functions.

Nested Aggregate Functions.

Conditional Clauses in Aggregate Expressions.

Group By Ports.

Non-Aggregate Expressions.

Default Values of Group By Ports

Sorted Input for an Aggregator Transformation.

Sorted Input Conditions.

Sorting Data in an Aggregator Transformation.

Creating a Reusable Aggregator Transformation.

Creating a Non-Reusable Aggregator Transformation.

Tips for Aggregator Transformations.

Troubleshooting Aggregator Transformations.

Chapter 4: Association Transformation

Association Transformation Overview.

Chapter 5: Case Converter Transformation

Case Converter Transformation Overview.

Case Strategy Properties.

Configuring a Case Converter Strategy.

Chapter 6: Comparison Transformation

Comparison Transformation Overview.

Field Matching Strategies.

Bigram.
Chapter 14: Java Expressions

Java Expressions Overview
Expression Function Types
Using the Define Function Dialog Box to Define an Expression
Step 1. Configure the Function
Step 2. Create and Validate the Expression
Step 3. Generate Java Code for the Expression
Creating an Expression and Generating Java Code by Using the Define Function Dialog Box
Java Expression Templates
Working with the Simple Interface
invokeJExpression
Simple Interface Example
Working with the Advanced Interface
Invoking an Expression with the Advanced Interface
Rules and Guidelines for Working with the Advanced Interface
EDDataType Class
JExprParamMetadata Class
defineJExpression
JExpression Class
Advanced Interface Example
JExpression Class API Reference
getBytes
getDouble
getInt
getLong
getResultDataType
getResultMetadata
getStringBuffer
invoke
isResultNull

Chapter 15: Joiner Transformation

Joiner Transformation Overview
Advanced Properties for Joiner Transformations
Joiner Transformation Ports
Join Types
Normal Join
Master Outer Join
Detail Outer Join
Full Outer Join
Sorted Input for a Joiner Transformation
<table>
<thead>
<tr>
<th>Chapter 16: Key Generator Transformation</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Generator Transformation Overview.</td>
<td>105</td>
</tr>
<tr>
<td>Soundex Strategy.</td>
<td>106</td>
</tr>
<tr>
<td>Soundex Strategy Properties.</td>
<td>106</td>
</tr>
<tr>
<td>String Strategy.</td>
<td>106</td>
</tr>
<tr>
<td>String Strategy Properties.</td>
<td>107</td>
</tr>
<tr>
<td>NYSIIS Strategy.</td>
<td>107</td>
</tr>
<tr>
<td>Key Generator Output Ports.</td>
<td>108</td>
</tr>
<tr>
<td>Configuring a Grouping Strategy.</td>
<td>108</td>
</tr>
<tr>
<td>Key Creation Properties.</td>
<td>108</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 17: Labeler Transformation</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labeler Transformation Overview.</td>
<td>110</td>
</tr>
<tr>
<td>Labeler Transformation Strategies.</td>
<td>110</td>
</tr>
<tr>
<td>Labeler Transformation Ports.</td>
<td>111</td>
</tr>
<tr>
<td>Character Labeling Operations.</td>
<td>111</td>
</tr>
<tr>
<td>Character Labeling Properties.</td>
<td>111</td>
</tr>
<tr>
<td>Token Labeling Operations.</td>
<td>112</td>
</tr>
<tr>
<td>Token Sets.</td>
<td>113</td>
</tr>
<tr>
<td>Token Labeling Properties.</td>
<td>113</td>
</tr>
<tr>
<td>Configuring a Labeling Strategy.</td>
<td>114</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 18: Lookup Transformation</th>
<th>115</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup Transformation Overview.</td>
<td>115</td>
</tr>
<tr>
<td>Developing a Lookup Transformation.</td>
<td>115</td>
</tr>
<tr>
<td>Runtime Properties for Lookup Transformations.</td>
<td>116</td>
</tr>
<tr>
<td>Advanced Properties for Flat File Lookup Transformations.</td>
<td>116</td>
</tr>
<tr>
<td>Advanced Properties for Relational Lookup Transformations.</td>
<td>117</td>
</tr>
<tr>
<td>Lookup Condition.</td>
<td>118</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>19</td>
<td>Match Transformation</td>
</tr>
<tr>
<td>20</td>
<td>Merge Transformation</td>
</tr>
<tr>
<td>21</td>
<td>Parser Transformation</td>
</tr>
<tr>
<td>22</td>
<td>Sorter Transformation</td>
</tr>
</tbody>
</table>
## Table of Contents

**Chapter 23: SQL Transformation** .................................................. 144

SQL Transformation Overview. .................................................. 144
Advanced Properties. .................................................. 144
SQL Transformation Ports. .................................................. 145
  - Input Ports. .................................................. 146
  - Output Ports. .................................................. 146
  - Pass-through Ports. .................................................. 147
  - SQLERROR Port. .................................................. 147
  - Number of Rows Affected. .................................................. 147
SQL Transformation Query. .................................................. 148
  - Define the SQL Query. .................................................. 148
  - Input Row to Output Row Cardinality. .................................................. 149
  - Query Statement Processing. .................................................. 150
  - Port Configuration. .................................................. 150
  - Maximum Output Row Count. .................................................. 150
  - Error Rows. .................................................. 150
  - Continue on SQL Error. .................................................. 152
SQL Transformation Example. .................................................. 152
  - Logical Data Object Mapping. .................................................. 152
  - Salary Table. .................................................. 153
  - Employee Table. .................................................. 153
  - SQL Transformation .................................................. 154
  - Output. .................................................. 155

**Chapter 24: Standardizer Transformation** ........................................ 157

Standardizer Transformation Overview. ........................................ 157
Standardization Strategies. .................................................. 157
Standardization Properties. .................................................. 158
Configuring a Standardization Strategy. ........................................ 159

**Chapter 25: Union Transformation** ............................................... 160

Union Transformation Overview. .................................................. 160
Groups and Ports. .................................................. 160
Union Transformation Advanced Properties. ........................................ 161
Union Transformation Processing. .................................................. 161
Creating a Union Transformation. .................................................. 162
Preface

The Informatica Developer Transformation Guide is written for data quality and data services developers. This guide assumes that you have an understanding of data quality concepts, flat file and relational database concepts, and the database engines in your environment. This guide also assumes that you are familiar with the concepts presented in the Informatica Developer User Guide.

Informatica Resources

Informatica Customer Portal

As an Informatica customer, you can access the Informatica Customer Portal site at http://mysupport.informatica.com. The site contains product information, user group information, newsletters, access to the Informatica customer support case management system (ATLAS), the Informatica How-To Library, the Informatica Knowledge Base, the Informatica Multimedia Knowledge Base, Informatica Product Documentation, and access to the Informatica user community.

Informatica Documentation

The Informatica Documentation team takes every effort to create accurate, usable documentation. If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at info_documentation@informatica.com. We will use your feedback to improve our documentation. Let us know if we can contact you regarding your comments.

The Documentation team updates documentation as needed. To get the latest documentation for your product, navigate to Product Documentation from http://mysupport.informatica.com.

Informatica Web Site

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

Informatica How-To Library

As an Informatica customer, you can access the Informatica How-To Library at http://mysupport.informatica.com. The How-To Library is a collection of resources to help you learn more about Informatica products and features. It includes articles and interactive demonstrations that provide solutions to common problems, compare features and behaviors, and guide you through performing specific real-world tasks.
Informatica Knowledge Base

As an Informatica customer, you can access the Informatica Knowledge Base at http://mysupport.informatica.com. Use the Knowledge Base to search for documented solutions to known technical issues about Informatica products. You can also find answers to frequently asked questions, technical white papers, and technical tips. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team through email at KB_Feedback@informatica.com.

Informatica Multimedia Knowledge Base

As an Informatica customer, you can access the Informatica Multimedia Knowledge Base at http://mysupport.informatica.com. The Multimedia Knowledge Base is a collection of instructional multimedia files that help you learn about common concepts and guide you through performing specific tasks. If you have questions, comments, or ideas about the Multimedia Knowledge Base, contact the Informatica Knowledge Base team through email at KB_Feedback@informatica.com.

Informatica Global Customer Support

You can contact a Customer Support Center by telephone or through the Online Support. Online Support requires a user name and password. You can request a user name and password at http://mysupport.informatica.com.

Use the following telephone numbers to contact Informatica Global Customer Support:

<table>
<thead>
<tr>
<th>North America / South America</th>
<th>Europe / Middle East / Africa</th>
<th>Asia / Australia</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Toll Free</strong></td>
<td><strong>Toll Free</strong></td>
<td><strong>Toll Free</strong></td>
</tr>
<tr>
<td>Brazil: 0800 891 0202</td>
<td>France: 00800 4632 4357</td>
<td>Australia: 1 800 151 830</td>
</tr>
<tr>
<td>Mexico: 001 888 209 8853</td>
<td>Germany: 00800 4632 4357</td>
<td>New Zealand: 1 800 151 830</td>
</tr>
<tr>
<td>North America: +1 877 463 2435</td>
<td>Italy: 800 915 985</td>
<td>Singapore: 001 800 4632 4357</td>
</tr>
<tr>
<td><strong>Standard Rate</strong></td>
<td>Portugal: 800 208 360</td>
<td><strong>Standard Rate</strong></td>
</tr>
<tr>
<td>North America: +1 650 653 6332</td>
<td>Spain: 900 813 166</td>
<td>India: +91 80 4112 5738</td>
</tr>
<tr>
<td></td>
<td>Switzerland: 00800 4632 4357</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or 0800 463 200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>United Kingdom: 00800 4632 4357</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or 0800 023 4632</td>
<td></td>
</tr>
<tr>
<td><strong>Standard Rate</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belgium: +31 30 6022 797</td>
<td></td>
<td></td>
</tr>
<tr>
<td>France: 0805 804632</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany: 01805 702702</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netherlands: 030 6022 797</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Working with Transformations

Working with 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 mapplet.
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, and it maintains the row type.
You can connect multiple transformations to the same downstream transformation or transformation input group if all transformations in the upstream branches are passive. The transformation that originates the branch can be active or passive.
Multi-Strategy Transformations

You can define multiple transformation strategies in the following data quality transformations:

- Case
- Decision
- Key Generator
- Labeler
- Match
- Merge
- Parser
- Standardizer

You can assign a different set of input and output ports to each strategy in the transformation. The transformation stores the strategies you define in a single transformation object.

**Note:** Use the Dependencies view to view the inputs and outputs that each strategy uses.

Transformation Descriptions

The Developer tool contains common and data quality transformations. Common transformations are available in Informatica Data Quality and Informatica Data Services. Data quality transformations are available in Informatica Data Quality.

The following table describes each transformation:

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregator</td>
<td>Active/Connected</td>
<td>Performs aggregate calculations.</td>
</tr>
<tr>
<td>Custom Data</td>
<td>Active or Passive/Connected</td>
<td>Calls a procedure in a shared library or DLL.</td>
</tr>
<tr>
<td>Expression</td>
<td>Passive/Connected</td>
<td>Calculates a value.</td>
</tr>
<tr>
<td>Filter</td>
<td>Active/Connected</td>
<td>Filters data.</td>
</tr>
<tr>
<td>Input</td>
<td>Passive/Connected</td>
<td>Defines mapplet input rows.</td>
</tr>
<tr>
<td>Java</td>
<td>Active or Passive/Connected</td>
<td>Executes user logic coded in Java. The byte code for the user logic is stored in the repository.</td>
</tr>
<tr>
<td>Joiner</td>
<td>Active/Connected</td>
<td>Joins data from different databases or flat file systems.</td>
</tr>
<tr>
<td>Lookup</td>
<td>Active or Passive/Connected or Unconnected</td>
<td>Look up and return data from a flat file, relational table, view, or synonym.</td>
</tr>
<tr>
<td>Transformation</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Output</td>
<td>Passive/Connected</td>
<td>Defines mapplet output rows.</td>
</tr>
<tr>
<td>Rank</td>
<td>Active/Connected</td>
<td>Limits records to a top or bottom range.</td>
</tr>
<tr>
<td>Router</td>
<td>Active/Connected</td>
<td>Routes data into multiple transformations based on group conditions.</td>
</tr>
<tr>
<td>Sorter</td>
<td>Active/Connected</td>
<td>Sorts data based on a sort key.</td>
</tr>
<tr>
<td>SQL</td>
<td>Active or Passive/Connected</td>
<td>Executes SQL queries against a database.</td>
</tr>
<tr>
<td>Union</td>
<td>Active/Connected</td>
<td>Merges data from different databases or flat file systems.</td>
</tr>
<tr>
<td>Update Strategy</td>
<td>Active/Connected</td>
<td>Determines whether to insert, delete, update, or reject rows.</td>
</tr>
</tbody>
</table>

### 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. Create a non-reusable transformation as part of a mapping or mapplet. Or, create a reusable transformation to 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 transformations and mapping or mapplet outputs. Drag one or more ports to other ports to link them in the mapping or mapplet.

### 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.

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, and the total and average salaries you calculate through this transformation. For this reason, 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:

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Expression</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregator</td>
<td>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.</td>
<td>Result of an aggregate calculation for a port.</td>
</tr>
<tr>
<td>Expression</td>
<td>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.</td>
<td>Result of a row-level calculation for a port.</td>
</tr>
<tr>
<td>Filter</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Rank</td>
<td>Sets the conditions for rows included in a rank. For example, you can rank the top 10 salespeople who are employed with the organization.</td>
<td>Result of a condition or calculation for a port.</td>
</tr>
<tr>
<td>Router</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Update Strategy</td>
<td>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.</td>
<td>Numeric code for update, insert, delete, or reject. The transformation applies this value to each row passed through it.</td>
</tr>
</tbody>
</table>

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.
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.

### 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, the Parser transformation requires that you choose either token parsing mode or pattern parsing mode when you create the transformation.

After you create a transformation, it appears in the editor. Some transformations contain predefined ports and groups. Other transformations are empty.
CHAPTER 2

Address Validator Transformation

This chapter includes the following topics:
- Address Validator Transformation Overview, 7
- Address Validator Transformation Input Port Groups, 8
- Address Validator Transformation Output Port Groups, 9
- CASS and SERP Certification Reports, 10
- Formatted Addresses in the Address Validator Transformation, 11
- Address Validator Status Ports, 11
- Address Validator Transformation Advanced Properties, 17
- Address Validator Transformation General Settings, 19
- Configuring an Address Validator Transformation, 20
- Adding Ports to the Address Validator Transformation, 21
- Creating User-Defined Templates, 21
- Defining Address Validator Models, 21

Address Validator Transformation Overview

The Address Validator transformation examines input address data and creates columns with corrected data and validation information.

By default, the Address Validator transformation is a passive transformation, but setting the value of Execution Instances property higher than one makes the transformation active.

The Address Validation transformation compares input address data with address reference data to determine the accuracy of input addresses and fix errors in those addresses. You can validate and update address data in the following ways:
- Compare input addresses to address reference data provided by Informatica.
- Generate detailed status reports on the validity of each input address, its deliverable status, and the nature of any errors or ambiguities it contains.
- Fix errors and complete partial address records using information from the reference data. To fix an address, the transformation must find a positive match with an address in the reference data.
- Write output addresses in an envelope-ready format. You define the format when you select the output ports on the transformation.
• Add information that does not appear in the standard address but that helps postal delivery, such as Geocoding information.

**Address Reference Data**

An address reference data set contains data that describes all deliverable addresses in a country. The address validation process searches the reference data set for the address that most closely resembles the input address data. If the process finds a close match in the reference data set, it writes new values for any incorrect or incomplete data values. The process creates a set of alphanumeric codes that describe the type of match found between the input address and the reference addresses.

Address validation can also restructure the address, and it can add information that is absent from the input address, such as a four-digit ZIP code suffix for a United States address.

The address reference data sets read by the Address Validator transformation do not install with Informatica applications. You must download the address reference data sets separately and install them using the Data Quality Content Installer. Contact your Informatica Administrator user for information on the address reference data sets available on your system.

**Transformation Ports**

When you configure an Address Validator transformation, you add input and output ports by selecting them from predefined templates. The input ports correspond to address fields, and the output ports correspond to address fields, delivery information, and validation information.

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

**Templates and Models**

The Address Validator transformation contains predefined templates that contain the input and output ports you can use. You can also create user-defined templates to create groups of address ports that you plan to reuse.

By default, Address Validator transformations do not contain default input and output ports. However, you can define Address Validator models to specify the input and output ports included for new Address Validator transformations.

---

**Address Validator Transformation Input Port Groups**

Address Validator transformation ports are organized in groups. Before you can connect address data to input ports on the transformation, you must browse the input groups and select the ports that correspond to the structure and content of the input data. You add the selected ports to a template in the transformation.

The transformation has the following input port groups:

- **Discrete.** A discrete port accepts a single address element. Connect address fields to these ports if each field represents a unique address element, such as house number, apartment number, street name, city, state, or ZIP code.

- **MultiLine.** A multiline port accepts a string containing multiple address elements. Connect address fields to these ports if each field represents multiple address elements, for example when Address Line 1 contains “100 Main Street” and Address Line 2 contains “New York, NY 10001.”

- **Hybrid.** A hybrid port accepts a string containing single or multiple address elements. Connect address fields to these ports if the address contains a mix of discrete and multi-line elements. A common format combines house number, street name, and apartment information in a single field and uses discrete fields for city, state, and ZIP code.
Note: Select ports from one input port group only.

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.

Address Validator Transformation Output Port Groups

Address Validator transformation ports are organized in groups. The Address Validator transformation has predefined output ports. Before you can connect the Address Validator transformation output ports to other transformations or targets, you must decide the format your output addresses will take. You must also decide on the information you need about the data quality of the input addresses.

Browse the output groups and select the ports that match your data requirements. Add the selected ports to a template in the transformation.

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.** Use to write street address elements, such as house number, apartment number, and street name, to separate ports.
- **LastLine Elements.** Use to write locality information, such as post codes and city and state names, to separate ports.
- **Geo Coding.** Use to write geo-coding information, such as latitude and longitude coordinates.
- **US Specific.** Use to write additional data on United States addresses that can help postal delivery, such as DPV data.
- **Canada Specific.** Use to write SERP report data for Canadian addresses.
- **UK Supplementary.** Use to write Delivery Point Suffix data for United Kingdom addresses.
- **US Supplementary.** Use to write supplemental data such as county FIPS and state FIPS codes for United States addresses.
- **Country.** Use to write the country name or ISO country code.
- **Status Info.** Use to write information on the quality of each input address.
- **Formatted Address Line.** Use to write addresses that are formatted for mailing. Formatted lines are not linked to specific types of address information. The transformation treats each address uniquely and writes each address line to the first available port in this group.
- **Residue.** Use for unrecognized elements in the input address.

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

The following port groups are available in the Advanced Model only:

- **Contact Elements.** Use to write information related to business contacts, such as salutations and job titles.
- **ID Elements.** Use to write RecordID and TransactionKey data.
CASS and SERP Certification Reports

You can configure the Address Validator transformation to generate Coding Accuracy Support System (CASS) reports for United States addresses and Software Evaluation and Recognition Program (SERP) reports for Canadian addresses.

CASS and SERP reports verify that the address validation operations performed on the source address data meet the standards of the USPS and Canada Post. United States and Canadian addresses qualify for postal discount when they are CASS and SERP certified.

Adding a Certification Report to the Address Validator Transformation

To add a certification report to the Address Validator transformation, perform the following steps:

1. On the General Settings view, set the Mode option to Certified.
2. On the Reports view, select the types of reports to generate.
3. For each report type that you select, enter the report details.

Table 1. CASS Report Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Filename</td>
<td>File name for the report that the transformation creates. By default, the transformation creates the report in the shared/bin directory of the Informatica installation on the Data Integration Service machine. To write the report file to another location on the machine, enter a full path and file name.</td>
</tr>
<tr>
<td>List Name/ID</td>
<td>Name or identification number of the address list.</td>
</tr>
<tr>
<td>List Processor Name</td>
<td>Name of the organization that performs the address validation.</td>
</tr>
<tr>
<td>Name/Address</td>
<td>Postal name and address of the organization that performs the address validation.</td>
</tr>
</tbody>
</table>

Table 2. SERP Report Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Filename</td>
<td>File name for the report that the transformation creates. By default, the transformation creates the report in the shared/bin directory of the Informatica installation on the Data Integration Service machine. To write the report file to another location on the machine, enter a full path and file name.</td>
</tr>
<tr>
<td>Customer CPC Number</td>
<td>Customer number issued by the Canada Post Corporation to the organization that performs the address validation.</td>
</tr>
<tr>
<td>Customer Name/Address</td>
<td>Name and address of the organization that performs the address validation.</td>
</tr>
</tbody>
</table>
Formatted Addresses in the Address Validator Transformation

The **Formatted Address Line** output port group eliminates the need to manually parse address values into postal formats. Use the ports in these output groups to create mailing addresses.

The Address Validator transformation parses the data from each address and writes each address line to the first available port in this group, so that no blank lines appear in an output address. You can also select the **CompleteAddress** port, which writes the formatted address on a single port.

Formatted Address Line ports can carry any type of address information, and the ports are named generically.

**Formatted Address Example: United States**

The Address Validator transformation uses up to four lines to create an address in the standard United States Postal Service format:

<table>
<thead>
<tr>
<th>Transformation Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormattedAddressLine1</td>
<td>Company or organization name</td>
</tr>
<tr>
<td>FormattedAddressLine2</td>
<td>Urbanization (if applicable, for example in Puerto Rican addresses)</td>
</tr>
<tr>
<td>FormattedAddressLine3</td>
<td>Street address</td>
</tr>
<tr>
<td>FormattedAddressLine4</td>
<td>City, State, and ZIP code</td>
</tr>
</tbody>
</table>

**Address Validator Status Ports**

The Address Validator transformation writes status information on the address elements it reads and writes on input and output ports.

The following status ports are available:

**ElementInputStatus**

Select from the StatusInfo port group in the Basic Model. The values on this port represent the type of match found between the input address element and the reference data.

**ElementRelevance**

Select from the StatusInfo port group in the Basic Model. The values on this port identify the address elements that are necessary for local address delivery.

**ElementResultStatus**

Select from the StatusInfo port group in the Basic Model. The values on this port describe any edit made to the input data during processing.

**GeoCodingStatus**

Select from the Geo Coding port group in the Basic Model. The values on this port describe the level of geocoding information returned for the input address.
MailabilityScore

Select from the StatusInfo port group in the Basic Model. The values on this port represent the overall deliverability of an input address.

MatchCode

Select from the StatusInfo port group in the Basic Model. The values on this port describe the results of the validation operation on the input address.

Add the Status Info ports to the address template to view the status of input and output address elements. If you have selected Geo Coding output ports, you can also add the GeoCodingStatus port to the address template.

ElementInputStatus Output Port Values

ElementInputStatus is a twenty-character string in which each location on the string represents a different input address element. The value for a character represents the type of processing performed on the related address element.

Find this port in the Status Info port group.

The following table describes the address elements identified by each character in the string:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PostalCode level 0</td>
</tr>
<tr>
<td>2</td>
<td>PostalCode level 1 (for example, United States ZIP+4 Code)</td>
</tr>
<tr>
<td>3</td>
<td>Locality level 0</td>
</tr>
<tr>
<td>4</td>
<td>Locality level 1 (for example, Urbanization, Dependent Locality)</td>
</tr>
<tr>
<td>5</td>
<td>Province level 0</td>
</tr>
<tr>
<td>6</td>
<td>Province level 1 (for example, Sub-Province)</td>
</tr>
<tr>
<td>7</td>
<td>Street level 0</td>
</tr>
<tr>
<td>8</td>
<td>Street level 1 (for example, Dependent Street)</td>
</tr>
<tr>
<td>9</td>
<td>Number level 0</td>
</tr>
<tr>
<td>10</td>
<td>Number level 1</td>
</tr>
<tr>
<td>11</td>
<td>Delivery service level 0 (for example, PO Box, GPO, Packstation, Private Bags)</td>
</tr>
<tr>
<td>12</td>
<td>Delivery service level 1</td>
</tr>
<tr>
<td>13</td>
<td>Building level 0</td>
</tr>
<tr>
<td>14</td>
<td>Building level 1</td>
</tr>
<tr>
<td>15</td>
<td>SubBuilding level 0</td>
</tr>
<tr>
<td>16</td>
<td>SubBuilding level 1</td>
</tr>
<tr>
<td>17</td>
<td>Organization level 0</td>
</tr>
</tbody>
</table>
Value | Description
---|---
18 | Organization level 1
18 | Country level 0
20 | Country level 1 (for example, Territory)

The following table describes the type of processing performed on each address element represented in ElementInputStatus:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Matched with changes (data inserted or deleted)</td>
</tr>
<tr>
<td>4</td>
<td>Matched with errors on this address element</td>
</tr>
<tr>
<td>3</td>
<td>Bad address element. Input is copied and not corrected.</td>
</tr>
<tr>
<td>2</td>
<td>Unchecked (no reference data)</td>
</tr>
<tr>
<td>1</td>
<td>Address element not found. No processing performed.</td>
</tr>
<tr>
<td>0</td>
<td>Address element empty. No processing performed.</td>
</tr>
</tbody>
</table>

**ElementRelevance Output Port Values**

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

ElementRelevance values identify the address elements that are relevant to the postal carrier that delivers the address. All address elements with a value of 1 must be present for an output address to be deemed valid by the local postal carrier. ElementRelevance values are meaningful for address elements with a MatchCode value of C or V.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Address element is not needed by the local postal carrier.</td>
</tr>
<tr>
<td>1</td>
<td>Address element is relevant to the local postal carrier.</td>
</tr>
</tbody>
</table>

**ElementResultStatus Output Port Values**

ElementResultStatus is a twenty-character string in which each location on the string represents a different input address element. The value for a character describes any edit made to the input data during processing.

Find this port in the Status Info port group.

The following table describes the address elements identified by each character in the string:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PostalCode level 0</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>2</td>
<td>PostalCode level 1 (for example, United States ZIP+4 Code)</td>
</tr>
<tr>
<td>3</td>
<td>Locality level 0</td>
</tr>
<tr>
<td>4</td>
<td>Locality level 1 (for example, Urbanization, Dependent Locality)</td>
</tr>
<tr>
<td>5</td>
<td>Province level 0</td>
</tr>
<tr>
<td>6</td>
<td>Province level 1 (for example, Sub-Province)</td>
</tr>
<tr>
<td>7</td>
<td>Street level 0</td>
</tr>
<tr>
<td>8</td>
<td>Street level 1 (for example, Dependent Street)</td>
</tr>
<tr>
<td>9</td>
<td>Number level 0</td>
</tr>
<tr>
<td>10</td>
<td>Number level 1</td>
</tr>
<tr>
<td>11</td>
<td>Delivery service level 0 (for example, PO Box, GPO, Packstation, Private Bags)</td>
</tr>
<tr>
<td>12</td>
<td>Delivery service level 1</td>
</tr>
<tr>
<td>13</td>
<td>Building level 0</td>
</tr>
<tr>
<td>14</td>
<td>Building level 1</td>
</tr>
<tr>
<td>15</td>
<td>SubBuilding level 0</td>
</tr>
<tr>
<td>16</td>
<td>SubBuilding level 1</td>
</tr>
<tr>
<td>17</td>
<td>Organization level 0</td>
</tr>
<tr>
<td>18</td>
<td>Organization level 1</td>
</tr>
<tr>
<td>19</td>
<td>Country level 0</td>
</tr>
<tr>
<td>20</td>
<td>Country level 1 (for example, Territory)</td>
</tr>
</tbody>
</table>

The following table describes the type of edit made to each address element represented in ElementResultStatus:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Address element empty.</td>
</tr>
<tr>
<td>1</td>
<td>Input is not validated and not edited. Source data is used.</td>
</tr>
<tr>
<td>2</td>
<td>Input is not validated but is standardized</td>
</tr>
<tr>
<td>3</td>
<td>Input is validated but contains some invalid data and is not edited. Input is copied, not corrected.</td>
</tr>
<tr>
<td>4</td>
<td>Input is validated but not edited due to lack of reference data</td>
</tr>
<tr>
<td>5</td>
<td>Input is validated but not edited as multiple candidate matches are available</td>
</tr>
</tbody>
</table>
Value | Description
--- | ---
6 | Input is validated and edited by eliminating an input value
7 | Input is validated and edited based on reference data
8 | Input is validated and edited by adding value from reference data
9 | Input is validated and not edited. Delivery status is not clear.
C | Input is validated and verified but out-of-date name is edited
D | Input is validated and verified but exonym is edited to official name
E | Input is validated and verified but standardized. This value is set if the input fully matches a language alternative.
F | Input is validated, verified, and not edited. Perfect match.

**MailabilityScore Output Port Values**

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

The Address Validator estimates the deliverability of an address and writes a figure representing this estimate to the MailabilityScore column. Consult this figure if the MatchCode score is in the range I1-I4.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Fully certain of delivery</td>
</tr>
<tr>
<td>4</td>
<td>Almost certain of delivery</td>
</tr>
<tr>
<td>3</td>
<td>Confident of delivery</td>
</tr>
<tr>
<td>2</td>
<td>Reasonably confident of delivery</td>
</tr>
<tr>
<td>1</td>
<td>Delivery is doubtful</td>
</tr>
<tr>
<td>0</td>
<td>Undeliverable</td>
</tr>
</tbody>
</table>

**MatchCode Output Port Values**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V4</td>
<td>Verified. Input data correct. All postally relevant elements were checked and input matched perfectly.</td>
</tr>
<tr>
<td>V3</td>
<td>Verified. Input data correct, but some or all elements were standardized, or input contains outdated names.</td>
</tr>
<tr>
<td>V2</td>
<td>Verified. Input data correct, but some elements could not be verified because of incomplete reference data.</td>
</tr>
<tr>
<td>V1</td>
<td>Verified. Input data correct, but user standardization has negatively impacted deliverability.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>C4</td>
<td>Corrected. All postally relevant elements have been checked.</td>
</tr>
<tr>
<td>C3</td>
<td>Corrected. Some elements could not be checked.</td>
</tr>
<tr>
<td>C2</td>
<td>Corrected, but the delivery status is unclear due to absent reference data.</td>
</tr>
<tr>
<td>C1</td>
<td>Corrected, but the delivery status is unclear because user standardization introduced errors.</td>
</tr>
<tr>
<td>I4</td>
<td>Data could not be corrected completely, but is very likely to be deliverable. There is a single match with an address in the reference data.</td>
</tr>
<tr>
<td>I3</td>
<td>Data could not be corrected completely, but is very likely to be deliverable. There are multiple matches with addresses in the reference data.</td>
</tr>
<tr>
<td>I2</td>
<td>Data could not be corrected. The address may be delivered.</td>
</tr>
<tr>
<td>I1</td>
<td>Data could not be corrected and not likely to be delivered.</td>
</tr>
<tr>
<td>Q3</td>
<td>FastCompletion Status. Address suggestions are available.</td>
</tr>
<tr>
<td>Q2</td>
<td>FastCompletion Status. Suggested address is complete but is mixed with elements from the input address.</td>
</tr>
<tr>
<td>Q1</td>
<td>FastCompletion Status. Suggested address is not complete. More information required.</td>
</tr>
<tr>
<td>Q0</td>
<td>FastCompletion Status. Insufficient input available to generate suggestions.</td>
</tr>
<tr>
<td>RA</td>
<td>Country recognized from ForceCountryISO3 Setting.</td>
</tr>
<tr>
<td>R9</td>
<td>Country recognized from DefaultCountryISO3 Setting.</td>
</tr>
<tr>
<td>R8</td>
<td>Country recognized from name without errors.</td>
</tr>
<tr>
<td>R7</td>
<td>Country recognized from name with errors.</td>
</tr>
<tr>
<td>R6</td>
<td>Country recognized from territory.</td>
</tr>
<tr>
<td>R5</td>
<td>Country recognized from province.</td>
</tr>
<tr>
<td>R4</td>
<td>Country recognized from major town.</td>
</tr>
<tr>
<td>R3</td>
<td>Country recognized from format.</td>
</tr>
<tr>
<td>R2</td>
<td>Country recognized from script.</td>
</tr>
<tr>
<td>R1</td>
<td>Country not recognized. Multiple candidate matches.</td>
</tr>
<tr>
<td>R0</td>
<td>Country not recognized.</td>
</tr>
<tr>
<td>S4</td>
<td>Parsed perfectly.</td>
</tr>
<tr>
<td>S3</td>
<td>Parsed with multiple results.</td>
</tr>
<tr>
<td>S2</td>
<td>Parsed with Errors. Elements changed position.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>S1</td>
<td>Parsing Error. Input Format Mismatch.</td>
</tr>
<tr>
<td>N1</td>
<td>Validation Error. No validation performed because country was not recognized.</td>
</tr>
<tr>
<td>N2</td>
<td>Validation Error. No validation performed because required reference database is not available.</td>
</tr>
<tr>
<td>N3</td>
<td>Validation Error. No validation performed because country could not be unlocked.</td>
</tr>
<tr>
<td>N4</td>
<td>Validation Error. No validation performed because reference database is corrupt or in wrong format.</td>
</tr>
<tr>
<td>N5</td>
<td>Validation Error. No validation performed because reference database is out of date.</td>
</tr>
</tbody>
</table>

**GeoCodingStatus Output Port Values**

The following table describes the GeoCodingStatus output port values. Find this port in the Geo Coding port group. Select this port if you have installed geocoding reference data for an input address country.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGCU</td>
<td>Geocoding database not unlocked</td>
</tr>
<tr>
<td>EGCN</td>
<td>Geocoding database not found</td>
</tr>
<tr>
<td>EGCC</td>
<td>Geocoding database corrupt</td>
</tr>
<tr>
<td>EGC0</td>
<td>No Geocode available</td>
</tr>
<tr>
<td>EGC5</td>
<td>Geocode with postal code level accuracy</td>
</tr>
<tr>
<td>EGC6</td>
<td>Geocode with locality level accuracy</td>
</tr>
<tr>
<td>EGC7</td>
<td>Geocode with street level accuracy</td>
</tr>
<tr>
<td>EGC8</td>
<td>Geocode with house number accuracy (approximated)</td>
</tr>
<tr>
<td>EGC9</td>
<td>Geocode with house number accuracy</td>
</tr>
</tbody>
</table>

**Address Validator Transformation Advanced Properties**

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

Configure the following properties on the Advanced view:

**Matching Alternatives**

Determines whether the transformation considers alternative terms when performing validation. Select one of the following options:
None
- Validates no alternative terms.

Synonyms only
- Validates known synonyms for terms. For example, "Londres" is a foreign language synonym for "London."

Archives only
- Validates historically alternative terms. For example, "Constantinople" is a historical alternative for the city name "Istanbul."

All
- This is the default option. Validates all known alternatives.

Matching Scope
- Determines the amount of data that the transformation validates. Select one of the following options:
  - Locality
    - Validates province, locality, and postcode data.
  - Street
    - Validates street address data in addition to data that the Locality option validates.
  - Delivery Point
    - Validates delivery point address data in addition to data that the Street option validates.

Optimization Level
- Determines the balance of processing speed and validation accuracy. Select one of the following options:
  - Narrow
    - Performs the fastest validations but may return less accurate results.
  - Standard
    - Equally balances search speed and accuracy.
  - Wide
    - Performs the most accurate validations but takes longer to process.

Input Format Type
- Describes the information contained in the input data. Select the option that best describes the input data.

Output Format Type
- Describes the information contained in the output data. Select the option that best describes the output data you require.

Input Format With Country
- Determines whether the input contains country data.

Output Format With Country
- Determines if the output includes country data.

Country Type
- Determines the format of the output data. Different countries format address data in different ways.

Country of Origin
- Describes the country that data originates from.
Preferred Script
Determines the encoding that the transformation uses for output data. Select one of the following options:

- **Database**
  Uses Latin I or ASCII characters, based on reference database format.

- **Postal Admin**
  Uses Latin I or ASCII characters, based on postal service preferences.

- **Postal Admin (Alt)**
  Uses Latin I or ASCII characters, based on postal services alternative preferences.

- **ASCII (Simplified)**
  Uses ASCII characters.

- **ASCII (Extended)**
  Uses ASCII characters with expansion of special characters. For example, Ö transliterates to OE.

- **Latin**
  Uses Latin I characters.

- **Latin (Alt)**
  Uses Latin I characters with alternative transliteration.

Preferred Language
Determines the language of the output data. Select one of the following options:

- **Database**
  Uses the language derived from reference data validation for each address. This is the default option.

- **English**
  Uses English output, if available for the address.

- **ALTERNATIVE 1,2,3**
  Uses alternative languages for multilingual countries such as Belgium or Switzerland.

Execution Instances
Determines the number of processes that the transformation uses at runtime. Setting the number higher than one causes the transformation to run in active mode.

Incrementing the number of execution instances can improve validation performance, provided the number does not exceed the number of CPU cores used by the Data Integration Service. If you need to run simultaneous address validation mappings or if the Data Integration Service experiences heavy loads, set the number of execution instances lower than the number of CPU cores.

---

Address Validator Transformation General Settings

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

Configure the following properties on the **General Settings** view:

**Default Country**
Sets the reference data set that the transformation uses if it cannot determine country information from the input address data. Select None if your data includes country information.
Force Country

Forces the transformation to use the reference data set regardless of country-specific information an address may contain.

Line Separator

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

Casing Style

Sets the character case style for output data. Select Mixed to follow the reference data standard for initial capital letters. Select Preserved to preserve the input case style.

Mode

Determines the type of validation that the transformation performs. The default mode is Batch. You can choose from the following options:

Country Recognition

Determines a destination country for the postal address without performing address validation.

Parse

Parses data into address fields without performing validation.

Batch

Performs address validation.

Certified

Performs validation that meets postal service certification standards.

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 view, 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.
Aggregator Transformation

This chapter includes the following topics:
- Aggregator Transformation Overview, 23
- Developing an Aggregator Transformation, 24
- Aggregator Transformation Ports, 24
- Advanced Properties for Aggregator Transformations, 24
- Aggregate Caches, 25
- Aggregate Expressions, 26
- Group By Ports, 27
- Sorted Input for an Aggregator Transformation, 28
- Creating a Reusable Aggregator Transformation, 29
- Creating a Non-Reusable Aggregator Transformation, 30
- Tips for Aggregator Transformations, 30
- Troubleshooting Aggregator Transformations, 30

Aggregator Transformation Overview

Use the Aggregator transformation to perform aggregate calculations, such as averages and sums. The Data Integration Service performs aggregate calculations as it reads and stores data group and row data in an aggregate cache. The Aggregator transformation is an active transformation.

The Aggregator transformation is unlike the Expression transformation, in that you use the Aggregator transformation to perform calculations on groups. The Expression transformation permits you to perform calculations on a row-by-row basis only.

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

You can enable incremental aggregation. When the Data Integration Service performs incremental aggregation, it passes source data through the mapping and uses historical cache data to perform aggregation calculations incrementally.
Developing an Aggregator Transformation

When you develop an Aggregator transformation, you need to consider factors, such as the type of calculations you want to perform and transformation performance.

Consider the following factors when you develop an Aggregator transformation:

- The expressions that you want to use in output ports to calculate data.
- Whether to use aggregate caches to process transformation data.
- Whether to use group by ports to define groups for aggregations, instead of performing the aggregation across all input data. Group by ports can increase performance.
- Whether to use sorted input so that the Data Integration Service performs aggregate calculations as it reads rows for a group.

Aggregator Transformation Ports

An Aggregator transformation has different port types that allow you to perform different transformation tasks, such as aggregate data and group data.

An Aggregator transformation has the following port types:

- **Input**
  - Receives data from upstream transformations.

- **Output**
  - Provides the return value of an expression. The expression can include non-aggregate expressions and conditional clauses. You can create multiple aggregate output ports.

- **Pass-Through**
  - Passes data unchanged.

- **Variable**
  - Used for local variables.

- **Group by**
  - Indicates how to create groups. The port can be any input, input/output, output, or variable port. When grouping data, the Aggregator transformation outputs the last row of each group unless otherwise specified.

Advanced Properties for Aggregator Transformations

Configure properties that help determine how the Data Integration Service processes data for the Aggregator transformation.

The following table describes the advanced properties for an Aggregator transformations:

<table>
<thead>
<tr>
<th>Cache Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local 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 aggregate caches.</td>
</tr>
</tbody>
</table>
If you have enabled incremental aggregation, the Data Integration Service creates a backup of the files each time you run the mapping. The cache directory must contain enough disk space for two sets of the files.

**Data Cache Size**
Data cache size for the transformation. Default is Auto.

**Index Cache Size**
Index cache size for the transformation. 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.

**Scope**
Specifies how the Data Integration Service applies the transformation logic to incoming data:
- Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions.
- All Input. Applies the transformation logic on all incoming data. The Data Integration Service drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source.

**Tracing Level**
Amount of detail displayed in the log for this transformation.
Default is normal.

The following table describes the tracing levels:

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terse</td>
<td>Logs initialization information and error messages and notification of rejected data.</td>
</tr>
<tr>
<td>Normal</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose Initialization</td>
<td>In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.</td>
</tr>
<tr>
<td>Verbose Data</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Aggregate 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 process the transformation. If the Data Integration Service requires more space, it stores overflow values in cache files.

You can configure the index cache and data cache in the Aggregator transformation.
The Data Integration Service uses memory to process an Aggregator transformation with sorted ports. It does not use cache memory. You do not need to configure cache memory for Aggregator transformations that use sorted ports.

Aggregate Expressions

The Developer tool allows aggregate expressions only in the Aggregator transformation.

An aggregate expression can include conditional clauses and non-aggregate functions. It can also include one aggregate function nested within another aggregate function, such as:

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

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

```
SUM( QUANTITY )
```

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

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

Aggregate Functions

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

The transformation language includes the following aggregate functions:

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

You must use these functions in an expression within an Aggregator transformation.

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:

\[
\text{SUM( COMMISSION, COMMISSION > QUOTA )}
\]

### Group By Ports

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

To define a group for the aggregate expression, select the appropriate input, input/output, 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 typically returns the last row of each group (or the last row received) with the result of the aggregation. However, if you specify a particular row to be returned (for example, by using the FIRST function), the Data Integration Service then returns the specified row.

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

If you send the following data through this Aggregator transformation:

<table>
<thead>
<tr>
<th>STORE_ID</th>
<th>ITEM</th>
<th>QTY</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>'battery'</td>
<td>3</td>
<td>2.99</td>
</tr>
<tr>
<td>101</td>
<td>'battery'</td>
<td>1</td>
<td>3.19</td>
</tr>
<tr>
<td>101</td>
<td>'battery'</td>
<td>2</td>
<td>2.59</td>
</tr>
<tr>
<td>101</td>
<td>'AAA'</td>
<td>2</td>
<td>2.45</td>
</tr>
<tr>
<td>201</td>
<td>'battery'</td>
<td>1</td>
<td>1.99</td>
</tr>
<tr>
<td>201</td>
<td>'battery'</td>
<td>4</td>
<td>1.59</td>
</tr>
<tr>
<td>301</td>
<td>'battery'</td>
<td>1</td>
<td>2.45</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>STORE_ID</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>'battery'</td>
</tr>
<tr>
<td>101</td>
<td>'AAA'</td>
</tr>
<tr>
<td>201</td>
<td>'battery'</td>
</tr>
<tr>
<td>301</td>
<td>'battery'</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>STORE_ID</th>
<th>ITEM</th>
<th>QTY</th>
<th>PRICE</th>
<th>SALES_PER_STORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>'battery'</td>
<td>2</td>
<td>2.59</td>
<td>17.34</td>
</tr>
<tr>
<td>101</td>
<td>'AAA'</td>
<td>2</td>
<td>2.45</td>
<td>4.90</td>
</tr>
<tr>
<td>201</td>
<td>'battery'</td>
<td>4</td>
<td>1.59</td>
<td>8.35</td>
</tr>
<tr>
<td>301</td>
<td>'battery'</td>
<td>1</td>
<td>2.45</td>
<td>2.45</td>
</tr>
</tbody>
</table>
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:

\[ \text{IIF}( \text{ITEM} = 'AAA battery', \text{battery}, \text{ITEM} ) \]

Default Values of Group By Ports

Define a default value for each port in the group to replace null input values. This allows the Data Integration Service to include null item groups in the aggregation.

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.

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:

<table>
<thead>
<tr>
<th>STORE_ID</th>
<th>ITEM</th>
<th>QTY</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>&quot;battery&quot;</td>
<td>3</td>
<td>2.99</td>
</tr>
<tr>
<td>101</td>
<td>&quot;battery&quot;</td>
<td>1</td>
<td>3.19</td>
</tr>
<tr>
<td>101</td>
<td>&quot;battery&quot;</td>
<td>2</td>
<td>2.59</td>
</tr>
<tr>
<td>201</td>
<td>&quot;battery&quot;</td>
<td>4</td>
<td>1.59</td>
</tr>
<tr>
<td>201</td>
<td>&quot;battery&quot;</td>
<td>1</td>
<td>1.99</td>
</tr>
</tbody>
</table>

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

Sorted Input Conditions

Certain conditions might prevent you from using sorted input.

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

- The aggregate expression uses nested aggregate functions.
- The transformation uses incremental aggregation.

If you use sorted input and do not sort data correctly, the Data Integration Service fails the mapping run.

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:

![Sorter transformation to sort data in ascending order by ITEM_NO](image)

The Sorter transformation sorts the data as follows:

<table>
<thead>
<tr>
<th>ITEM_NO</th>
<th>ITEM_NAME</th>
<th>QTY</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>345</td>
<td>Soup</td>
<td>4</td>
<td>2.95</td>
</tr>
<tr>
<td>345</td>
<td>Soup</td>
<td>1</td>
<td>2.95</td>
</tr>
<tr>
<td>345</td>
<td>Soup</td>
<td>2</td>
<td>3.25</td>
</tr>
<tr>
<td>546</td>
<td>Cereal</td>
<td>1</td>
<td>4.49</td>
</tr>
<tr>
<td>546</td>
<td>Cereal</td>
<td>2</td>
<td>5.25</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ITEM_NAME</th>
<th>QTY</th>
<th>PRICE</th>
<th>INCOME_PER_ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cereal</td>
<td>2</td>
<td>5.25</td>
<td>14.99</td>
</tr>
<tr>
<td>Soup</td>
<td>2</td>
<td>3.25</td>
<td>21.25</td>
</tr>
</tbody>
</table>

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. Configure aggregate expressions for output ports.
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 pallette to the editor. The transformation appears in the editor.
2. In the Properties view, edit the transformation name and the description.
3. In the Ports view, 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.

Troubleshooting Aggregator Transformations

You can troubleshoot Aggregator transformations.

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

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

- The aggregate expression contains nested aggregate functions.
- You use incremental aggregation.
- Source data is data driven.

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

This chapter includes the following topic:
- Association Transformation Overview, 32

**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.

Association transformation output ports must contain data of type String.

**Example: Associating Match Transformation Outputs**

The following table contains three records that could identify the same individual:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>ZIP</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David Jones</td>
<td>100 Admiral Ave.</td>
<td>New York</td>
<td>NY</td>
<td>10547</td>
<td>987-65-4321</td>
</tr>
<tr>
<td>2</td>
<td>Dennis Jones</td>
<td>1000 Alberta Ave.</td>
<td>New Jersey</td>
<td>NY</td>
<td>987-65-4321</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D. Jones</td>
<td>Admiral Ave.</td>
<td>New York</td>
<td>NY</td>
<td>10547-1521</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>SSN</th>
<th>Name and Address Cluster ID</th>
<th>Name and SSN Cluster ID</th>
<th>Association ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David Jones</td>
<td>100 Admiral Ave.</td>
<td>New York</td>
<td>NY</td>
<td>10547</td>
<td>987-65-4320</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Dennis Jones</td>
<td>1000 Alberta Ave.</td>
<td>New Jersey</td>
<td>NY</td>
<td></td>
<td>987-65-4320</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>D. Jones</td>
<td>Alberta Ave.</td>
<td>New York</td>
<td>NY</td>
<td>10547-1521</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

You can consolidate the duplicate record data in the Consolidation transformation.
CHAPTER 5

Case Converter Transformation

This chapter includes the following topics:
- Case Converter Transformation Overview, 34
- Case Strategy Properties, 34
- Configuring a Case Converter Strategy, 35

Case Converter Transformation Overview

The Case Converter transformation is a passive transformation that creates data uniformity by standardizing the case of strings in input data.

To improve data quality, use the Case Converter transformation to format data to similar character formats. You can select predefined case conversion types, such as uppercase, lowercase, toggle case, title case, and sentence case.

You can also use reference tables to control case conversion. Use reference tables to change the case of input strings to the values from the reference table "Valid" column. You can use reference tables when the case conversion type is Title Case or Sentence case.

Within a Case Converter transformation, you can create multiple case conversion strategies. Each strategy uses a single conversion type. The Case Converter transformation provides a wizard that you use to create strategies.

Case Strategy Properties

You can configure properties for case conversion strategies.

On the Strategies view, you can configure the following case conversion properties:

Reference Tables

Applies the capitalization format specified by a reference table. Applies only if the case conversion option is Title Case or Sentence case.

If a reference table match occurs at the start of a token, the next character in that 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.
Conversion Types

Defines the case conversion method that a strategy uses. You can apply the following case conversion types:

- **Uppercase.** Converts all letters to uppercase.
- **Lowercase.** Converts all letters to lowercase.
- **Toggle Case.** Converts lowercase letters to uppercase and uppercase letters to lowercase.
- **Title Case.** Capitalizes the first letter in each substring.
- **Sentence Case.** Capitalizes the first letter of the field data string.

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.

---

**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.
This chapter includes the following topics:

- Comparison Transformation Overview, 36
- Field Matching Strategies, 36
- Identity Matching Strategies, 39
- Configuring a Comparison Strategy, 39

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 mapplet. You can add multiple Comparison transformations to the mapplet. Use the Match transformation to define match comparisons in a single transformation. You can embed a matching mapplet 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 strategy 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 and divides the number of matching character pairs by the total number of character pairs.
Bigram Example

Consider the following strings:

- Larder
- Lerder

These strings yield the following Bigram groups:

<table>
<thead>
<tr>
<th>L</th>
<th>a</th>
<th>r</th>
<th>d</th>
<th>e</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>r</td>
<td>d</td>
<td>e</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

The highlighted groups indicate matches between the input strings. Note that the second occurrence of the string “er” within the string “Larder” is not matched, as there is no corresponding second occurrence of “er” in the string “Lerder”.

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 strategy 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
- Harlowes

The highlighted characters indicate the positions that the Hamming strategy 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 strategy 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 strategy 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 matching 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 strategy, 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 strategy considers character case when performing matching.

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 strategy 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 strategy 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 strategy:

- 1-999-9999
- 011-01-999-9991

The highlighted characters indicate the positions that the Reverse Hamming Distance strategy 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.

<table>
<thead>
<tr>
<th>Identity Matching Strategy</th>
<th>Match Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Identifies an address match.</td>
</tr>
<tr>
<td>Contact</td>
<td>Identifies a contact within an organization at a single location.</td>
</tr>
<tr>
<td>Corp Entity</td>
<td>Identifies an organization by its legal corporate name.</td>
</tr>
<tr>
<td>Division</td>
<td>Identifies an organization at an address.</td>
</tr>
<tr>
<td>Family</td>
<td>Identifies a family by a family name and address or telephone number.</td>
</tr>
<tr>
<td>Fields</td>
<td>Identifies custom fields that you select.</td>
</tr>
<tr>
<td>Household</td>
<td>Identifies members of the same family at the same residence.</td>
</tr>
<tr>
<td>Individual</td>
<td>Identifies an individual by name and either ID or date of birth.</td>
</tr>
<tr>
<td>Organization</td>
<td>Identifies an organization by name.</td>
</tr>
<tr>
<td>Person Name</td>
<td>Identifies a person by name.</td>
</tr>
<tr>
<td>Resident</td>
<td>Identifies a person at an address.</td>
</tr>
<tr>
<td>Wide Contact</td>
<td>Identifies a contact within an organization regardless of location.</td>
</tr>
<tr>
<td>Wide Household</td>
<td>Identifies members of the same family at regardless of location.</td>
</tr>
</tbody>
</table>

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 Strategy view of a Comparison transformation.

1. Select the Strategy view.
2. Select a comparison strategy from the Strategy section.
3. In the Inputs 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.
Consolidation Transformation

This chapter includes the following topic:
• Consolidation Transformation Overview, 40

Consolidation Transformation Overview

The Consolidation transformation is an active transformation that creates a single, consolidated record from records identified as duplicates by the Match transformation.

The Consolidation transformation reads the AssociationIDs created by the Association transformation to perform this task.
Custom Data Transformation

This chapter includes the following topics:

- Custom Data Transformation Overview, 41
- Custom Data Transformation Views, 42
- Ports Configuration, 47
- Creating a Custom Data Transformation, 48

Custom Data Transformation Overview

The Custom Data transformation is an active transformation that processes data in unstructured and semi-structured file formats, such as messaging formats, HTML pages and PDF documents. It also transforms structured formats such as ACORD, HIPAA, HL7, EDI-X12, EDIFACT, AFP, and SWIFT.

The Custom Data transformation passes data to a Data Transformation service. Data Transformation is the application that transforms the unstructured and semi-structured file formats. The Data Integration Service passes data from the Custom Data transformation to a Data Transformation application. The Data Transformation application transforms the data, and returns the transformed data to the Data Integration Service.

The Data Transformation application can write the output file or return output data to the Data Integration Service. When Data Transformation returns output to the Data Integration Service, it returns XML data. You can configure the Custom Data transformation to return the XML in an output port, or you can configure output groups to return row data.

The following processes occur when you run a Custom Data transformation mapping:

1. The Data Integration service passes source data to Data Transformation Engine.
2. Data Transformation Engine runs a Data Transformation service to transform the data. The Data Transformation service is in the Data Transformation repository folder.
3. Data Transformation Engine writes transformed data directly to output files, or it returns the transformed data to the Data Integration Service. The Custom Data transformation passes output data to the mapping.
4. The Custom Data transformation returns XML data or data rows.

Data Transformation Repository Location

The Developer tool and the Data Integration Service must access a Data Transformation service from a Data Transformation repository. You can configure an environment variable that defines the location of the Data Transformation configuration file.
If Data Transformation is on the same machine as the Developer tool or the Data Integration Service machine, you do not need to configure the Data Transformation repository location. If Data Transformation is installed on a different machine, you must set an environment variable to the path of the Data Transformation configuration file.

Set the IFCONTENTMASTER_HOME environment variable to the path of the CMConfig.xml file. CMConfig.xml is in the Data Transformation installation directory.

The configuration file contains the location of the Data Transformation repository for the Data Transformation installation. The default path to the Data Transformation repository is:

    <Data Transformation Installation Directory>\ServiceDB

## Custom Data Transformation Views

The Custom Data transformation has the following views:

**Overview**
- Description of the transformation and the Custom Data transformation ports.

**Service**
- Service name or dynamic service names.

**Structure**
- Hierarchy of output groups and output ports to enable the Custom Data transformation to write rows to relational targets.

**Advanced**
- Custom Data transformation attributes such as the input type, output type, service name, and streamer chunk size.

### Overview View

Add Custom Data transformation ports in the Overview view. You can also enter a description and rename the transformation.

When you create a Custom Data transformation, the Developer tool creates default ports. The Developer tool creates other ports based on how you configure the transformation. The input type and output type determine how the transformation passes data to and receives data from Data Transformation Engine.

The following table lists the Custom Data transformation default ports:

<table>
<thead>
<tr>
<th>Port</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputBuffer</td>
<td>Input</td>
<td>Receives source data when the input type is buffer. Receives a source file name and source file path when the input type is file.</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>Input</td>
<td>Receives a name for an output file when the output type is file.</td>
</tr>
<tr>
<td>ServiceName</td>
<td>Input</td>
<td>Receives the name of a Data Transformation service when you enable Dynamic Service Name.</td>
</tr>
<tr>
<td>Port</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OutputBuffer</td>
<td>Output</td>
<td>Returns XML data when the output type is buffer. Returns the output file name when the output type is file. Returns no data when you configure hierarchical output groups of ports.</td>
</tr>
<tr>
<td>Status_Code</td>
<td>Output</td>
<td>Returns a status code from Data Transformation Engine when the status tracing level is Description Only or Full Status.</td>
</tr>
<tr>
<td>Status_Message</td>
<td>Output</td>
<td>Returns a status message from Data Transformation Engine when the status tracing level is Description Only or Full Status.</td>
</tr>
</tbody>
</table>

**Note:** You can add groups of output ports for relational targets in the Structure view. When you configure groups of ports, a message appears in the Overview view that hierarchical groups of ports are in another view.

**Service View**

Select a Data Transformation service in the Service view.

When you create a Custom Data transformation, you must choose the name of the Data Transformation service to transform the data. You can choose one service to process all input rows or you can choose a different service to process each input row.

To call a different Data Transformation service for each source row, enable **Dynamic Service**. The Developer tool creates a ServiceName input port. Each source row must pass the name of the Data Transformation service to process the row.

To select a Data Transformation service to process all rows, enable **From the Data Transformation Service**. Click **Browse**. The Developer tool displays a list of the Data Transformation services that are in the Data Transformation repository. Choose the name of a Data Transformation service that you want to call from the Custom Data transformation.

When you enable dynamic service names, you cannot create ports from a Data Transformation service.

**Data Transformation Service Types**

A Data Transformation service transforms data based on the service type.

Data Transformation can transform data with the following service types:

**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. A mapper processes XML input. The mapper generates XML output, which is similar to the parser. The input and the output are fully structured XML.
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 standalone component.

Streamer

Splits large input documents, such as multigigabyte data streams, into segments. The streamer processes documents that have multiple messages or multiple records in them, such as HIPAA or EDI files.

Structure View

You can pass row data to relational tables or other targets from the Custom Data transformation. Configure output ports in the Structure view. Define groups of output ports and define a relational structure for the groups.

The output groups represent the relational tables or the targets that you want to pass the output data to. The Custom Data transformation returns rows to the group ports instead of writing an XML file to the OutputBuffer port. The transformation writes rows based on the output type.

Create a hierarchy of groups in the left pane of the Structure view. All groups are under the root group, PC_XSD_ROOT. You cannot delete the root. Each group can contain ports and other groups. The group structure represents the relationship between target tables. When you define a group within a group, you define a parent-child relationship between the groups. The Developer tool defines a primary key-foreign key relationship between the groups with a generated key.

Select a group to display the ports for the group. You can add or delete ports in the group. When you add a port, the Developer tool creates a default port configuration. Change the port name, datatype, and precision. If the port must contain data select Not Null. Otherwise, the output data is optional.

When you copy a Decimal port from another transformation, the port changes to a Double datatype in the Custom Data transformation. You cannot define a port as a Decimal datatype if the port passes data to the Data Transformation Engine. You can create a pass-through port that is a Decimal datatype.

When you view the Custom Data transformation in the Overview view, each port in a transformation group has a prefix that contains the group name.

When you configure hierarchical groups of output ports, the Data Integration Service writes to the groups of ports instead of writing to the OutputBuffer port. The Data Integration Service writes to the groups of ports regardless of the output type you define for the transformation.

**Note:** When you delete a group, you delete the ports in the group and the child groups.

Exporting the Hierarchy Schema

When you define hierarchical output groups in the Custom Data transformation, you must define the same structure in the Data Transformation service that transforms the data. Export the output hierarchy structure as an XML schema file from the Custom Data transformation. Import the schema to your Data Transformation project. You can map the content of a source document to the XML elements and attributes in the Data Transformation project.

To export the group hierarchy from the Structure view, click Export to XML Schema. Choose a name and a location for the .xsd file. Choose a location that you can access when you import the schema with Data Transformation Studio.

The Developer tool creates an XML schema file with the following namespace:

"www.informatica.com/CDET/XSD/mappingName_Unstructured_Data"

The schema includes the following comment:

<!-- ----- AUTO-GENERATED FILE - DO NOT EDIT ----- -->
<!-- ----- This file has been generated by Informatica Developer ----- -->
If you change the schema, the Data Transformation Engine might return data that is not the same format as the output ports in the Custom Data transformation.

The XML elements in the schema represent the output ports in the hierarchy. Columns that can contain null values have a minOccurs=0 and maxOccurs=1 XML attribute.

**Advanced View**

Configure the Custom Data transformation attributes in the Advanced view.

The following list describes the attributes in the Advanced view:

- **InputType**
  
  Type of input data that the Custom Data transformation passes to Data Transformation Engine. Choose one of the following input types:
  
  - Buffer. The Custom Data transformation receives source data in the InputBuffer port and passes data from the port to Data Transformation Engine.
  
  - File. The Custom Data transformation receives a source file path in the InputBuffer port and passes the source file path to Data Transformation Engine. Data Transformation Engine opens the source file.

- **OutputType**
  
  Type of output data that the Custom Data transformation or Data Transformation Engine returns. Choose one of the following output types:
  
  - Buffer. The Custom Data transformation returns XML data through the OutputBuffer port unless you configure a relational hierarchy of output ports. If you configure a relational hierarchy of ports, the Custom Data transformation returns output to the relational hierarchy of ports instead of the OutputBuffer port.
  
  - File. Data Transformation Engine writes the output to a file. The Data Transformation Engine does not return the data to the Custom Data transformation unless you configure a relational hierarchy of ports in the Custom Data transformation.

- **Data Transformation Service Type**
  
  Name of the Data Transformation service to run. The service must be present in the local Data Transformation repository.

- **Dynamic Service Name**
  
  Runs a different Data Transformation service for each input row. When you enable dynamic service names, the Custom Data transformation receives the service name in the Service Name input port.

  When you disable dynamic service names, the Custom Data transformation runs the same service for each input row. The Service Name attribute in the Service view must contain a service name. Default is disabled.

- **Status Tracing Level**
  
  Sets the level of status messages from the Data Transformation service.

  - Description Only. Return a status code and a short description to indicate if the Data Transformation service was successful or if it failed.
  
  - Full Status. Return a status code and a status message from the Data Transformation service in XML.
  
  - None. Do not return status from the Data Transformation service. Default is none.

- **Streamer Chunk Size**
  
  Buffer size of the data that the Custom Data transformation passes to Data Transformation Engine when the Data Transformation service runs a streamer. Valid values are from 1 to 1 million KB. Default is 256 KB.
Tracing Level

Amount of detail displayed in the log for this transformation.

Default is normal.

The following table describes the tracing levels:

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terse</td>
<td>Logs initialization information and error messages and notification of rejected data.</td>
</tr>
<tr>
<td>Normal</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose Initialization</td>
<td>In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.</td>
</tr>
<tr>
<td>Verbose Data</td>
<td>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.</td>
</tr>
</tbody>
</table>

Status Tracing Messages

You can configure the Custom Data Transformation to return status messages from the Data Transformation service.

Set the status tracing level to Description Only or Full Status. The Developer tool creates the Status_Code port and the Status_Message output ports in the Custom Data transformation.

When you choose Description Only, the Data Transformation service returns a status code and one of the following status messages:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Status Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td>2</td>
<td>Warning</td>
</tr>
<tr>
<td>3</td>
<td>Failure</td>
</tr>
<tr>
<td>4</td>
<td>Error</td>
</tr>
<tr>
<td>5</td>
<td>Fatal Error</td>
</tr>
</tbody>
</table>

When you choose Full Status, Data Transformation service returns a status code and the error message. The message is in XML format.
Ports Configuration

When you create a Custom Data transformation, the Developer tool creates ports based on the input and output configuration. You can create additional ports for the Custom Data transformation based on the Data Transformation service requirements.

Ports by Input and Output Type

The input type determines the type of data that the Data Integration Service passes to Data Transformation Engine. The input type determines whether the input is data or a source file path.

Configure one of the following input types:

- **Buffer.** The Custom Data transformation receives source data in the InputBuffer port. The Data Integration Service passes source rows from the InputBuffer port to Data Transformation Engine.
- **File.** The Custom Data transformation receives the source file path in the InputBuffer port. The Data Integration Service passes the source file path to Data Transformation Engine. Data Transformation Engine opens the source file. Use the file input type to parse binary files such as Microsoft Excel or Microsoft Word files.

If you do not define output groups and ports, the Custom Data transformation returns data based on the output type.

Configure one of the following output types:

- **Buffer.** The Custom Data transformation returns XML through the OutputBuffer port. You must connect an XML Parser transformation to the OutputBuffer port.
- **File.** Data Transformation Engine writes the output file instead of passing data to the Data Integration Service. Data Transformation Engine names the output file based on the file name from the OutputFilename port. Choose the File output type when you want to transform XML to binary data. For example, you can create a PDF file or a Microsoft Excel file.

The Data Integration Service returns the output file name in the OutputBuffer port for each source row. 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 OutputBuffer and returns a row error.

If a Custom Data transformation has the File output type, and you have not defined group output ports, you must link the OutputBuffer port to a downstream transformation. Otherwise, the mapping is not valid.

Additional Ports

A Data Transformation service might require multiple input files, file names, and parameters. It can return multiple output files.

When you create a Custom Data transformation, the Designer creates one InputBuffer port and one OutputBuffer port. If you need to pass additional files or file names between the Custom Data transformation and Data Transformation Engine, add input or output ports. You can add ports manually or from the Data Transformation service.
The following table describes the ports you can create in the Overview view:

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Input (buffer)</td>
<td>Input</td>
<td>Receives input data to pass to Data Transformation Engine.</td>
</tr>
<tr>
<td>Additional Input (file)</td>
<td>Input</td>
<td>Receives the file name and path for Data Transformation Engine to open.</td>
</tr>
<tr>
<td>Additional Output (file)</td>
<td>Input</td>
<td>Receives an output file name. The Data Transformation service creates an additional output file of that name.</td>
</tr>
<tr>
<td>Service Parameter</td>
<td>Input</td>
<td>Receives an input parameter for a Data Transformation service.</td>
</tr>
<tr>
<td>Additional Output (buffer)</td>
<td>Output</td>
<td>Receives XML data from Data Transformation Engine.</td>
</tr>
<tr>
<td>Pass-through</td>
<td>Input/Output</td>
<td>Passes data through the Custom Data transformation without changing it.</td>
</tr>
</tbody>
</table>

Create Ports from a Data Transformation Service

You can add ports that pass parameters, additional input files, and additional output files to a Data Transformation service. You can create ports that correspond to the ports in the Data Transformation service.

A Data Transformation service can require input parameters, additional input files, or user-defined variables. The service might return more than one output file to the Custom Data transformation.

1. After you choose the Data Transformation service, click Select.
   The Developer tool displays the service parameters, additional input, and additional output port requirements from the Data Transformation service. Service parameters include Data Transformation system variables and user-defined variables.
2. Select ports from the list. You can select all ports that appear.
3. Click OK to create the ports in the Custom Data transformation.

Creating a Custom Data Transformation

Create a Custom Data transformation in the Developer tool.

   The New Object wizard appears.
2. Select Custom Data.
3. Click Next.
   The Custom transformation dialog box appears.
4. Configure the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the transformation.</td>
</tr>
<tr>
<td>Location</td>
<td>Location for the Custom Data transformation. Location includes the Model Repository service, project, and folder. Default is the current location.</td>
</tr>
<tr>
<td>Create as</td>
<td>Creates the ports and attributes for the Custom Data transformation. Choose one of the following options:</td>
</tr>
<tr>
<td></td>
<td>- Create as Empty. Do not create default ports. Configure the ports and attributes after you create the transformation.</td>
</tr>
<tr>
<td></td>
<td>- Create as Dynamic Transformation. Configure the transformation to call different Data Transformation services based on the value in the ServiceName input port.</td>
</tr>
<tr>
<td></td>
<td>- Create from Data Transformation Service. Browse for a Data Transformation service from the Data Transformation repository. Click Select to choose input, output, or variable ports that the service requires.</td>
</tr>
<tr>
<td>Service Name</td>
<td>Name of the Data Transformation service you want to use. The Developer tool displays the Data Transformation services in the Data Transformation repository folder. Do not choose a name if you plan to enable dynamic service names. You can add a service name on the Service view after you create the transformation. Service Name</td>
</tr>
<tr>
<td>Ports</td>
<td>Click Select to choose additional input or output ports that the Data Transformation service might require.</td>
</tr>
<tr>
<td>Input Type</td>
<td>Data Transformation Engine receives input data.</td>
</tr>
<tr>
<td></td>
<td>- Buffer. The Custom Data transformation passes rows to the Data Transformation service.</td>
</tr>
<tr>
<td></td>
<td>- File. The Custom Data transformation passes a file name to the Data Transformation service. Default is Buffer.</td>
</tr>
<tr>
<td>Output Type</td>
<td>Data Transformation Engine returns output data.</td>
</tr>
<tr>
<td></td>
<td>- Buffer. The Data Transformation Service returns XML or row data.</td>
</tr>
<tr>
<td></td>
<td>- File. The Data Transformation service writes the output file. Default is Buffer.</td>
</tr>
</tbody>
</table>

5. Click OK.

6. You can change the service name and description and ports in the **Overview** view. If the Data Transformation service has more than one input or output file, or if it requires input parameters, you can add ports in the **Overview** view. You can also add pass-through ports.

7. Configure the Custom Data transformation attributes in the **Advanced** view.

8. If you want to return row data from the Custom Data transformation, create groups of output ports in the **Structure** view.

9. If you create groups of ports, export the schema that describes them from the **Structure** view.

   You can import the schema to the Data Transformation project to define the project output.

10. If you configure the Custom Data transformation with a relational hierarchy of output ports, you can preview the output data in the Developer tool Data Viewer.

    When you preview the output data, the Data Transformation Engine runs the Data Transformation service from the local Data Transformation repository.

    **Note:** If you configure the Custom Data transformation to return data in hierarchical groups of ports, the **OutputBuffer** port does not return XML data. The **OutputBuffer** port returns only the XML header in a data preview.
CHAPTER 9

Decision Transformation

This chapter includes the following topics:

- Decision Transformation Overview, 50
- Decision Transformation Functions, 51
- Decision Transformation Conditional Statements, 52
- Decision Transformation Operators, 53
- Configuring a Decision Strategy, 54

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**

- ADD_TO_DATE
- ASCII
- CEIL
- CHOOSE
- CONCAT
- CONVERT_BASE
- COS
- COSH
- CRC32
- CUME
- DATE_DIFF
- EXP
- FLOOR
- FV
- GET_DATE_PART
- GREATEST
- IN
- INDEXOF
- INITCAP
- ISNULL
- LAST_DAY
- LEAST
- LN
- LOG
- LOWER
- LPAD
- MAKE_DATE_TIME
- MD5
- METAPHONE
- MOD
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.
Use Decision transformation operators to define decision strategies.

The following table describes the decision transformation operators:

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>:=</td>
<td>Assigns a value to a port.</td>
</tr>
<tr>
<td>Boolean</td>
<td>AND</td>
<td>Adds a required logical condition. For the parent Boolean expression to be true, all logical conditions linked by this operator must be true.</td>
</tr>
<tr>
<td>Boolean</td>
<td>OR</td>
<td>Adds a logical condition. For the parent Boolean expression to be true, at least one logical condition linked by this operator must be true.</td>
</tr>
<tr>
<td>Boolean</td>
<td>NOT</td>
<td>Specifies a negative logical condition. For the parent Boolean expression to be true, the negative condition specified by this operator must be true.</td>
</tr>
<tr>
<td>Decision</td>
<td>=</td>
<td>Tests whether compared items are equal. Use with string or numeric datatypes.</td>
</tr>
<tr>
<td>Decision</td>
<td>&lt;&gt;</td>
<td>Tests whether compared items are not equal. Use with string or numeric datatypes.</td>
</tr>
<tr>
<td>Decision</td>
<td>&lt;</td>
<td>Tests whether a value is less than another value. Use with numeric datatypes.</td>
</tr>
<tr>
<td>Decision</td>
<td>&lt;=</td>
<td>Tests whether a value is less than or equal to another value. Use with numeric datatypes.</td>
</tr>
<tr>
<td>Decision</td>
<td>&gt;</td>
<td>Tests whether a value is greater than another value. Use with numeric datatypes.</td>
</tr>
<tr>
<td>Decision</td>
<td>&gt;=</td>
<td>Tests whether a value is greater than or equal to another value. Use with numeric datatypes.</td>
</tr>
<tr>
<td>Numerical</td>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>Numerical</td>
<td>NEG</td>
<td>Negation</td>
</tr>
<tr>
<td>Numerical</td>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>Numerical</td>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>Numerical</td>
<td>/</td>
<td>Division</td>
</tr>
</tbody>
</table>
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.

---

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical</td>
<td>%</td>
<td>Modulo. Returns the remainder after dividing one number by another.</td>
</tr>
<tr>
<td>String</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 10

Expression Transformation

This chapter includes the following topics:
- Expression Transformation Overview, 55
- Developing an Expression Transformation, 55
- Expression Transformation Advanced Properties, 56
- Expression Transformation Ports, 56

Expression Transformation Overview

The Expression transformation is a passive transformation that you can use to perform nonaggregate calculations in a single row. You can also use the Expression transformation to test conditional statements before you pass the results to a target or other transformations.

For example, you might need to adjust employee salaries, concatenate first and last names, or convert strings to numbers.

You can enter multiple expressions in a single Expression transformation by creating an expression for each output port. For example, you might want to calculate different types of withholding taxes from each employee paycheck, such as local and federal income tax, Social Security and Medicare. Because these calculations require the employee salary, the withholding category, and may require the corresponding tax rate, you can create pass-through ports for the salary and withholding category. You can also create a separate output port for each calculation.

Developing an Expression Transformation

When you develop an Expression transformation, you need to consider factors, such as the port types and expression types you want to create.

Consider the following factors when you develop an Expression transformation:
- The port type, such as input, output, pass-through, or variable. You need to include an input or pass-through port and an output port to use an expression in an Expression transformation.
- The expressions you want to use in the transformation.
Expression Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Expression transformation.

Configure the following property on the Advanced tab:

Tracing Level

Amount of detail displayed in the log for this transformation.

Default is normal.

The following table describes the tracing levels:

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terse</td>
<td>Logs initialization information and error messages and notification of rejected data.</td>
</tr>
<tr>
<td>Normal</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose Initialization</td>
<td>In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.</td>
</tr>
<tr>
<td>Verbose Data</td>
<td>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.</td>
</tr>
</tbody>
</table>

Expression Transformation Ports

An Expression transformation has different port types that allow you to perform different transformation tasks, such as pass data to downstream transformations and enter expressions.

An Expression transformation has the following port types:

Input

Receives data from upstream transformations. Input ports provide values used in a calculation.

Output

Provides 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.

Pass-Through

Passes data unchanged.

Variable

Used for local variables.
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.

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 Transformation Advanced Properties

Configure properties that help determine how the Data Integration Service processes data for the Filter transformation.

Configure the following property on the Advanced Properties tab:

Tracing Level

Amount of detail displayed in the log for this transformation.

Default is normal.
The following table describes the tracing levels:

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terse</td>
<td>Logs initialization information and error messages and notification of rejected data.</td>
</tr>
<tr>
<td>Normal</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose Initialization</td>
<td>In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.</td>
</tr>
<tr>
<td>Verbose Data</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 $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 also enter a constant for the filter condition. The numeric equivalent of FALSE is zero (0). Any non-zero value is the equivalent of TRUE. For example, the transformation contains a port named NUMBER_OF_UNITS with a numeric datatype. You configure a filter condition to return FALSE if the value of NUMBER_OF_UNITS equals zero. Otherwise, the condition returns TRUE.

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 treated as FALSE.

**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:

```
IF(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 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.
CHAPTER 12

Java Transformation

This chapter includes the following topics:
- Java Transformation Overview, 60
- Designing a Java Transformation, 63
- Java Transformation Ports, 63
- Java Transformation Advanced Properties, 64
- Developing Java Code, 66
- Java Transformation Java Properties, 69
- Creating a Java Transformation, 72
- Compiling a Java Transformation, 73
- Troubleshooting a Java Transformation, 73

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.
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.

Editor Views for a Reusable Java Transformation

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

To create a reusable Java transformation, see "Creating a Reusable Java Transformation" on page 72.

For reusable Java transformations, the following views are available:

Overview

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

Java

Define, compile, and fix compilation errors in Java code. In the Java view, the following tabs are available:

- Code entry tabs, on which you can define Java code snippets.
- The Full Code tab, on which you can view the complete Java class code for the Java transformation.

Advanced

Set advanced properties for the Java transformation.

Editor Views for a Non-Reusable Java Transformation

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

To create a non-reusable Java transformation, see "Creating a Non-Reusable Java Transformation" on page 72.

For non-reusable Java transformations, the following views are available:

General

Enter the name and description of the transformation.

Ports

Create and configure input and output ports.

Java

Define, compile, and fix compilation errors in Java code. In the Java view, the following tabs are available:

- Code entry tabs, on which you can define Java code snippets.
- The Full Code tab, on which you can view the complete Java class code for the Java transformation.

Advanced

Set advanced properties for the Java transformation.

Active and Passive Java Transformations

When you create a Java transformation, you define its type as active or passive.

After you set the transformation type, you cannot change it.
A Java transformation runs the Java code that you define on the On Input tab one time for each row of input data. A Java transformation handles output rows based on the transformation type as follows:

- A passive Java transformation generates one output row for each input row in the transformation after processing each input row.
- An active Java transformation generates multiple output rows for each input row in the transformation.

Use the generateRow method to generate each output row. For example, if the transformation contains two input ports that represent a start date and an end date, you can use the generateRow method to generate an output row for each date between the start date and the end date.

### Datatype Conversion

A Java transformation converts Developer tool datatypes to Java datatypes, based on the Java transformation port type.

When a Java transformation reads input rows, it converts input port datatypes to Java datatypes. When a Java transformation writes output rows, it converts Java datatypes to output port datatypes.

For example, the following processing occurs for an input port with the integer datatype in a Java transformation:

1. The Java transformation converts the integer datatype of the input port to the Java primitive int datatype.
2. In the transformation, the transformation treats the value of the input port as the Java primitive int datatype.
3. When the transformation generates the output row, it converts the Java primitive int datatype to the integer datatype.

The following table shows how the Java transformation maps Developer tool datatypes to Java primitive and complex datatypes:

<table>
<thead>
<tr>
<th>Developer Tool Datatype</th>
<th>Java Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigint</td>
<td>long</td>
</tr>
<tr>
<td>binary</td>
<td>byte[]</td>
</tr>
<tr>
<td>date/time</td>
<td>With nanoseconds processing enabled, BigDecimal with nanosecond precision</td>
</tr>
<tr>
<td></td>
<td>With nanoseconds processing disabled, long with millisecond precision (the number of milliseconds since January 1, 1970 00:00:00.000 GMT)</td>
</tr>
<tr>
<td>decimal</td>
<td>With high precision processing disabled, double with precision 15</td>
</tr>
<tr>
<td></td>
<td>With high precision processing enabled, BigDecimal</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>integer</td>
<td>int</td>
</tr>
<tr>
<td>string</td>
<td>String</td>
</tr>
<tr>
<td>text</td>
<td>String</td>
</tr>
</tbody>
</table>

In Java, the String, byte[], and BigDecimal datatypes are complex datatypes, and the double, int, and long datatypes are primitive datatypes.
Note: The Java transformation sets null values in primitive datatypes 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 81.

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 view 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.

The following table shows how the Java transformation initializes ports based on the port datatype:

<table>
<thead>
<tr>
<th>Port Datatype</th>
<th>Port Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td></td>
</tr>
<tr>
<td>Complex</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port Datatype</th>
<th>Port Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>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.</td>
</tr>
<tr>
<td>Complex</td>
<td>If you define a default value for the port, the transformation creates a new String or byte[] object, and initializes the object to the default value. Otherwise, the transformation initializes the port variable to null. <strong>Note:</strong> If you access an input port variable with a null value in the Java code, a NullPointerException occurs.</td>
</tr>
</tbody>
</table>

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.

In the **Advanced** view, you can define the following advanced properties for the Java transformation:

**Tracing Level**

Amount of detail displayed in the log for this transformation.

Default is normal.

The following table describes the tracing levels:

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terse</td>
<td>Logs initialization information and error messages and notification of rejected data.</td>
</tr>
<tr>
<td>Normal</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose Initialization</td>
<td>In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.</td>
</tr>
<tr>
<td>Verbose Data</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Enable high precision**

Processes a decimal datatype port with a precision greater than 15 and less than 28 as a Java BigDecimal datatype port.
Disable high precision processing to process a decimal datatype port with a precision greater than 15 and
less than 28 as a Java Double datatype port.

The following table shows how a Java transformation treats a value in a decimal datatype input port based on
whether you have enabled or disabled the high precision option:

<table>
<thead>
<tr>
<th>Example</th>
<th>High Precision Processing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A decimal type input port receives a value of 40012030304957666903.</td>
<td>Enabled</td>
<td>The Java transformation leaves the value as is.</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>The Java transformation converts the value to the following value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.00120303049577 x 10^-19</td>
</tr>
</tbody>
</table>

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

The transformation can generate more than one output row for each input row.

You cannot change this property after you create the Java transformation. If you need to change this property,
create a new Java transformation.

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 client runs, complete one of the following
tasks:

♦ Configure the CLASSPATH environment variable. Set the CLASSPATH environment variable on the Developer
tool client client machine. This applies to all java processes run on the machine.
For a non-reusable Java transformation, configure 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. You cannot configure this property for a reusable Java transformation.

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 some or all of the following components, which enable you 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:

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigator</td>
<td>Shows input ports, output ports, and callable Java transformation API methods. Click an item in the navigator to display a description of the item. 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. The navigator is available on the following code entry tabs:</td>
</tr>
<tr>
<td></td>
<td>- Helpers</td>
</tr>
<tr>
<td></td>
<td>- On Input</td>
</tr>
<tr>
<td></td>
<td>- At End</td>
</tr>
<tr>
<td>Java code window</td>
<td>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. Note: On the Full Code tab, you can view but not edit the full class code for the Java transformation. The Java code window is available on the following tabs:</td>
</tr>
<tr>
<td></td>
<td>- Imports</td>
</tr>
<tr>
<td></td>
<td>- Helpers</td>
</tr>
<tr>
<td></td>
<td>- On Input</td>
</tr>
<tr>
<td></td>
<td>- At End</td>
</tr>
<tr>
<td></td>
<td>- Functions</td>
</tr>
<tr>
<td></td>
<td>- Full Code</td>
</tr>
<tr>
<td>New Function command</td>
<td>Opens the Define Function dialog box that you use to define functions that invoke Java expressions. The Function command is available on the Functions tab.</td>
</tr>
<tr>
<td>Editing toolbar</td>
<td>Enables you to click tool icons, such as cut, copy, and paste, to edit Java code. The editing toolbar is available on the following tabs:</td>
</tr>
<tr>
<td></td>
<td>- Imports</td>
</tr>
<tr>
<td></td>
<td>- Helpers</td>
</tr>
<tr>
<td></td>
<td>- On Input</td>
</tr>
<tr>
<td></td>
<td>- At End</td>
</tr>
<tr>
<td></td>
<td>- Functions</td>
</tr>
</tbody>
</table>

Compilation properties

Provides controls that let you compile and debug Java code. The following table describes the controls that are available in Compilation properties:

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile command</td>
<td>Compiles the Java code for the transformation.</td>
</tr>
<tr>
<td>Results window</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imports</td>
<td>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.</td>
</tr>
<tr>
<td>Helpers</td>
<td>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.</td>
</tr>
<tr>
<td>On Input</td>
<td>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.</td>
</tr>
<tr>
<td>At End</td>
<td>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.</td>
</tr>
<tr>
<td>Functions</td>
<td>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.</td>
</tr>
<tr>
<td>Full Code</td>
<td>Read-only. On this tab, you can view and compile the full class code for the Java transformation.</td>
</tr>
</tbody>
</table>

2. 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.

3. 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.

4. 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:

```java
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:

  ```java
  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:

  ```java
  // 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:

  ```java
  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**

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:

```java
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 BONUSES, 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:

```java
TOTAL_COMP = myTXAdd (BASE_SALARY, BONUSES);
generateRow();
```
When the Java transformation receives an input row, it adds the values of the BASE_SALARY and BONUSES input ports, assigns the value to the TOTAL_COMP output port, and generates an output row.

- **Java transformation API methods.** You can call API methods provided by the Java transformation.

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:

```java
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.
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.
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 mapplet, drag a Java transformation from the Transformation pallette 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. In the General view, edit the transformation name and the description.
6. In the Ports view, 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:

```java
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:

```java
int interest;
interest = calInterest(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.
Java Transformation API Reference

This chapter includes the following topics:
- Java Transformation API Methods Overview, 75
- defineJExpression, 76
- failSession, 77
- generateRow, 77
- getInRowType, 78
- incrementErrorCount, 78
- invokeJExpression, 79
- isNull, 80
- logError, 80
- logInfo, 81
- setNull, 81

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.

**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 JExprParaMetadata objects that contains the input parameters and a string value that defines the expression syntax.

Use the following syntax:

```java
defineJExpression(
    String expression,
    Object[] paramMetadataArray
);
```

The following table describes the parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Input</td>
<td>String</td>
<td>String that represents the expression.</td>
</tr>
<tr>
<td>paramMetadataArray</td>
<td>Input</td>
<td>Object[]</td>
<td>Array of JExprParaMetadata objects that contain the input parameters for the expression.</td>
</tr>
</tbody>
</table>

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 JExprParaMetadata 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:

```java
JExprParaMetadata params[] = new JExprParaMetadata[2];
params[0] = new JExprParaMetadata(EDataType.STRING, 20, 0);
params[1] = new JExprParaMetadata(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:

```java
failSession(String errorMessage);
```

The following table describes the parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorMessage</td>
<td>Input</td>
<td>String</td>
<td>Error message string.</td>
</tr>
</tbody>
</table>

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:

```java
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:

```java
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:

```java
// 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:

```java
rowType getInRowType();
```

The following table describes the parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowType</td>
<td>Output</td>
<td>String</td>
<td>Returns the update strategy type, which is one of the following values: DELETE, INSERT, REJECT, UPDATE</td>
</tr>
</tbody>
</table>

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.

incrementErrorCount

Increments the error count. If the error count reaches the error threshold, the mapping fails.

Use the following syntax:

```java
incrementErrorCount(int nErrors);
```

The following table describes the parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nErrors</td>
<td>Input</td>
<td>Integer</td>
<td>Number by which to increment the error count.</td>
</tr>
</tbody>
</table>

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:

```java
// Check if input employee id and name is null.
if (isNull("EMP_ID_INF") || isNull("EMP_NAME_INF"))
{
    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:

```java
{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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Input</td>
<td>String</td>
<td>String that represents the expression.</td>
</tr>
<tr>
<td>paramMetadataArray</td>
<td>Input</td>
<td>Object[]</td>
<td>Array of objects that contain the input parameters for the expression.</td>
</tr>
</tbody>
</table>

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”:

```java
(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:

```java
Boolean isNull(String strColName);
```

The following table describes the parameter:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strColName</td>
<td>Input</td>
<td>String</td>
<td>Name of an input column.</td>
</tr>
</tbody>
</table>

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:

```java
// 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:

```java
// 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:

```java
logError(String msg);
```
The following table describes the parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>Input</td>
<td>String</td>
<td>Error message string.</td>
</tr>
</tbody>
</table>

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:

```java
// 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:

```
[JTJ_1013] [ERROR] Cannot process a null salary field.
```

### logInfo

Writes an informational message to the log.

Use the following syntax:

```java
logInfo(String msg);
```

The following table describes the parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>Input</td>
<td>String</td>
<td>Information message string.</td>
</tr>
</tbody>
</table>

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:

```java
if (numRowsProcessed >= messageThreshold) {
    logInfo("Processed " + messageThreshold + " rows.");
}
```

### setNull

Sets the value of an output column to null in an active or passive Java transformation.

Use the following syntax:

```java
setNull(String strColName);
```
The following table describes the parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strColName</td>
<td>Input</td>
<td>String</td>
<td>Name of an output column.</td>
</tr>
</tbody>
</table>

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:

```java
// 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:

```java
// check value of Q3RESULTS input column
String strColName = "Q3RESULTS";
if(isNull(strColName)) {
    // set the value of output column to null
    setNull(strColName);
}
```
This chapter includes the following topics:

- Java Expressions Overview, 83
- Using the Define Function Dialog Box to Define an Expression, 84
- Working with the Simple Interface, 86
- Working with the Advanced Interface, 88
- JExpression Class API Reference, 92

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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define Function</td>
<td>Enables you to create a function that invokes an expression and generate the code for an expression.</td>
</tr>
<tr>
<td>Simple interface</td>
<td>Enables you to call a single API method to invoke an expression and get the result of the expression.</td>
</tr>
<tr>
<td>Advanced interface</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Expression Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation language functions</td>
<td>SQL-like functions designed to handle common expressions.</td>
</tr>
<tr>
<td>User-defined functions</td>
<td>Functions that you create in the Developer tool based on transformation language functions.</td>
</tr>
<tr>
<td>Custom functions</td>
<td>Functions that you create with the Custom Function API.</td>
</tr>
</tbody>
</table>

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.
   - 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:

```java
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:

```java
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:

```java
{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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Input</td>
<td>String</td>
<td>String that represents the expression.</td>
</tr>
<tr>
<td>paramMetadataArray</td>
<td>Input</td>
<td>Object[]</td>
<td>Array of objects that contain the input parameters for the expression.</td>
</tr>
</tbody>
</table>

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”:

```java
(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.

**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:

```java
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:

<table>
<thead>
<tr>
<th>Class or API Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDataType class</td>
<td>Enumerates the datatypes for an expression.</td>
</tr>
<tr>
<td>JExprParamMetadata class</td>
<td>Contains the metadata for each parameter in an expression. Parameter metadata includes datatype, precision, and scale.</td>
</tr>
<tr>
<td>defineJExpression API method</td>
<td>Defines the expression. Includes expression string and parameters.</td>
</tr>
<tr>
<td>invokeJExpression API method</td>
<td>Invokes an expression.</td>
</tr>
<tr>
<td>JExpression class</td>
<td>Contains the methods to create, invoke, get the metadata and get the expression result, and check the return datatype.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Enumerated Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>1</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>2</td>
</tr>
<tr>
<td>STRING</td>
<td>3</td>
</tr>
<tr>
<td>BYTE_ARRAY</td>
<td>4</td>
</tr>
<tr>
<td>DATE_AS_LONG</td>
<td>5</td>
</tr>
</tbody>
</table>

The following example Java code shows how to use the EDataType class to assign a datatype of String to a JExprParamMetadata object:

```java
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:

```java
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:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument Type</th>
<th>Argument Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datatype</td>
<td>Input</td>
<td>EDataType</td>
<td>Datatype of the parameter.</td>
</tr>
<tr>
<td>precision</td>
<td>Input</td>
<td>Integer</td>
<td>Precision of the parameter.</td>
</tr>
<tr>
<td>scale</td>
<td>Input</td>
<td>Integer</td>
<td>Scale of the parameter.</td>
</tr>
</tbody>
</table>
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:

```java
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(DataType.STRING, 20, 0);
params[1] = new JExprParamMetadata(DataType.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:

```java
defineJExpression(      
    String expression,  
    Object[] paramMetadataArray  
);
```

The following table describes the parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Input</td>
<td>String</td>
<td>String that represents the expression.</td>
</tr>
<tr>
<td>paramMetadataArray</td>
<td>Input</td>
<td>Object[]</td>
<td>Array of JExprParamMetadata objects that contain the input parameters for the expression.</td>
</tr>
</tbody>
</table>

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:

```java
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(DataType.STRING, 20, 0);
params[1] = new JExprParamMetadata(DataType.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:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoke</td>
<td>Invokes an expression.</td>
</tr>
<tr>
<td>getResultDataType</td>
<td>Returns the datatype of the expression result.</td>
</tr>
</tbody>
</table>
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:

```java
JExprParamMetadata addressLookup() throws SdkException {
    JExprParamMetadata params[] = new JExprParamMetadata[2];
    params[0] = new JExprParamMetadata {
        EDataType.STRING, // data type
        50, // precision
        0 // scale
    };
    params[1] = new JExprParamMetadata {
        EDataType.STRING, // data type
        50, // precision
        0 // scale
    };
    return defineExpression("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:

```java
if(!isJExprObjCreated) {
    lookup = addressLookup();
    isJExprObjCreated = true;
}
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`
- `getResultDataType`
- `getResultMetadata`
- `invoke`
- `isResultNull`
- `getLong`
- `getStringBuffer`

**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:

```java
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:

```java
byte[] newBytes = JExprEncryptData.getBytes();
```

**getDouble**

Returns the value of an expression result as a Double datatype.

Use the following syntax:

```java
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:

```java
double salary = JExprSalary.getDouble();
```

**getInt**

Returns the value of an expression result as an Integer datatype.

Use the following syntax:

```java
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:

```java
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:

```java
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:

```java
long currDate = JExprCurrentDate.getLong();
```

getResultDataType

Returns the datatype of an expression result. Returns a value of EDataType.

Use the following syntax:

```java
objectName.getResultDataType();
```

Use the following example Java code to invoke an expression and assign the datatype of the result to the variable `dataType`:

```java
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:

```java
objectName.getResultMetadata();
```

Use the following example Java code to assign the scale, precision, and datatype of the return value of `myObject` to variables:

```java
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:

```java
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:

```java
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:

```java
ObjectName.invoke(
    new Object[] { param1, ... paramN },
    rowType
);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Datatype</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectName</td>
<td>JExpression</td>
<td>Input</td>
<td>JExpression object name.</td>
</tr>
<tr>
<td>parameters</td>
<td>n/a</td>
<td>Input</td>
<td>Object array that contains the input values for the expression.</td>
</tr>
</tbody>
</table>

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:

```java
JExpression myObject = address_lookup();
myObject.invoke(new Object[] { 'NAME,COMPANY ' }, ERowType INSERT);
```

isResultNull

Checks the value of an expression result.

Use the following syntax:

```java
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:

```java
JExpression myObject = address_lookup();
myObject.invoke(new Object[] { 'NAME,COMPANY ' }, ERowType INSERT);
if(!myObject.isResultNull()) {
    String address = myObject.getStringBuffer();
}
```
Joiner Transformation

This chapter includes the following topics:

- Joiner Transformation Overview, 95
- Advanced Properties for Joiner Transformations, 95
- Joiner Transformation Ports, 97
- Join Types, 97
- Sorted Input for a Joiner Transformation, 99
- Joining Data from the Same Source, 101
- Blocking the Source Pipelines, 102
- Joiner Transformation Performance Tips, 103
- Rules and Guidelines for a Joiner Transformation, 104

Joiner Transformation Overview

Use the Joiner transformation to join source data from two related heterogeneous sources residing in different locations or file systems. You can also join data from the same source. The Joiner transformation is an 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.

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.

Advanced Properties for Joiner Transformations

Configure properties that help determine how the Data Integration Service processes data for the Joiner Transformation.

Configure the following properties on the Advanced Properties tab:
Joiner Data Cache Size

Data cache size for the transformation. Default is Auto.

Joiner Index Cache Size

Index cache size for the transformation. Default is Auto.

Cache Directory

Local directory where the Data Integration Service creates the index cache files and data cache files. Default is a period.

If you override the directory, make sure the directory exists and contains enough disk space for the cache files. The directory can be a mapped or mounted drive.

Case Sensitive String Comparison

If selected, the Data Integration Service uses case-sensitive string comparisons when performing joins on string columns.

Null Ordering in Master

Not applicable for this transformation type.

Sorted Input

Indicates that input data is presorted by groups. Choose Sorted Input to join sorted data. Using sorted input can increase performance.

Null Ordering in Detail

Not applicable for this transformation type.

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.

Scope

Specifies how the Data Integration Service applies the transformation logic to incoming data.

Tracing Level

Amount of detail displayed in the log for this transformation.

Default is normal.

The following table describes the tracing levels:

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terse</td>
<td>Logs initialization information and error messages and notification of rejected data.</td>
</tr>
<tr>
<td>Normal</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose Initialization</td>
<td>In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.</td>
</tr>
<tr>
<td>Verbose Data</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 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.

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.

### 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 with the following data:

<table>
<thead>
<tr>
<th>PARTS_SIZE (master source)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PART_ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
When you join the sample tables with a master outer join and the same condition, the result set includes the following data:

<table>
<thead>
<tr>
<th>PART_ID</th>
<th>DESCRIPTION</th>
<th>SIZE</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seat Cover</td>
<td>Large</td>
<td>Blue</td>
</tr>
<tr>
<td>3</td>
<td>Floor Mat</td>
<td>Medium</td>
<td>Black</td>
</tr>
</tbody>
</table>

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 these tables with a normal join, the result set includes the following data:

<table>
<thead>
<tr>
<th>PART_ID</th>
<th>DESCRIPTION</th>
<th>SIZE</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seat Cover</td>
<td>Large</td>
<td>Blue</td>
</tr>
<tr>
<td>3</td>
<td>Floor Mat</td>
<td>Medium</td>
<td>Black</td>
</tr>
</tbody>
</table>

The following example shows the equivalent SQL statement:

```
SELECT * FROM PARTS_SIZE, PARTS_COLOR WHERE PARTS_SIZE.PART_ID = PARTS_COLOR.PART_ID
```

**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:

<table>
<thead>
<tr>
<th>PART_ID</th>
<th>DESCRIPTION</th>
<th>SIZE</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seat Cover</td>
<td>Large</td>
<td>Blue</td>
</tr>
<tr>
<td>3</td>
<td>Floor Mat</td>
<td>Medium</td>
<td>Black</td>
</tr>
</tbody>
</table>

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_ID)
```

**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:

<table>
<thead>
<tr>
<th>PART_ID</th>
<th>DESCRIPTION</th>
<th>SIZE</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seat Cover</td>
<td>Large</td>
<td>Blue</td>
</tr>
<tr>
<td>2</td>
<td>Ash Tray</td>
<td>Small</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>Floor Mat</td>
<td>Medium</td>
<td>Black</td>
</tr>
</tbody>
</table>

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_ID = 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:

<table>
<thead>
<tr>
<th>PARTED</th>
<th>DESCRIPTION</th>
<th>SIZE</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seat Cover</td>
<td>Large</td>
<td>Blue</td>
</tr>
<tr>
<td>2</td>
<td>Ash Tray</td>
<td>Small</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>Floor Mat</td>
<td>Medium</td>
<td>Black</td>
</tr>
<tr>
<td>4</td>
<td>Fuzzy Dice</td>
<td>NULL</td>
<td>Yellow</td>
</tr>
</tbody>
</table>

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_ID = PARTS_COLOR.PART_ID)
```

**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.

**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
  - Mapplet 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.

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.

Rules and Guidelines for Join Conditions

Certain rules and guidelines apply when you create join conditions.

Use the following guidelines when you create join conditions:

• 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.

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.

When you use the Joiner transformation to join the master and detail pipelines, you can configure any one of the following join conditions:

\[
\text{ITEM_NO} = \text{ITEM_NO}
\]

or

\[
\text{ITEM_NO} = \text{ITEM_NO1} \\
\text{ITEM_NAME} = \text{ITEM_NAME1}
\]

or

\[
\text{ITEM_NO} = \text{ITEM_NO1} \\
\text{ITEM_NAME} = \text{ITEM_NAME1} \\
\text{PRICE} = \text{PRICE1}
\]

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.
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.

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 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. You see the greatest performance increase when you work with large data sets. For an unsorted Joiner transformation, designate the source with fewer rows as the master source.
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.
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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundex Depth</td>
<td>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.</td>
</tr>
</tbody>
</table>

Related Topics:
- “String Strategy Properties” on page 107
- “Key Creation Properties” on page 108
- “Configuring a Grouping Strategy” on page 108

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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start from left</td>
<td>Configures the transformation to read the input field from left to right.</td>
</tr>
<tr>
<td>Start from right</td>
<td>Configures the transformation to read the input field from right to left.</td>
</tr>
<tr>
<td>Start position</td>
<td>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.</td>
</tr>
<tr>
<td>Length</td>
<td>Specifies the length of the string to use as a group key. Enter 0 to use the whole input field.</td>
</tr>
</tbody>
</table>

Related Topics:
- “Soundex Strategy Properties” on page 106
- “Key Creation Properties” on page 108
- “Configuring a Grouping Strategy” on page 108

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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SequenceID</td>
<td>Creates an ID that identifies each record in the source data set.</td>
</tr>
<tr>
<td>GroupKey</td>
<td>Creates the group keys that the Match transformation uses to process records.</td>
</tr>
</tbody>
</table>

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 106
- “String Strategy Properties” on page 107
- “Key Creation Properties” on page 108

Key Creation Properties

Configure the key creation properties appropriate for the data you analyze.

The following table describes the key creation properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort results</td>
<td>Sorts the Key Generator transformation output using the GroupKey field. For field matching operations, you must</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>select this option or verify that you provide the Match transformation with sorted data. Do not select this option for identity matching operations.</td>
<td></td>
</tr>
<tr>
<td>Generate sequence key automatically</td>
<td>Generates a sequence key field using the order of input data.</td>
</tr>
<tr>
<td>Use field as sequence key</td>
<td>Generates a sequence field for a column that you specify.</td>
</tr>
<tr>
<td>Sequence key field</td>
<td>Specifies the name of the sequence key field.</td>
</tr>
</tbody>
</table>

**Related Topics:**
- “Soundex Strategy Properties” on page 106
- “String Strategy Properties” on page 107
- “Configuring a Grouping Strategy” on page 108
CHAPTER 17

Labeler Transformation

This chapter includes the following topics:
- Labeler Transformation Overview, 110
- Labeler Transformation Strategies, 110
- Labeler Transformation Ports, 111
- Character Labeling Operations, 111
- Token Labeling Operations, 112
- Configuring a Labeling Strategy, 114

Labeler Transformation Overview

The Labeler transformation is a passive transformation that examines input fields and creates labels that describe the type of characters or strings in each field.

You can use the Labeler transformation to perform the following types of labeling:

Character Labeling

Creates a label for each character identified in the input data. For example, the Labeler transformation can label the ZIP Code 10028 as "nnnnn," where "n" stands for a numeric character.

Token Labeling

Creates a label for each token identified in the input data. A token is a logical unit of meaning. For example, you can configure the Labeler transformation to label the string "John J. Smith" with the tokens "Word Init Word."

Labeler Transformation Strategies

Use labeling strategies to assign labels to input data.

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.

**Labeler Transformation Ports**

The Labeler transformation includes input ports and labeled output ports. Labeler transformations in token labeling mode also include a tokenized data output.

Labeler transformations use the following port types:

**Input Ports**
- Receive string input from upstream objects.

**Labeled Output Ports**
- Outputs the labels defined by transformation strategies.

**Tokenized Data Output Ports**
- Passes through the input strings that correspond to each label in the output. The Parser transformation requires the token port as input when operating in pattern-based parsing mode.

**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.

**Character Labeling Properties**

Configure properties for character labeling operations on the **Strategies** view in the Labeler transformation.

You can configure the following types of character labeling operations:

**Character Set Labeling Operations**

- Select **Label Characters Using Character Sets** to configure character set properties. Review and edit the following properties:
  - **Select Character Sets.** Specifies the character sets that the transformation uses to label strings
  - **Label.** Specifies the replacement text for input strings that match the character set. When you enable a character set, you can click the selection arrow in the Label column to enter custom replacement text.
  - **Show selected character sets.** Excludes character sets the user does not select.
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.

Reference Table Labeling Operations

Select Label Characters Using Reference Tables to configure the following reference table labeling properties.

- 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.

When configuring an operation, you can select the Ignore Text view to specify exceptions for character set operations and reference table labeling operations. Review and edit the following properties:

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 Text to Ignore string.

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 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 using predefined token sets provided with Informatica Developer.
Token Sets

Token sets define patterns within strings.
Informatica Developer includes token sets that cannot be created or edited. You can use these token sets to identify a range of patterns, including:
- Words
- Numbers
- Phone numbers
- Email addresses
- Postal codes
- National identification numbers, for example, Social Security Numbers
- Credit card numbers
You can also create user-defined token sets by entering custom regular expressions.

Token Labeling Properties

To configure properties for token labeling strategies and operations, select the Strategies view in the Labeler transformation.

You can configure the following properties for a token labeling strategy:

Reverse Enabled
Processes input data from right to left.

Delimiters
Specifies the characters that the transformation uses when evaluating substrings within input data.

You can configure the following types of token labeling operations:

Reference Table Labeling Operations
Select Label with Reference Table to configure the following reference table labeling options:
- Reference Table. Specifies the reference tables that the operation uses to label tokens.
- Label. Specifies the replacement text for input strings that match reference table entries.
- 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.
- Set Priority. Determines whether reference table labeling takes precedence in labeling operations.
- 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.

Token Set Labeling Operations
Select Label Tokens with Token Set to configure token set properties. Review and edit the following properties:
- Select Token Sets. Determines the token sets that the transformation uses to label strings.
- Label. Specifies the replacement text for input strings that match the token set. When you enable a token set, you can click the selection arrow in the Label column to enter custom replacement text.
- Show selected token sets. Excludes token sets the user does not select.
♦ **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.

When configuring an operation, you can select the **Custom Label** view to create labels for specific search terms. Review and edit the following properties:

**Search Term**
- Determines the string to search for.

**Case Sensitive**
- Determines whether the input data must match the case of the search term.

**Custom Label**
- Determines the custom label to apply.

**Configuring a Labeling Strategy**

To configure a labeling strategy, edit the settings in the **Strategies** view of a Labeler transformation.

1. Select the **Strategies** view.
2. Click **New** to create a strategy.
   - The **New Strategy** wizard opens.
3. Click the **Inputs** and **Outputs** fields to select ports for the strategy.
4. Select a labeling mode. If you select token labeling mode, configure the token labeling strategy properties. Click **Next**.
5. Select an operation and click **Next**.
6. Configure the operation properties and click **Next**.
7. Configure additional options.
   - If you selected character labeling mode, optionally configure **Ignore Text** properties.
   - If you selected token labeling mode, optionally configure **Custom Label** properties.
8. Optionally, click **Next** to add more operations to the strategy.
9. Click **Finish** to save the strategy.
10. Optionally, 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**.
11. Optionally, add more strategies to the transformation.
CHAPTER 18

Lookup Transformation

Lookup Transformation Overview

Use a Lookup transformation in a mapping to look up data in a flat file or relational table.

You can perform the following tasks with a Lookup transformation:

- Get a related value. Retrieve a value from the lookup table based on a value in the source. For example, the source has an employee ID. Retrieve the employee name from the lookup table.
- Perform a calculation. Retrieve a value from a lookup table and use it in a calculation. For example, retrieve a sales tax percentage, calculate a tax, and return the tax to a target.

When you create a Lookup transformation, you import a lookup definition from a flat file or relational data object in the Developer tool. The Developer tool adds the columns from the physical data object as lookup ports in the transformation. After you create the transformation, you add one or more outputs ports that return the lookup results. You also enter one or more lookup conditions and configure other lookup properties.

When you run a mapping or preview data, the Data Integration Service queries the lookup source based on the lookup ports in the transformation, lookup properties, and a lookup condition. The Lookup transformation returns the result of the lookup to the target or another transformation.

You must connect a Lookup transformation to an upstream transformation or source and a downstream transformation or target. You can use multiple Lookup transformations in a mapping.

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 or relational data object.
- The output ports for the transformation.
- The lookup conditions in the transformation.
- Whether you want the Data Integration Service to cache lookup data.
Runtime Properties for Lookup Transformations

Configure runtime properties to determine if the Data Integration Service caches lookup data.

The following table describes runtime properties for Lookup transformations:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup Caching Enabled</td>
<td>Indicates whether the Data Integration Service caches lookup values. When you enable lookup caching, the Data Integration Service queries the lookup source once, caches the values, and looks up values in the cache. Caching the lookup values can increase performance. When you disable caching, each time a row passes into the transformation, the Data Integration Service issues a select statement to the lookup source for lookup values. The Data Integration Service always caches flat file lookups.</td>
</tr>
<tr>
<td>Lookup Data Cache Size</td>
<td>Maximum size the Data Integration Service allocates to the data cache in memory. Default is Auto.</td>
</tr>
<tr>
<td>Lookup Index Cache Size</td>
<td>Maximum size the Data Integration Service allocates to the index in memory. Default is Auto.</td>
</tr>
<tr>
<td>Pre-build Lookup Cache</td>
<td>Allows the Data Integration Service to build the lookup cache before the Lookup transformation receives the data. The Data Integration Service can build multiple lookup cache files at the same time to increase performance. Configure one of the following options: - Auto. The Data Integration Service determines the value. - Always Allowed. Allows the Data Integration Service to build the lookup cache before the Lookup transformation receives the data. The Data Integration Service can build multiple lookup cache files at the same time to increase performance. - Always disallowed. The Data Integration Service cannot build the lookup cache before the Lookup transformation receives the first row.</td>
</tr>
<tr>
<td>Lookup Cache Directory Name</td>
<td>Directory used to build the lookup cache files when you configure the Lookup transformation to cache the lookup source.</td>
</tr>
<tr>
<td>Connection</td>
<td>Applies to Relational Lookup transformations. Source of the lookup.</td>
</tr>
</tbody>
</table>

Advanced Properties for Flat File Lookup Transformations

Configure advanced properties for relational Lookup transformations, such as the persistent lookup cache and datetime format.
The following table describes advanced properties for relational lookup transformations:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup Cache Persistent</td>
<td>Indicates whether the Data 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 Data Integration Service creates the files.</td>
</tr>
<tr>
<td>Case Sensitive String Comparison</td>
<td>Data Integration Service uses case-sensitive string comparisons when performing lookups on string columns.</td>
</tr>
<tr>
<td>Null Ordering</td>
<td>Determines how the Data Integration Service orders null values. You can choose to sort null values high or low. By default, the Data Integration Service sorts null values high. This overrides the Data 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.</td>
</tr>
<tr>
<td>Trace Level</td>
<td>Sets the amount of detail displayed in the log.</td>
</tr>
<tr>
<td>Custom SQL Query</td>
<td>Overrides the default SQL query.</td>
</tr>
<tr>
<td>Source Filter</td>
<td>Reduces the number of rows the Data Integration Service queries.</td>
</tr>
<tr>
<td>Datetime Format</td>
<td>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 MM/DD/YYYY HH24:MI:SS. The Datetime format does not change the size of the port.</td>
</tr>
<tr>
<td>Thousand Separator</td>
<td>Value is None.</td>
</tr>
<tr>
<td>Decimal Separator</td>
<td>Value is a period.</td>
</tr>
</tbody>
</table>

Advanced Properties for Relational Lookup Transformations

Configure advanced properties for relational Lookup transformations, such as the persistent lookup cache and pre-SQL.

The following table describes advanced properties for relational lookup transformations:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup Cache Persistent</td>
<td>Indicates whether the Data 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 Data Integration Service creates the files.</td>
</tr>
</tbody>
</table>
### Property Description

- **lookup cache and persistent lookup cache files do not exist, the Data Integration Service creates the files.**

- **Case Sensitive String Comparison**
  Data Integration Service uses case-sensitive string comparisons when performing lookups on string columns. The case-sensitive comparison is based on the database support.

- **Null Ordering**
  Determines how the Data Integration Service orders null values. You can choose to sort null values high or low. By default, the Data Integration Service sorts null values high. This overrides the Data 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.

- **Trace Level**
  Sets the amount of detail included in the log.

- **Custom SQL Query**
  Overrides the default SQL query.

- **Source Filter**
  Reduces the number of rows the Data Integration Service queries.

- **Pre SQL**
  SQL commands to run against the source database before the Data Integration Service reads from the source.

- **Post SQL**
  SQL commands to run against the source database after the Data Integration Service writes to the target.

### Lookup Condition

The Data Integration Service finds data in the lookup source with a lookup condition.

The lookup condition is similar to the WHERE clause in an SQL query. 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:

```sql
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:

```sql
employee_ID > employee_number
```

The Data Integration Service returns rows for all employee_ID numbers greater than the source employee number.

### Rules and Guidelines for Lookup Transformation Conditions

Certain rules and guidelines apply when you enter a condition for a Lookup transformation.
Use 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.
- Enter a lookup condition in all Lookup transformations.
- Use one input port for each lookup port in the lookup condition. Use the same input port in more than one condition in a transformation.
- When you enter multiple conditions, the Data Integration Service evaluates each condition as an AND, not an OR. The Data Integration Service returns rows that match all the conditions you configure.
- If you include multiple conditions, enter the conditions in the following order to increase lookup performance:
  - Equal to (=)
  - Less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=)
  - Not equal to (!=)
- Use the following operators when you create the lookup condition:
  - =, >, <, >=, <=, !=
- The input value must meet all conditions for the lookup to return a value.
- The Data Integration Service matches null values. For example, if an input lookup condition column is NULL, the Data Integration Service evaluates the NULL equal to a NULL in the lookup.
Match Transformation

This chapter includes the following topics:
- Match Transformation Overview, 120
- Performing Duplicate Analysis, 121
- Match Transformation Processes, 121
- Match Transformation Components, 122
- Field Matching, 124
- Identity Matching, 127
- Match Mapplets, 130
- Configuring a Matching Operation, 131

Match Transformation Overview

The Match transformation is an active transformation that calculates the degree of similarity between input records and generates output data for these calculations.

The process of calculating similarity between records is called duplicate analysis. You can use the Match transformation to configure the following types of duplicate analysis:

Field matching
- Finds similar or duplicate data in two or more records.

Identity matching
- Finds identities shared by multiple records.

The following types of data project can require duplicate analysis:

CRM projects
- For example, a store plans to mail its customers and needs to check its customer database for duplicate customer records.

Mergers or acquisitions
- For example, a bank buys another bank in its region, and the two banks may have customers in common.

Regulatory compliance initiatives
- For example, a business may operate under government or industry regulations that insist that all data systems are free of duplicate records.
Note: Identity matching operations read reference data files called populations that define different types of identity. These files do not install with Informatica applications. You must download population data files and install them using the Data Quality Content Installer. Contact your Informatica Administrator user for information on the populations available on your system.

Performing Duplicate Analysis

The complexity of the duplicate analysis process depends on the size and structure of the data set and on the project objectives. It also depends on whether you select individual fields or identities.

You can define a process that uses the Match transformation alone or that uses several transformations. You can create one or more mapplets or mappings to accommodate all aspects of the process.

You can also embed match mapplets in a Match transformation. Match mapplets use Comparison and Weighted Average transformations in place of the Match transformation. You must validate a mapplet as a rule before you can embed it in the Match transformation.

You must include some or all of the following tasks in your process:

- Organize your datasets into groups of records. This reduces processing time for large datasets.
  - Use a Key Generator transformation to define these groups. The Key Generator creates a group key value for each record.
  - Use the Key Generator transformation to create sequence IDs if they are not present in your data.
- Select the columns to compare, and assign one or more duplicate analysis strategies to the columns. You can do this in a Match transformation or Comparison transformation. Use a Comparison transformation if you will create a matching rule that contains transformations for each matching operation. Use a Match transformation to define matching operations in a single transformation.
  - In field matching, you select each pair individually. In identity matching, the identity match strategy you select defines the field pairs you can use.
- Set the match threshold, and calculate a match score for your match strategy results. You can do this in a Match transformation or a Weighted Average transformation. Use the Weighted Average transformation if you create a matching mapplet.
- Select how you want to view the results of the match analysis. Select Clusters, Matched Pairs, or Best Match in the Match Output view.

Match Transformation Processes

To define a match process and apply it to input data, you select the type of matching operation to perform, configure the matching strategies, select the data columns to analyze, and define the output.

Consider the following points when you define a matching process:

- You can analyze the records in a single data set or across two data sets. The Match transformation enables this by creating two copies of each input column. Search for duplicates in a single data set by selecting the cloned copies of a column. Search for duplicates across two data sets by selecting unique columns from each data set. You can match multiple pairs in the Match transformation.
The Match transformation contains a set of comparison strategies that compare values in different ways. Select the fields to be compared and the type of strategy to apply to the fields.

Matching strategies compare every value in one column against every value in another column and compute the degree of similarity between each value pair as a number between zero and 1. This is the match score. The scores are decimal, and a score of 1 indicates that the values match perfectly.

Note: The Match transformation provides a wizard that you use to define strategies.

Every matching strategy you define generates match scores, which means that the transformation can generate multiple scores related to values in a single record. The transformation calculates an average match score that summarizes the degree of similarity between different records and allows you identify the records that are most similar to one another.

You can change how the transformation calculates the match score by applying numerical weight to the outputs from each column pair. For example, you may decide that duplicate family names are more relevant than duplicate first names and increase and reduce the match scores accordingly.

Use the transformation to set a match threshold for the match scores. The match threshold represents the minimum level of similarity needed to determine that two records are potential duplicates.

Match Transformation Components

Match transformation components include views and ports. Configure the views and ports to define a matching operation.

Match Transformation Views

Use the Match transformation views to configure a matching operation.

When you configure a matching operation, configure the following views:

Match Type

Configure the type of duplicate analysis the transformation performs. You can select field matching or identity matching for one or two data sets.

Strategies

Configure the data columns to match and select the match strategy to apply to the columns. You can also use this view to set match scores for null values and to set the priority of each column in the duplicate analysis.

Match Output

Configure the format for the duplicate analysis results.

Advanced

Configure the amount of detail the transformation writes to the session log.

Match Transformation Ports

The Match transformation contains predefined input and output ports for data relating to matching operations.

Input Ports

Match transformation input ports provide the data that the transformation requires for matching operations. After you create a Match transformation, you can configure the following input ports:
**SequenceId**

Provides an ID that uniquely identifies each record in the source data set. Use the Key Generator transformation to create unique IDs if none exist in the data set.

**GroupKey**

Provides the group key that the Match transformation uses to process records. Identity matching and field matching can use a group key. Ensure that the group key and sequence ID fields you select come from the same transformation.

**Note:** To improve matching speeds, configure both the GroupKey input port and the output port that connects to it with the same Precision value.

**Output Ports**

Match transformation output ports provide information about the duplicate analysis that the transformation performs. After you create a Match transformation, you can configure the following output ports:

- **ClusterId**
  The ID of the cluster to which the record belongs. Used in Clusters match output.

- **Group Key**
  The group key of the record.

- **ClusterSize**
  The number of records in the cluster to which a record belongs. Records that do not match with other records have a cluster size of 1. Used in Clusters match output.

- **RowId and RowId1**
  A unique row ID for the record. The Match transformation creates this ID. This ID may not match the row number in the input data.

- **DriverId**
  The row ID of the driver record in a cluster. The driver record is the final record added to the cluster. Used in Clusters match output.

- **DriverScore**
  The match score between a record and the driver record in its cluster.

- **LinkId**
  The row ID of the record that matched with the current record and linked it to the cluster. Used in Clusters match output.

- **LinkScore**
  The match score between two records that results in the creation of a cluster or in the addition of a record to a cluster. Used in Clusters match output.

**Note:** The matching process generates a single set of scores for each strategy that you define. The DriverScore and LinkScore values represent the degree of similarity between different pairs of records and provide you with different types of information.

The LinkScore defines the contents of the cluster. It must exceed the match threshold.

The DriverScore may be higher or lower than the LinkScore, and it may be lower than the match threshold.
Field Matching

Use field matching to find similar or duplicate data in two or more records.

Field matching operations compare the values from two data fields and calculates the similarity between them. When you configure the Match transformation for field matching, you select one or more pairs of columns from the input data.

Field Matching Strategies

The Match transformation includes predefined field matching strategies that compare pairs of data values.

Bigram

Use the Bigram strategy 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 and divides the number of matching character pairs by the total number of character pairs.

**Bigram Example**

Consider the following strings:

- Larder
- Lerder

These strings yield the following Bigram groups:

<table>
<thead>
<tr>
<th>L</th>
<th>a</th>
<th>r</th>
<th>d</th>
<th>e</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>e</td>
<td>r</td>
<td>d</td>
<td>e</td>
<td>r</td>
</tr>
</tbody>
</table>

The highlighted groups indicate matches between the input strings. Note that the second occurrence of the string "er" within the string "Lerder" is not matched, as there is no corresponding second occurrence of "er" 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 strategy 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
- Harlowes

The highlighted characters indicate the positions that the Hamming strategy 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 strategy 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
- Levensteiin

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 strategy 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 matching 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 strategy, 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 strategy considers character case when performing matching.

**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 strategy 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 strategy 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 strategy:

- 1-999-9999
- 011-01-999-9991

The highlighted characters indicate the positions that the Reverse Hamming Distance strategy 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 Matching Strategy Properties

Configure the properties for each field match strategy.

When you configure a field matching strategy, you can configure the following properties:

**Null Match**

Determines the match score when one input value is null.

**Both Null Match**

Determines the match score when both inputs are null.

Field Match Output Properties

Configure Match Output properties to control output for field matching operations.

When you configure field matching, you can configure the settings in the Match Output Type and Properties areas.

**Match Output Type settings**

Select one of the following match output types:

- **Clusters**
  
  Output match pairs in clusters.

- **Matched Pairs**
  
  Output all pairs that exceed the match threshold.

- **Best Match (Dual Source Matching)**
  
  Output the best match for each row in the master data set.

**Properties settings**

After you pick a match output type, you can configure the following properties:
Cache Directory

Specifies the directory that contains the field matching cache. The Data Integration service must be able to write to this directory. This directory must be present in the directory structure before the mapping runs.

Cache Size

Allocates the amount of system memory, in bytes, that the Match transformation can use to process matching operations. Whenever possible, the Match transformation uses system memory for processing. When the system memory cache is full, the transformation writes to the cache directory that you specify.

Threshold

Sets the minimum match score that the Match transformation uses to organize records into clusters. For a record to enter a cluster, it must link to at least one record in the cluster with a match score that meets or exceeds the threshold.

Scoring Method

Determines the match score types that appear in transformation output. Used for match cluster outputs only. The default setting for this property is Link Score. If you choose either Driver Score or Both, you may significantly increase processing time.

Dual Source Field Matching Properties

For dual source field matching operations, configure the Master Data Set property on the Match Type view.

When you use two data sources, you must select one data source as the master data set. If neither data source is a master data set, it does not matter which data source you select as the master.

Identity Matching

Use identity matching to find identities shared by multiple records.

An identity is a set of data values within a record that collectively provide enough information to identify a person or organization.

Some identity matches indicate record duplication. Other identity matches indicate a connection between records, such as a shared family identity or a shared employer identity. When you configure the Match transformation for identity matching, you can select as many pairs of columns as the match strategy allows.

Identity Match Type Properties

Configure identity match type properties to analyze records for matching identities.

When you configure identity match types, you can configure the following properties.

Population

Determines the reference data file that contains key-building algorithms customized for locales and languages.

Key Level

Determines the number of keys generated by identity matching algorithms. 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 category of data contained in the key field. Identity matching can generate keys for person names, organizations, and addresses. Select the key type that best describes the column you choose for the Key Field property.

Search Level

Indicates the balance of search quality and search speed. The search speed is inversely related 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 group keys. Verify that the column you select contains the kind of data you specified in the Key Type option.

Index Folder

Specifies the directory in which the Match transformation writes the identity key index. The Data Integration service must be able to write to this directory.

Note: This property specifies the path to the parent directory for the Index Folder. The identity matching process creates a directory at this path, writes the index files, and deletes the directory when the mapping finishes.

Cache Directory

Specifies the directory that contains the identity matching cache. The Data Integration service must be able to write to this directory.

Note: This directory must be present in the directory structure before the mapping runs.

Master Data Set

Specifies the data source that contains the master data. Used in dual source matching.

Identity Matching Strategies

The Match transformation includes predefined identity matching strategies that compare pairs of data values.

The following table describes match operations and required inputs for each identity matching strategy:

<table>
<thead>
<tr>
<th>Identity Match Strategy</th>
<th>Match Operation</th>
<th>Required Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Identifies records that share an address.</td>
<td>Address</td>
</tr>
<tr>
<td>Contact</td>
<td>Identifies records that share a contact at a single organization location.</td>
<td>Person_Name Organization_Name Address_Part1</td>
</tr>
<tr>
<td>Corp Entity</td>
<td>Identifies records that share organization data. Optionally, configure this strategy to analyze address and telephone data.</td>
<td>Organization_Name</td>
</tr>
<tr>
<td>Division</td>
<td>Identifies records that share an organization office at a specific address.</td>
<td>Organization_Name Address_Part1</td>
</tr>
<tr>
<td>Family</td>
<td>Identifies individuals that belong to a family by analyzing name, address, and telephone number data.</td>
<td>Person_Name Address_Part1 Telephone Number</td>
</tr>
<tr>
<td>Identity Match Strategy</td>
<td>Match Operation</td>
<td>Required Inputs</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Fields</td>
<td>Identifies records that share data for custom fields that you select.</td>
<td>N/A</td>
</tr>
<tr>
<td>Household</td>
<td>Identifies individuals that belong to a household by analyzing name data and address data.</td>
<td>Person_Name Address_Part1</td>
</tr>
<tr>
<td>Individual</td>
<td>Identifies duplicate individuals by analyzing name, date of birth, and ID data. ID examples include Social Security numbers, account numbers, and vehicle identification numbers.</td>
<td>Person_Name ID Date</td>
</tr>
<tr>
<td>Organization</td>
<td>Identifies records that share organization data.</td>
<td>Organization_Name</td>
</tr>
<tr>
<td>Person Name</td>
<td>Identifies duplicate individuals by analyzing name data.</td>
<td>Person_Name</td>
</tr>
<tr>
<td>Resident</td>
<td>Identifies duplicate individuals at an address. Optionally, configure this strategy to analyze ID information.</td>
<td>Person_Name Address_Part1</td>
</tr>
<tr>
<td>Wide Contact</td>
<td>Identifies records that share a contact at an organization.</td>
<td>Person_Name Organization_Name</td>
</tr>
<tr>
<td>Wide Household</td>
<td>Identifies individuals that belong the same household.</td>
<td>Address_Part1</td>
</tr>
</tbody>
</table>

**Identity Matching Strategy Properties**

Configure the properties for each identity strategy.

When you configure identity matching strategies, you can configure the following strategy properties:

**Population**

Determines the population to apply to identity matching. 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 Properties**

Configure **Match Output** properties to control output for identity matching operations.

When you configure identity matching, you can configure the settings in the **Match output type** and **Properties** areas:

**Match Output Type settings**

Select one of the following match output types:
Clusters
Output identity match clusters.

Matched Pairs
Output all pairs that exceed the match threshold.

Best Match (Dual Source Matching Only)
Output the best match for each row in the master data set.

Properties settings
After you pick a match output type, you can configure the following properties:

Cache Directory
Determines the directory where the Match transformation writes temporary files.

Cache Size
Allocates the amount of system memory, in bytes, that the Match transformation can use to process matching operations. Whenever possible, the Match transformation uses system memory for processing. When the system memory cache is full, the transformation writes to the cache directory that you specify.

Threshold
Sets the minimum match score that the Match transformation uses to organize records into clusters. For a record to enter a cluster, it must link to at least one record in the cluster with a match score that meets or exceeds the threshold.

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 matching processes. 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 reuse matching operations in multiple Match transformations and mappings.

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, matching 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 mapplet in the repository.

   **Important:** You can only select mapplets created by the Match transformation.

   The **Select Match Mapplet** window opens.

5. Select a match mapplet 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 Matching Operation

To configure a matching 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 matching process 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, Null Match, and Both 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 mapplet in the **Strategies** view. A match mapplet is a type of mapplet that you can embed in a Match transformation.
Merge Transformation

This chapter includes the following topics:
- Merge Transformation Overview, 133
- Configuring a Merge Strategy, 133

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.
This chapter includes the following topics:
- Parser Transformation Overview, 134
- Designing a Parser Transformation, 134
- Token Parsing Mode, 135
- Pattern-Based Parsing Mode, 137

Parser Transformation Overview

The Parser transformation is a passive transformation that creates new fields for specific information categories and parses input data into these fields.

The Parser transformation parses data according to the following categories:

**Parsed Data**
- Matches a user-defined token or pattern.

**Overflow Data**
- Matches a user-defined token or pattern, but the transformation cannot assign the data to an output.

**Unparsed Data**
- Does not match a user-defined token or pattern.

Designing a Parser Transformation

When you create a Parser transformation, select either token parsing mode or pattern-based parsing mode.

- **Token parsing mode.** Use this mode to create strategies that parse individual strings that match token sets, regular expressions, or reference table entries. You can use multiple token parsing strategies in a transformation.

- **Pattern-based parsing mode.** Use this mode to parse patterns made of multiple strings. You can enter custom patterns or configure the transformation to read patterns from reference tables. In this mode, the Parser transformation requires the output from a Labeler transformation as input.
Token Parsing Mode

In token parsing mode, the Parser transformation parses strings that match token sets, regular expressions, 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 definitions to parse input data. You can select from predefined token sets included with Informatica Developer. Example token sets include words, ZIP codes, and phone numbers. You can also define custom regular expressions.

**Parse using Reference Table**

Parse input data using reference table entries.

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.

**Unparsed**

Contains strings that the transformation cannot parse successfully. The Parser transformation creates an unparsed port for each strategy that you add.
Token Parsing Properties

Token parsing properties include strategy properties and operation properties. Configure these properties on the Strategies view.

Strategy Properties

Strategy properties apply to all the operations within a strategy. You can configure the following strategy properties:

- **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 this string as a whole.
- **Output Join Character.** Specifies the character used to join output data when multiple parsing operations use the same output.
- **Reverse Enabled.** Configures the strategy to parse data from right to left.
- **Overflow Reverse Enabled.** Configures the strategy to parse the overflow data from right to left.
- **Detailed Overflow Enabled.** Creates a unique overflow field for each parsing operation.
- **Delimiters.** Determines the delimiters that separate the input data into separate tokens. If you do not select a delimiter, the strategy uses the space delimiter by default.

Operation Properties

Operation properties apply to a single operation. You can configure the properties for the following types of operations:

Token Set Operations

Select the Parse using Token Set operation to parse input with token sets. You can set the following properties for token definition operations:

- **Fixed Token Sets (Single Output Only).** Specifies the token sets the operation uses for parsing.
- **Custom Regular Expression (Single or Multiple Outputs).** Defines a custom regular expression that you enter.
- **Outputs.** Determines the outputs for parsed data.

Reference Table Operations

Select the Parse using Reference Table operation to parse input with reference tables. You can set the following properties for reference table operations:

- **Reference table.** Specifies the reference table the operation uses for parsing.
- **Case sensitive.** Sets the operation to compare input data to the reference data in a case-sensitive manner.
- **Standardize.** Replace parsed data with the "Valid" value from the reference table.
- **Outputs.** Determines the outputs for parsed data.

Configure a Token Parsing Strategy

To configure a token parsing strategy, open a Parser transformation that uses the 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.

Pattern-Based Parsing Mode

In pattern-based parsing mode, the Parser transformation parses patterns made of multiple strings.

The following list describes the methods you can use to parse data in pattern-based parsing mode:

- Reference tables. 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.
- User-defined patterns. Parse input data using patterns that you define.

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 the 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.

Configure a Parsing Pattern

To configure a parsing pattern, open a Parser transformation that uses the pattern parsing mode and select the Patterns view.

Before you configure parsing patterns, add output ports to the transformation to handle successfully parsed data.

1. On the Patterns view, click New and select New Pattern or New Reference Table.
2. If you added a pattern, double-click the "Enter Patterns Here" text in the Patterns column and enter a custom pattern.
3. If you added a reference table, expand the reference table and select a row.
4. Configure the transformation to parse tokens to outputs.
   ♦ To parse a single token to an output, select the pattern row and click in the column for that output. Select a pattern to parse to that output.
   ♦ To parse multiple tokens to a single output, click in the column for that output and select Custom. Select one or more delimiters for the output column if you want the output tokens to be delimited.
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 have a database table that contains information about product orders. You need to create invoice for the items ordered. Use a Sorter transformation on the product orders table to sort the data in descending order according to the order ID. 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.

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 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:

**Distinct**
- Treats output rows as distinct. 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 compared during the sort operation.

**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**
- Determines the maximum amount of memory required to perform the sort operation. The Data Integration Service passes all incoming data into the Sorter transformation before it performs the sort operation. The default value for the Sorter Cache Size property is Auto. You can configure a numeric value for the Sorter cache.

**Work Directory**
- The Data Integration Service uses the work directory to create temporary files while it sorts data. After the Data Integration Service sorts data, it deletes the temporary files. You can specify any directory on the Data Integration Service machine to use as a work directory. The following directory is the default value represented by a period:
  
  `<Informatica Services Installation Directory>\tomcat\bin`

**Tracing Level**
- The amount of detail displayed in the log for the Sorter transformation.

**Sorter Cache Size**

The Data Integration Service passes all incoming data into the Sorter transformation before it performs the sort operation.
If you set the Sorter Cache Size to Auto, the Data Integration Service determines the cache size at run time. Before starting the sort operation, the Data Integration Service allocates the amount of memory configured for the Sorter cache size. If the Data Integration Service cannot allocate enough memory, the mapping fails.

For best performance, configure Sorter cache size with a value less than or equal to the amount of available physical RAM on the machine that hosts Data Integration Service. Allocate at least 16 MB (16,777,216 bytes) of physical memory to sort data using a Sorter transformation. Sorter cache size is set to Auto by default.

If the amount of incoming data is greater than the amount of 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. If the amount of incoming data is significantly greater than the Sorter cache size, the Data Integration Service may require more than twice the amount of disk space available to the work directory.

Use the following formula to determine the size of incoming data:

\[(\text{number of input rows} \times (\text{column size}) + 16)\]

The following table describes the column size values by datatype for Sorter data calculations:

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Column Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>precision + 8</td>
</tr>
<tr>
<td></td>
<td>Round to nearest multiple of 8</td>
</tr>
<tr>
<td>Date/Time</td>
<td>29</td>
</tr>
<tr>
<td>Decimal, high precision off (all precision)</td>
<td>16</td>
</tr>
<tr>
<td>Decimal, high precision on (precision &lt;=18)</td>
<td>24</td>
</tr>
<tr>
<td>Decimal, high precision on (precision &gt;18, &lt;=28)</td>
<td>32</td>
</tr>
<tr>
<td>Decimal, high precision on (precision &gt;28)</td>
<td>16</td>
</tr>
<tr>
<td>Decimal, high precision on (negative scale)</td>
<td>16</td>
</tr>
<tr>
<td>Double</td>
<td>16</td>
</tr>
<tr>
<td>Real</td>
<td>16</td>
</tr>
<tr>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>String, Text</td>
<td>Unicode mode: 2*(precision + 5)</td>
</tr>
<tr>
<td></td>
<td>ASCII mode: precision + 9</td>
</tr>
</tbody>
</table>

**Sorter Transformation Ports**

All ports of the Sorter transformation allows you to input data, sort, and output data to other transformations. The Sorter transformation has the following port types:
Key

The sort key is one or more ports that you want to use as the sort criteria. The Data Integration Service sorts data according to the ports specified as key. Configure the Direction of the Key port to sort data in ascending or descending order. The Developer tool sets the default Direction value as ascending.

Other

The ports that are not specified as sort key.

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.
   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. Select Key to indicate the port as sort key.
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 mapplets.

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. In the Ports view, 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 view and edit the transformation properties.
You have a database table PRODUCT_ORDERS that contains information about all the orders which were placed by the customer.

<table>
<thead>
<tr>
<th>ORDER_ID</th>
<th>ITEM_ID</th>
<th>ITEM</th>
<th>QUANTITY</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>123456</td>
<td>ItemA</td>
<td>3</td>
<td>3.04</td>
</tr>
<tr>
<td>41</td>
<td>456789</td>
<td>ItemB</td>
<td>2</td>
<td>12.02</td>
</tr>
<tr>
<td>43</td>
<td>000246</td>
<td>ItemC</td>
<td>6</td>
<td>34.55</td>
</tr>
<tr>
<td>45</td>
<td>000468</td>
<td>ItemD</td>
<td>5</td>
<td>0.56</td>
</tr>
<tr>
<td>41</td>
<td>123456</td>
<td>ItemA</td>
<td>4</td>
<td>3.04</td>
</tr>
<tr>
<td>45</td>
<td>456789</td>
<td>ItemA</td>
<td>5</td>
<td>3.04</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>ORDER_ID</th>
<th>ITEM_ID</th>
<th>ITEM</th>
<th>QUANTITY</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>000468</td>
<td>ItemD</td>
<td>5</td>
<td>0.56</td>
</tr>
<tr>
<td>45</td>
<td>123456</td>
<td>ItemA</td>
<td>5</td>
<td>3.04</td>
</tr>
<tr>
<td>45</td>
<td>456789</td>
<td>ItemB</td>
<td>3</td>
<td>12.02</td>
</tr>
<tr>
<td>43</td>
<td>123456</td>
<td>ItemA</td>
<td>3</td>
<td>3.04</td>
</tr>
<tr>
<td>43</td>
<td>000246</td>
<td>ItemC</td>
<td>6</td>
<td>34.55</td>
</tr>
<tr>
<td>41</td>
<td>456789</td>
<td>ItemB</td>
<td>2</td>
<td>12.02</td>
</tr>
<tr>
<td>41</td>
<td>123456</td>
<td>ItemA</td>
<td>4</td>
<td>3.04</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>ORDER_ID</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>54.06</td>
</tr>
<tr>
<td>43</td>
<td>216.42</td>
</tr>
<tr>
<td>41</td>
<td>36.2</td>
</tr>
</tbody>
</table>
CHAPTER 23

SQL Transformation

This chapter includes the following topics:
- SQL Transformation Overview, 144
- Advanced Properties, 144
- SQL Transformation Ports, 145
- SQL Transformation Query, 148
- Input Row to Output Row Cardinality, 149
- SQL Transformation Example, 152

SQL Transformation Overview

The SQL transformation processes SQL queries midstream in a mapping. You 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 the mapping.

An SQL transformation runs an SQL query that you define in the transformation SQL Editor. The SQL transformation processes the query, returns rows, and returns any database error. You can pass input port values to parameters in the query.

The SQL transformation is an active transformation. The transformation can return multiple rows for each input row.

When you configure an SQL transformation perform the following tasks:
1. Define the transformation properties including the database type to connect to.
2. Define the ports.
3. Create an SQL query in the SQL Editor.
4. Configure the SQL transformation in a mapping, connecting upstream ports.
5. Preview the data to verify the results.

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.
The following SQL transformation properties appear on the Advanced Properties view:

**Tracing Level**
Sets the amount of detail included in the mapping log when you run a mapping containing the SQL transformation. 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.

**Database Type**
Type of database that the SQL transformation connects to. Choose a database type from the list. You can choose Oracle, Microsoft SQL Server, IBM DB2, or ODBC. The database type affects the datatypes that you can assign on the Ports view. 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.

---

**SQL Transformation Ports**

When you create an SQL transformation, the Developer tool creates the SQLError 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.

SQL Error

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.

Input Ports

You can create input ports in the SQL transformation for data that you do not intend to pass to output ports. You can reference input ports in a SQL query and include port names as a parameters in the query.

To add input ports, click Input in the Ports view. Click New.

When you add the port, enter the native datatype for the port. The native datatype is a datatype that is valid for the database 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.

Note: If you select Copy to Output for a port, the input port becomes a pass-through port.

Output Ports

Output ports return values from a SELECT statement. Create an output port for each column in the SELECT statement.

When you configure an output port, choose the native datatype for the port. The native datatype of an output port 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:

<table>
<thead>
<tr>
<th>Output Port</th>
<th>Native Datatype</th>
<th>Transformation Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstNm</td>
<td>varchar2</td>
<td>string</td>
</tr>
<tr>
<td>LastNm</td>
<td>varchar2</td>
<td>string</td>
</tr>
<tr>
<td>Age</td>
<td>number</td>
<td>double</td>
</tr>
</tbody>
</table>

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 SELECT statement returns. When the number of output ports is more than the number of columns in the SELECT clause, the extra ports return a null value. When the number of output ports is less than the number of columns in the SELECT clause, 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.
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 **Ports** 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.

### SQLError Port

The SQLError port returns SQL errors from the database.

When the SQL query contains syntax errors, the error port contains the error text from the database. For example, the following SQL query generates a row error from an Oracle database:

```sql
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-00944: "Product_ID": invalid identifier
```

You can configure multiple query statements in the SQL query. 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 statements change for each input row. 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:

```sql
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 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. 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 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:

```sql
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:

```sql
SELECT Name, Address FROM Employees WHERE Employee_Num = ?Employee_ID? and Dept = ?Dept?
```

The source might have the following rows:

<table>
<thead>
<tr>
<th>Employee_ID</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Products</td>
</tr>
<tr>
<td>123</td>
<td>HR</td>
</tr>
<tr>
<td>130</td>
<td>Accounting</td>
</tr>
</tbody>
</table>
The Data Integration Service generates the following query statements from the rows:

```sql
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~:

```sql
SELECT Emp_ID, Address from ~Table_Port~ where Dept = 'HR'
```

The source might pass the following values to the **Table_Port** column:

<table>
<thead>
<tr>
<th>Table_Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employees_USA</td>
</tr>
<tr>
<td>Employees_England</td>
</tr>
<tr>
<td>Employees_Australia</td>
</tr>
</tbody>
</table>

The Data Integration Service replaces the ~Table_Port~ variable with the table name value in the input port:

```sql
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:

<table>
<thead>
<tr>
<th>Query Statement</th>
<th>Output Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE, INSERT, DELETE only</td>
<td>One row for each statement in the query.</td>
</tr>
<tr>
<td>One or more SELECT statements</td>
<td>Total number of database rows retrieved.</td>
</tr>
<tr>
<td>DDL queries such as CREATE, DROP, TRUNCATE</td>
<td>One row for each statement in the query.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Query Statement</th>
<th>Output Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE, INSERT, DELETE only</td>
<td>One row for each statement with the NumRowsAffected for the statement.</td>
</tr>
<tr>
<td>One or more SELECT statements</td>
<td>Total number of database rows retrieved. NumRowsAffected is zero in each row.</td>
</tr>
<tr>
<td>DDL queries such as CREATE, DROP, TRUNCATE</td>
<td>One row with zero NumRowsAffected.</td>
</tr>
</tbody>
</table>

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:

```sql
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:

<table>
<thead>
<tr>
<th>NumRowsAffected Port or Pass-Through Port Configured</th>
<th>SQLError</th>
<th>Rows Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neither port configured</td>
<td>No</td>
<td>One row with NULL in the SQLError port.</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>One row with the error in the SQLError port.</td>
</tr>
<tr>
<td>Either port configured</td>
<td>No</td>
<td>One row for each query statement with the NumRowsAffected or the pass-through column data.</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>One row with the error in the SQLError port, the NumRowsAffected port, or the pass-through port data.</td>
</tr>
</tbody>
</table>

The following table describes the number of output rows that the SQL transformation generates for SELECT statements:

<table>
<thead>
<tr>
<th>NumRowsAffected Port or Pass-Through Port Configured</th>
<th>SQLError</th>
<th>Rows Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neither port configured</td>
<td>No</td>
<td>One or more rows, based on the rows returned from each SELECT statement.</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>One row greater than the sum of the output rows for the successful statements. The last row contains the error in the SQLError port.</td>
</tr>
</tbody>
</table>
| Either port configured                             | No       | One or more rows, based on the rows returned for each SELECT statement:
  - 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. |
|                                                   | Yes      | One or more rows, based on the rows returned for each SELECT statement. The last row contains the error in the SQLError port:
  - 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:

<table>
<thead>
<tr>
<th>NumRowsAffected Port or Pass-Through Port Configured</th>
<th>SQLError</th>
<th>Rows Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neither port configured</td>
<td>No</td>
<td>One row with NULL in the SQLError port.</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>One row that contains the error in the SQLError port.</td>
</tr>
<tr>
<td>Either port configured</td>
<td>No</td>
<td>One row that includes the NumRowsAffected column with value zero and the pass-through column data.</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>One row with the error in the SQLError port, the NumRowsAffected column with value zero, and the pass-through column data.</td>
</tr>
</tbody>
</table>

**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:

```sql
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.

**SQL Transformation Example**

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:

<table>
<thead>
<tr>
<th>Employee_Num</th>
<th>HireDate</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3-May-97</td>
<td>232000</td>
</tr>
<tr>
<td>11</td>
<td>11-Sep-01</td>
<td>444000</td>
</tr>
<tr>
<td>12</td>
<td>17-Oct-89</td>
<td>656000</td>
</tr>
<tr>
<td>13</td>
<td>13-Aug-07</td>
<td>332100</td>
</tr>
</tbody>
</table>

Employee Table

The source is the Employee table from the Human Resources database.

The following table shows sample rows from the Employee table:

<table>
<thead>
<tr>
<th>EmpID</th>
<th>LastName</th>
<th>FirstName</th>
<th>DeptId</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Smith</td>
<td>Martha</td>
<td>FIN</td>
<td>(415) 552-1623</td>
</tr>
<tr>
<td>11</td>
<td>Jones</td>
<td>Cynthia</td>
<td>ENG</td>
<td>(415) 552-1744</td>
</tr>
<tr>
<td>12</td>
<td>Russell</td>
<td>Cissy</td>
<td>SLS</td>
<td>(415) 552-1656</td>
</tr>
<tr>
<td>13</td>
<td>Goyal</td>
<td>Girish</td>
<td>FIN</td>
<td>(415) 552-1656</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Native Type</th>
<th>Precision</th>
<th>Scale</th>
<th>Copy to Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpID</td>
<td>decimal</td>
<td>number(p,s)</td>
<td>4</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>LastName</td>
<td>string</td>
<td>varchar2</td>
<td>30</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>FirstName</td>
<td>string</td>
<td>varchar2</td>
<td>20</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>DeptID</td>
<td>string</td>
<td>varchar2</td>
<td>4</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>Phone</td>
<td>string</td>
<td>varchar2</td>
<td>16</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

The SQL transformation has the following output ports:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Native Type</th>
<th>Precision</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpID</td>
<td>decimal</td>
<td>Number(p,s)</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
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:

```sql
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:
<table>
<thead>
<tr>
<th>EmpId</th>
<th>LastName</th>
<th>FirstName</th>
<th>DeptId</th>
<th>Phone</th>
<th>HireDate</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Russell</td>
<td>Cissy</td>
<td>SLS</td>
<td>(415) 552-1656</td>
<td>19891017 00:00:00</td>
<td>6560.00</td>
</tr>
<tr>
<td>13</td>
<td>Goyal</td>
<td>Girish</td>
<td>FIN</td>
<td>(415) 552-1660</td>
<td>20070813 00:00:00</td>
<td>3210.00</td>
</tr>
</tbody>
</table>
Standardizer Transformation

This chapter includes the following topics:

- Standardizer Transformation Overview, 157
- Standardization Strategies, 157
- Standardization Properties, 158
- Configuring a Standardization Strategy, 159

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." If you do not select a delimiter, 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.
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**.

- **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.
CHAPTER 25

Union Transformation

This chapter includes the following topics:
• Union Transformation Overview, 160
• Groups and Ports, 160
• Union Transformation Advanced Properties, 161
• Union Transformation Processing, 161
• Creating a Union Transformation, 162

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, customer account data from American Bank resides in an Oracle database, and customer account data from California Bank resides in an IBM DB2 database. You want to create a single view of customer account data. Import a logical data object model that defines the relationship between customers and accounts. The model contains an Account logical data object with attributes for accounts and customers. You then create a logical data object read mapping for the Account data object. The mapping contains a Union transformation to merge data from the source objects and create a single view of customer account data.

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.

Configure the following property on the Advanced tab:

**Tracing Level**

Amount of detail displayed in the log for this transformation.

Default is normal.

The following table describes the tracing levels:

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terse</td>
<td>Logs initialization information and error messages and notification of rejected data.</td>
</tr>
<tr>
<td>Normal</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose Initialization</td>
<td>In addition to normal tracing, logs additional initialization details, names of index and data files used, and detailed transformation statistics.</td>
</tr>
<tr>
<td>Verbose Data</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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.
   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. In the General view, 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.
Chapter 26

Weighted Average Transformation

This chapter includes the following topic:

- Weighted Average Transformation Overview, 163

Weighted Average Transformation Overview

The Weighted Average transformation is a passive transformation that reads match scores from matching operations, weights the scores, and produces an aggregate match score.

A weight is a percentage value. You can edit the weight applied to each input score to increase or decrease the percentage of its contribution to the aggregate score. The output score from the Weighted Average transformation reflects the relative importance of each data field in the duplicate analysis.

To assign weights to input match scores, click an input port in the Weighted Average transformation and edit the value in the Weight column. You can choose values ranging from 0.001 to 1.0.

Use the Weighted Average transformation when you have configured Comparison transformations to apply two or more match strategies to the records in a data set. The Weighted Average transformation calculates an aggregate match score for each pair of records, so that you can group duplicate records for further processing.

Note: You can also assign weights in a Match transformation. Use the Weighted Average transformation with Comparison transformations in a matching mapplet. 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.

Weighting Match Scores Example

You are using the Match transformation to generate match scores for the ZIP code and Surname columns in a customer 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 set Weight value of the Surname input to 0.8, and set the value of the ZIP code input to 0.4.
Transformation Delimiters

This appendix includes the following topic:

- Transformation Delimiters Overview, 164

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:

<table>
<thead>
<tr>
<th>Delimiter Name</th>
<th>Delimiter Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>at symbol</td>
<td>@</td>
</tr>
<tr>
<td>comma</td>
<td>.</td>
</tr>
<tr>
<td>dash</td>
<td>-</td>
</tr>
<tr>
<td>double quote</td>
<td>&quot;</td>
</tr>
<tr>
<td>forward slash</td>
<td>/</td>
</tr>
<tr>
<td>full stop</td>
<td>.</td>
</tr>
<tr>
<td>hash</td>
<td>#</td>
</tr>
<tr>
<td>pipe</td>
<td></td>
</tr>
<tr>
<td>semi-colon</td>
<td>:</td>
</tr>
<tr>
<td>single quote</td>
<td>'</td>
</tr>
<tr>
<td>space</td>
<td>[Spacebar]</td>
</tr>
<tr>
<td>tab</td>
<td>[Tab key]</td>
</tr>
<tr>
<td>underscore</td>
<td>_</td>
</tr>
</tbody>
</table>
INDEX

A
active transformations
description 1
Java 60, 61
advanced interface
EDataType class 89
element 91
invoking Java expressions 88
Java expressions 88
JExpression class 90, 92
JExprParaMetadata class 89
advanced properties
Java transformations 64
Advanced Properties
SQL transformation 145
Advanced view
Custom Data transformation 45
Aggregator transformation
advanced properties 24
aggregate caches 25
aggregate expressions 26
aggregate functions 26
creating non-reusable 30
creating reusable 29
developing 24
group by ports 27
nested aggregate functions 26
non-aggregate expressions 28
overview 23
ports 24
sorted input 28
sorting data 28
tips 30
troubleshooting 30
API methods
Java transformation 75
Java transformations 75
At End tab
Java transformations 71

B
buffer port
Custom Data transformation 47

C
classpath
mapping property 66
code snippets
creating for Java transformations 68
compilation errors
identifying the source of in Java transformations 74

compiling
Java transformations 73
compiling Java code
Full Code tab 71
connected transformations
Java transformations 63
considerations
Java transformations 63
Continue on SQL Error
SQL transformation 149, 152
creating
Java transformations 72
Custom Data transformation
adding ports 47
additional ports 42
Advanced View 45
configuring a service 43
creating 48
creating ports from a service 48
default ports 42
exporting to XML schema 44
overview 41
previewing output data 48
status tracing messages 46
types of ports 47
views 42
writing to relational targets 44

D
Data Transformation
creating ports from 48
description 41
repository location 42
Data Transformation Studio
description 41
datatypes
Java transformations 62
defineJExpression method
Java transformations 76, 90
designing
Java transformations 63

E
EDataType class
Java expressions 89
editor views
non-reusable Java transformations 61
reusable Java transformations 61
error counts
incrementing for Java transformations 78
errors
increasing threshold in Java transformations 78
Java packages
importing 68
Java primitive datatypes
Java transformations 62
Java transformation
API methods 75
overview 60
Java transformations
active 61
advanced properties 64
API methods 75
At End tab 71
checking null values in 80
compilation errors 73
compiling 73
creating 72
creating Java code snippets 68
creating ports 63
datatype conversion 62
default port values 63
defineJExpression method 76, 90
designing 63
failing mappings in 77
failSession method 77
Full Code tab 71
Function tab 71
generateRow method 77
getInRowType method 78
getting the input row type 78
Helpers tab 69, 70
high precision processing 64
identifying the source of compilation errors 74
Imports tab 68, 70
incrementErrorCount method 78
input ports 64
invokeJExpression method 79, 86
isNull method 80
Java code 66
Java primitive datatypes 62
logError method 80
logInfo method 81
logs 81
mapping-level classpath 66
nanosecond processing 64
non-reusable 61
non-user code errors 74
On Inputs tab 70
output ports 64
passive 61
reusable 61
session logs 80
setQuery method 81
setting null values in 81
troubleshooting 73
user code errors 74
JDK
Java transformation 60
JExpression class
Java expressions 90, 92
JExprParaMetadata class
Java expressions 89
Joiner transformation
advanced properties 95
blocking source pipelines 102
caching master rows 103
configuring sort order 100
detail outer join 98
full outer join 99
joining data from the same source 101
master outer join 98
normal join 97
overview 95
performance 103
ports 97
rules and guidelines 104
sorted 103
sorted input 99
unsorted 102
Joiner transformations
join types 97
JRE
Java transformation 60
logError method
Java transformations 80
logInfo method
Java transformations 81
logs
Java transformations 81
Lookup transformation
advanced properties for flat file lookups 117
advanced properties for relational lookups 117
developing 115
lookup condition 118
lookup condition rules and guidelines 119
overview 115
runtime properties 116
mapper
Data Transformation 43
Max Output Row Count
SQL transformation 149
Maximum Output Row Count
SQL transformation 150
methods
Java transformation API 75
nanosecond processing
Java transformations 64
non-user code errors
in Java transformations 74
null values
checking in Java transformations 80
setting for Java transformation 81
Number of Rows Affected
SQL transformation 147
NumRowsAffected
rows output 151
On Inputs tab
Java transformations 70
output ports
Java transformation 63
Java transformations 64
Overview

Custom Data transformation 42

P
parameter binding
SQL transformation 148
parser
Data Transformation 43
pass-through ports
SQL transformation 147
passive transformations
description 1
Java 60, 61
port values
Java transformations 63
ports
Custom Data transformation 47
Java transformations 63
Ports view
SQL transformation output 146
preview data
Custom Data transformation 48

R
reusable transformations
description 3
editing 4
Runtime view
SQL transformation 155

S
serializer
Data Transformation 43
Service View
Custom Data transformation 43
services
Data Transformation 43
session logs
Java transformations 80
setNull method
Java transformations 81
simple interface
description 87
Java expressions 86
Java transformation API methods 86
SQL input ports
SQL transformation 146
SQL query
SQL transformation 148
SQL transformation
Advanced Properties view 145
continuing on SQL error 152
defining output ports 146
defining the database connection 155
defining the query 148
example 152
input ports description 146
input row to output row cardinality 149
number of rows affected 147
number of rows output 151
parameter binding 148
pass-through ports 147
ports 145
query statement 155
query string substitution 149
restricting output rows 150
SQLException port 147
SQL transformation
SQL port
SQL transformation 147
static code
Java transformations 69
static variables
Java transformations 69
status tracing message
Custom Data transformation 46
streamer
Data Transformation 43
string substitution
SQL transformation 149
Structure view
Custom Data transformation 44

T
transformations
active 1
creating 6
developing 3
editing reusable 4
expression validation 6
expressions 4
Java 60
overview 1
reusable 3
transformer
Data Transformation 43
troubleshooting
Java transformations 73

U
user code errors
Java transformations 74
user-defined methods
Java transformations 69

V
variables
Java transformations 69